CRANFIELD UNIVERSITY

MASOOD RAZA

COMMAND AGENTS WITH HUMAN-LIKE DECISION MAKING STRATEGIES

CRANFIELD DEFENCE AND SECURITY

PhD THESIS

CRANFIELD UNIVERSITY

CRANFIELD DEFENCE AND SECURITY

DEPARTMENT OF ENGINEERING SYSTEMS AND MANAGEMENT

PhD THESIS

Academic Year 2004-2008

Masood Raza

Command agents with human-like decision making strategies

Supervisor: Dr V V S S Sastry

July 2009

ABSTRACT

Human behaviour representation in military simulations is not sufficiently realistic, specially the decision making by synthetic military commanders. The decision making process lacks realistic representation of variability, flexibility, and adaptability exhibited by a single entity across various episodes. It is hypothesized that a widely accepted naturalistic decision model, suitable for military or other domains with high stakes, time stress, dynamic and uncertain environments, based on an equally tested cognitive architecture can address some of these deficiencies. And therefore, we have developed a computer implementation of Recognition Primed Decision Making (RPD) model using Soar cognitive architecture and it is referred to as RPD-Soar agent in this report. Due to the ability of the RPD-Soar agent to mentally simulate applicable courses of action it is possible for the agent to handle new situations very effectively using its prior knowledge.

The proposed implementation is evaluated using prototypical scenarios arising in command decision making in tactical situations. These experiments are aimed at testing the RPD-Soar agent in recognising a situation in a changing context, changing its decision making strategy with experience, behavioural variability within and across individuals, and learning. The results clearly demonstrate the ability of the model to improve realism in representing human decision making behaviour by exhibiting the ability to recognise a situation in a changing context, handle new situations effectively, flexibility in the decision making process, variability within and across individuals, and adaptability. The observed variability in the implemented model is due to the ability of the agent to select a course of action from reasonable but some times sub-optimal choices available. RPD-Soar agent adapts by using 'chunking' process which is a form of explanation based learning provided by Soar architecture. The agent adapts to enhance its experience and thus improve its efficiency to represent expertise.

ACKNOWLEDGEMENT

I thank my advisor, *Dr Venkat Sastry*, and my thesis committee members *Mr Mike Bathe* and *Dr Ken McNaught* for guiding me throughout my research and I specially thank them for putting up with me during the update meetings because I definitely know how fruitless those meetings would have been for them. I also thank *Dr Frank Ritter* for guiding me through out on all aspects of my research work and *Professor John Laird* for his support in solving the problems related to *Soar* and I thank both of them again for giving me permission to use diagrams related to *Soar* from their work in this thesis. I also appreciate the efforts put in by our department in keeping all the administrative tasks smooth through out the period of this research and I specially thank *Ros* and *Paula* for providing me all the support that I desperately needed in sorting out the administrative requirements related to funding and arranging my visits abroad and in country to attend the conferences to present my research work. I thank *NUST, Pakistan* for funding my research and at the same time acknowledge the generosity of *Cranfield University* for keeping me registered for an additional year free of cost with all the facilities for research. I also thank my fellow student *Tracey*, who always helped me in understanding any thing related to operational research and also for introducing me to eBay which proved fatal for my pocket. I am also indebted to my wife *Talat* for her ever present support and unshakeable trust in my abilities without which I could have never crossed the initial landmarks of this task. I must not forget to thank my two sons *Faizan* and *Hassaan* who remained undemanding through out this time and my mother who very patiently waited and prayed for my safe return to home. God be willing, I will try to give them some good time very soon.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This chapter first introduces the context in which the problem discussed in this thesis arises, followed by major contributions of the work and concludes with a brief description of the chapters.

Military simulations are extensively used for planning, training, acquisition of systems, evaluation of weapon systems and equipment, tactics and doctrines. Present day battle scenarios are very complex and highly dynamic. This complexity and dynamism is likely to increase in future. Decisions of present day human commanders have unprecedented effects on the outcome of the battle, due to availability of firepower, mobility, flexibility, and information (Killebrew, 1998). Human-in-the-loop simulations are time and personnel intensive (Peck, 2004). In simulations lacking human intervention, it is no longer a valid method to use the relative strength of opposing forces, together with their firepower, in order to predict battle outcomes (U.S. Army, 1997). The battle outcomes of aggregated forces is not as accurate as the results produced by different entities engaged in combat interactively with their individual plans. Computer implementations of human models populate both types of military simulations that are simulations with and without human intervention. These implementations include human models for individual combatants, followers, and leaders either leading a group of individuals or an integrated platform. The behaviour of these models mimicking humans in military simulations is not sufficiently realistic, particularly with regard to learning and decision making.

One of the problems in decision making is that the automated or computer generated decisions are predictable. In training simulations this predictability in behaviour allows the trainees to play the game of the simulation compromising the aims of the training. Predictability is caused by lack of flexibility in decision making strategies, variability in behaviour, and adaptability. Lack of flexibility in decision making strategies is directly related to the decision making model.

The decision making process in command agents in present day military simulations, such as Warsim 2000 (McNett et al., 1997), and ModSAF (Ceranowicz, 1994a and 1994b) and JANUS (Pratt and Johnson, 1995) interfaced with decision support systems such as DICE (Bowden et al., 1997) and course of action generators such as

Fox-GA (Hayes et al., 1998) and CADET (Ground et al., 2002), is predominantly based on the military decision making process (*MDMP*) which in turn is based on the well known multi attribute utility analysis (*MAUA*) model. This decision making process is prescriptive; instructing how humans should take decisions. It is in contrast to descriptive decision making models that explain how humans actually make decisions. The experienced decision makers do not follow the *MAUA* process of generating multiple options and evaluating them on abstract dimensions. They have been observed to make decisions according to the recognition primed decision making (*RPD*) model described by Klein and associates after studying fire-ground commanders, nurses in intensive care units, and other experts for sustained periods in their natural settings (Klein, 1998). The *RPD* model describes how decision makers can recognize a plausible course of action as the first one to consider. A commander's knowledge, training, and experience generally help in correctly assessing a situation, and developing and mentally wargaming a plausible course of action. The *RPD* model falls under the rubric of naturalistic decision making (*NDM*) (Lipshitz et al., 2001). *NDM* is characterized by features such as dynamic environments, uncertainty, ill defined goals, high stakes, and experienced decision maker. Mental simulation is an important part of Klein's *RPD* Model. The attempts to develop computer models of *RPD* so far (Warwick et al., 2001), (Forsythe and Xavier, 2002), (Liang et al., 2001), (Ji et al., 2007), (Gonzalez and Ahlers, 1998), (Kunde and Darken, 2005), (Norling et al., 2001) and (Sokolowski, 2002) have failed to implement mental simulation for sequential evaluation and modification of plausible courses of action. *Wargaming* a course of action by mentally simulating it before making decisions is required in situations where one course of action may not clearly be recognized as the most suitable for the present situation (Klein, 1998).

Although, the behaviour of command agents is observed externally for realism, this behaviour is real only if the human behaviour model is based on plausible psychological theory. If for ease of implementation the underlying theory is compromised then the model is brittle and displays non-understandable behaviour in unexpected situations. Alan Turing in his seminal work *Computing Machinery and Intelligence* in 1950 (Turing, 1950), proposed the '*Imitation game*' in which the performance of a machine mimicking humans is evaluated by observing the external behaviour of the model. But also in this test, a model based on a psychological theory of how human's converse is most likely to outperform other models that are developed aimed at only deceiving the observer.

Human cognition is modelled with the help of cognitive architectures. Varying levels of sophistication exist in cognitive models. One of these cognitive architectures is *Soar* (Newell, 1990). *Soar* has been developed as an architecture of general intelligence (Laird et al., 1987). It finds solutions of problems by exploring problem spaces through applying available operators to it. *Soar* provides the basic infrastructure to implement all aspects of *RPD* model and especially 'mental simulation' for course of action evaluation.

One important contributor in modelling human behaviour for military simulations may be the gaming industry but there is difference in the overall aims of model development. In military simulations the requirement is of realism to produce most accurate effects while for gaming applications entertainment is the primary consideration (Laird, 2000). In gaming artificial intelligence (AI), the requirement of human like behaviour reduces to an illusion of human like behaviour because the main aim of the development of this type of behaviour is only entertainment and there is no emphasis on accuracy or competence of the underlying psychological theories and the resulting behaviours. In the gaming industry, if the behaviour of an opponent provides a satisfying game experience to the player by not being very easy nor very difficult to kill then the purpose is served. Therefore, in gaming applications emphasis is on visual graphics, audio, and other features that enhance the user experience. But still both industries share a lot in common and can benefit from each other to a great extent (Peck, 2004).

In preceding paragraphs, the requirement of modelling and simulation and the importance of realistic human behaviour models for military simulations representing conventional warfare are discussed. Most of military conflicts now involve asymmetric warfare. Where the relative military power of the belligerents is significantly different the conduct of warfare changes form and is known as ***asymmetric warfare***. This form of war deals with uncertainties and surprises in terms of ends, ways and means. And this uncertainty increases with dissimilarity in the opponents. As the conduct of war has drastically changed therefore a commensurate change in doctrine, tactics, procedures, and force structure is required. The contingencies are numerous and the experience of the military does not match. Therefore, realistic models and simulations are needed to cover the gap in experience.

1.1 The problem

The human behaviour representation needs to be realistic in order to give a more accurate effect of a human commander's decisions on the course of the battle, as suggested in the annual report of army-after-next (U.S. Army, 1997) and (Killebrew, 1998).

Pew and Mavor (1998) have pin pointed common short comings of the existing decision models, their comments are presented in their own words, "*First the decision process is too stereotypical, predictable, rigid, and doctrine limited, so it fails to provide a realistic characterization of the variability, flexibility, and adaptability exhibited by a single entity across many episodes. Variability, flexibility, and adaptability are essential for effective decision making in a military environment.... Second, the decision process in previous models is too uniform, homogeneous, and invariable, so it fails to incorporate the role of such factors as stress, fatigue, experience, aggressiveness, impulsiveness, and attitudes toward risk, which vary widely across entities.*"

To address the problems of inflexibility in decision making strategy, predictability in behaviour, and inadaptability in command agents used in military simulations, this research proposes a computer model of command agent based on recognition primed decision making (*RPD*) model implemented in the *Soar* cognitive architecture. This research aims to address the problem of inflexibility in decision making strategy, by varying the decision making strategy according to psychologically plausible processes. It aims to address the problem of predictability in behaviour, by providing variability in behaviour not through randomness which produces undesirable behaviour but through satisficing which is giving suboptimal choices to the agent that promise a sufficient level of success in achieving the goals. It also addresses the problem of inadaptability, by making the agent learn from its experience using a learning procedure called chunking in *Soar* which is a form of explanation-based generalization. This learning process increases the efficiency of the agent with experience and it can also transfer knowledge to similar tasks. This model also promises to alleviate the problems of long development times of agents and knowledge elicitation from subject matter experts by incorporating mental simulation capability in the agent which assists the agent in handling new situations effectively. The ability to handle new situations is proposed to be further enhanced by incorporating a pre-trained artificial neural network in the architecture of the agent. The context in which this problem is addressed is discussed below.

The military community in general has recognized the importance of realistic simulations and identified the short comings in the present models of human behaviour and realized the importance of realistic representation of human behaviour for military simulations (Erwin, 2000), (Erwin, 2001), and (Book, 2002).

The panel on *Strategic directions in simulation research* (Nicol et al., 1999) emphasises the need to develop techniques to insert reactive and intelligent human behaviour in the virtual world for military training simulations and computer games. The usual techniques of modelling human behaviour like finite state machines (*FSM*) (Kohavi, 1978) that encode specific behaviours and define the transition conditions from one behaviour to the other, are discovered to be limited in representing realistic human behaviour. The humans interacting with these entities identify their limitations and take advantage of them, thereby compromising the aims of the simulation. The panel points out that these behaviour representations are unrealistic by exhibiting only correct and by-the-book behaviour.

The military is using distributed simulations for design and evaluation of equipment and weapon systems, military planning, and training. The popular use of computer generated forces (*CGF*s) to support the above simulations as opposing forces and also collateral friendly forces requires modelling realistic human behaviour. Moreover, in the same context higher and lower echelons are also modelled to see the effect of commands given by the higher echelon and reaction from and implementation of commands given to the lower echelons, on the progress of the battle.

In most of the applications of models of human behaviour, it's the external behaviour that is observed for realism. In constructive *wargame* simulations, it may only be the outcome of the battle or the movement of the troops. In distributed simulations, the individual behaviours of combatants and units are observed together with their execution of plans, and outcome of the encounters. The realism is judged on the measure of results and behaviours of individuals and groups meeting the expectancies of the observers.

Aggregation is at different levels. Representation may be at an individual level or at the group level. The individual entity may be an individual combatant like a dismounted infantry soldier, a ground vehicle or air system commander, a squad or platoon leader or a commander at a higher level.

The first step of realistic decision making is realistic situation awareness. Although, situation awareness is not directly observable in the simulation unless explicitly displayed but it is indirectly observed in the out come of decision making that is the action taken. The directly observable part is the actions such as which way the entity moves, given the plan, the environmental factors, and the situation presented by the opposing forces. More examples of such like actions are; shoot, retreat, seek cover, advance, follow, pursuit, and evade. To seem real the decisions should be consistent with the current goal. The goals should change according to the situation. Sometimes, while keeping the main goal in view, human commanders do take opportunistic approach and make decisions to take advantage out of an opportunity presented by the opposing forces. The expectation of the observers is to witness these types of decisions also.

1.2 Modelling and Simulation

A ***model*** is a physical, mathematical, or logical representation of a system, entity, or process and a ***simulation*** is a method of implementing a model over a period of time.

1.3 Types of military simulations

Military simulations are differentiated based on what is modelled and what is real. The spectrum is divided in three parts, starting from all real it moves up to completely synthetic environments and entities including humans. ***Live simulation*** involves real people operating real systems. It is used for maintaining readiness and testing new employment concepts. It is independent of *HBR*. In ***virtual simulation***, real people operate simulated systems. Virtual simulations require human-in-the-loop intervention. Human intervention is in the form of decision making, or exercising motor control skills, such as firing a weapon system, flying an aircraft, controlling fire of weapons and weapon systems. The Close Combat Tactical Trainer (*CCTT*) is an example of virtual simulator (Johnson et al., 1993). The human controller represents the decision making and tactics. Intelligent allied or opposing forces may be used and it needs *HBR*. ***Constructive Simulation*** involves simulated people operating simulated systems. It is used for planning, training, force development, organizational analysis, and resource assessment. Humans set up the simulation, after which the simulation runs on its own and produces outcomes that can not be controlled by humans. It is totally dependent on *HBR* either implicitly or explicitly.

1.4 Requirement of command agents

Human behaviour representation benefits users of following types of military simulations:

- Training
- Mission rehearsal
- Analysis
- Acquisition
- Joint force analysis (Pew and Mavor, 1998).

1.5 Computer generated forces (*CGF*) and how to judge them

U.S. Department of Defense Modelling and Simulation (M&S) Master Plan defines CGF as "A generic term used to refer to computer representations of entities in simulations which attempts to model human behaviour sufficiently so that the forces will take some actions automatically (without requiring man-in-the-loop interaction" (DoD, 1998). CGFs operate in synthetic environment.

1.5.1 Synthetic environment

A synthetic environment is defined in the words of Dompke (2001) as "*Internetted simulations that represent activities at an appropriate level of realism. These environments may be created by within a single computer or over a distributed network connected by local and wide area networks and augmented by realistic special effects and accurate behavioural models.*" A synthetic environment links any combination of models, simulations, people and equipment, real or simulated, into a common representation of a world. The environment of a simulation is represented with its contents like ground, objects, natural and man made features, etc. the effects of some actions are also represented. *CGFs* are one of the components of synthetic environment. For example for UK armoured vehicle training, there are two combined arms tactical trainer (CATT) sites at Warminster and Sennelager, UK. Each of those has nearly a hundred full mission, full crew, high fidelity, vehicle specific simulators, all together capable of hosting a full armoured brigade group. The *CGF*s in this synthetic environment simulate all the enemy forces and civilian population.

1.5.2 Synthetic forces

We define synthetic forces in the words of Ritter (2002), "*Synthetic forces exist in military simulations, sometimes alongside real forces that have been instrumented and linked to the simulation. The physical aspect represents the movement and state of platforms (objects) in the simulation, including such aspects as maximum speed and the actions that can be performed in the world. The behavioural aspects of a synthetic force platform determine where, when and how it performs the physical actions, that is, its behaviour".* Modular Semi-Automated Forces (*ModSAF*) (Ceranowicz, 1994a and 1994b) is an example of synthetic force.

1.5.3 Semi-automated forces

U.S. Department of Defense Modelling and Simulation (M&S) Master Plan defines semi-automated forces as, "Simulation of friendly, enemy and neutral platforms on the virtual battlefield in which the individual platform simulations are operated by computer simulation of the platform crew and command hierarchy. The term "semi-automated" implies that the automation is controlled and monitored by a human who injects command-level decision making into the automated command process" (DoD, 1998).

1.5.4 Intelligent software agent

Agency is the degree of autonomy vested in the agent and intelligence is the degree of reasoning and learned behaviour. Thus an intelligent agent must have some degree of autonomy in pursuit of the goal assigned to it which they must exhibit in their behaviour while interacting with the environment and other entities, and some ability of reasoning in order to carry out the assigned task and learning. According to Nwana's typology (Nwana, 1996) the smart agent is an autonomous, learning, and cooperating agent (Figure 1.1). The terms smart agent and intelligent agent are interchangeably used in the literature on agent typology.

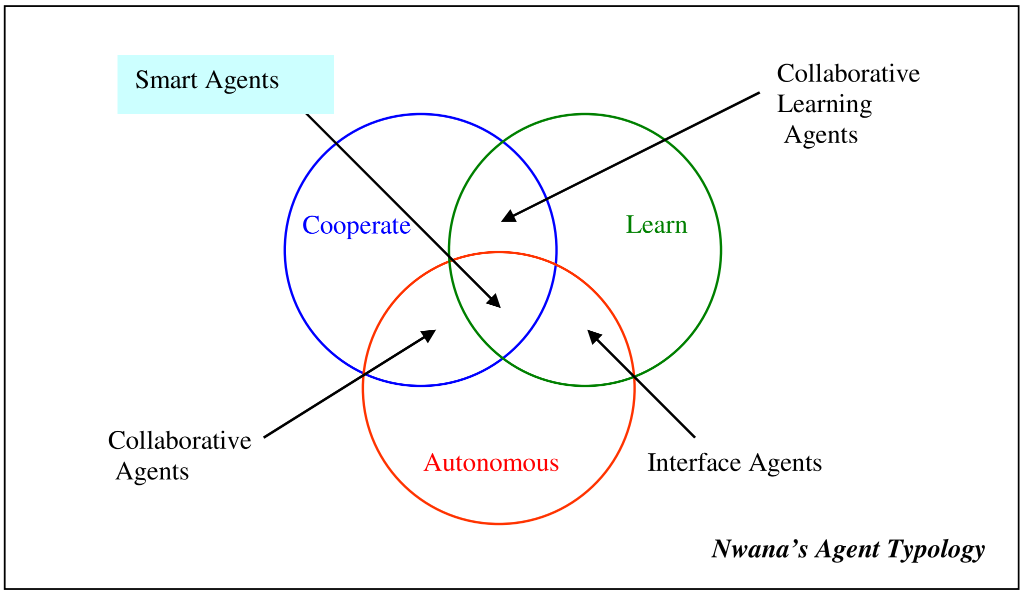


Figure 1.1 Nwana's agent typology (Nwana, 1996)

1.6 Cognitive science

Representing human behaviour involves comprehensive models of human abilities. Since last four decades, computer systems have been considered analogous to the information processing system of humans. Information is acquired, processed, stored, retrieved, and used to accomplish given tasks by both computers and human brains.

Cognitive science based on psychology, linguistics, anthropology, and artificial intelligence (*AI*) is developed to help us understand phenomena like human decision making, natural language processing, perception, motor action, memory, and learning. Decades of experimental data from research on human psychology has found regularities in human behaviour and some of them are very robust. Human regularity is defined as the behaviour that all humans seem to exhibit. One of the most robust regularity in motor behaviour is the Fitts' Law (Fitts, 1954), which predicts how long it will take a person to move a pointer from one point to a target location as a function of the distance to be travelled and the size of the target. Then there are other human behaviour regularities like: the garden path phenomenon, regularities about item recognition and verbal learning. The garden path phenomenon comes from the field of psycholinguistics, which contrasts sentences that are very easy for people to understand from those that are very difficult for people to understand (Gibson, 1990).

Regularity about item recognition is found by Sternberg (1975), which describes how the time taken to decide whether an item is on a memorized list of items increases linearly with the length of the list of items. Regularity about verbal learning is that if an ordered list of items is memorized by repeated exposure, then the items at the ends of the list are learned before the items in the middle of the list (Tulving, 1983). With these descriptions of regularities also come the theories that explain these regularities. As these theories come from different disciplines, they are not coherent and it is very difficult to put them together into a model and develop a human behaviour model straight away. However, there have been attempts at developing unified theories of cognition (*UTC*); the most popular implementations of *UTC* are *Soar* and *ACT-R*, developed by Newell (1990) and Anderson (1993) respectively. None of these implementations have modelled the complete phenomenon of human cognition rather these attempts at *UTC* are considered as a good starting point to bring all the incompatible 'micro theories' together to develop a bigger picture.

1.7 Definition of human behaviour representation (*HBR*)

Human behaviour representation is representing the behaviour of humans as individuals, leaders whether leading a group of men or an integrated platform like a vehicle with crew, followers, and groups, so that they can appear to be real to observers and to humans interacting with them. The human behaviour representation (*HBR*) in this thesis refers to representation of behaviour of humans involved in military activities such as operations and training. *HBR* and ***human behaviour model*** (*HBM*), in this thesis, are used interchangeably. An *HBM* may be an individual combatant like a dismounted infantry soldier, a ground vehicle or air system commander, a squad or platoon leader or a commander at a higher level.

1.8 Definition of command agent

We define **command agent** as "intelligent agents representing human combatant or a military commander leading a group of combatants or human controlled platforms that autonomously take decisions in military simulations".

1.9 Flexibility in decision making strategy

We define flexibility in decision making strategy as the ability of the decision maker to adopt different decision making method in different situations. For example lack of experience or knowledge in the problem under consideration may require the decision maker to contemplate more and give a detailed consideration to all the factors as compared to a situation where the decision maker has experience and knowledge in the problem under consideration. Adopting the same decision making strategy every time is not what humans do and is not considered realistic. Until there are a number of different decision making strategies the internal and external behaviour moderators like knowledge, stress, and fatigue etc. can not be realistically represented in decision making behaviour of command agents.

1.10 Variability in behaviour

We define variability in behaviour as the difference in observed behaviour when one or more entities are placed in the same situation while performing the same task. The entity may be real or virtual subjects. The situation includes the environment also, as part of the situation is formed by variables from the environment. The variability in behaviour is divided into two types: variability within an entity and variability across entities (Wray and Laird, 2003). ***Within-entity variability*** is defined as the variability observed in the behaviour of an entity in performing the same task in different episodes of the same situation. ***Across-entity variability*** is defined as the variability observed in the behaviour of more than one entity in comparison to each other in performing the same task in the same situation during a single episode.

1.10.1 Requirement of variability in behaviour of synthetic forces in military simulations

Synthetic forces populate both virtual and constructive military simulations for training and development and evaluation of new weapon systems and doctrines. These synthetic forces are representing either humans or human controlled platforms. These platforms include unarmed transport vehicles, tanks, planes, attack helicopters, ships, etc (Wray and Laird, 2003). These computer generated forces represent opposite forces, own and allied forces, and neutral forces. Trainees interact with these synthetic forces during training on these simulations. Trainees engage enemy forces, participate in operations alongside friendly forces, or command own forces. The most important reason for modelling variability in behaviour of these computer models of humans is to enhance the training benefit and prepare the trainees for real combat where every human combatant behaves differently. Training with predictable non-varying behaviour affects the training in various ways.

It has been noted in military simulations as well as in computer games that if the computer generated opponent is easily predictable in a given situation then the trainee or the player *games* the situation taking advantage of this limitation of the opponent (Wray and Laird, 2003). This is a short cut to actual training and creates incorrect performance measures that may prove to be fatal for the trainee during actual combat and may put the group or the unit of these trainees into an ambitious task not commensurate to their capabilities.

Usually an aim in designing the opponents is to design them such that they produce the best tactical behaviour with a view to make the trainees expert in fighting the most well trained opponents. This is good but not real and may prove counterproductive. The trainees should be able to handle all kinds of situations that may arise in the type of combat for the intended training. A variant from the behaviour of a well trained combatant may be a foolishly brave act of an opponent that may surprise the trainee in actual combat. For example, an opponent waiting in an open space after taking a turn around a building while he is expected to run along the building to find a cover and then wait should be able to surprise everybody. Therefore, training against a combatant with well trained behaviour is not the complete training rather training against all possible behaviours from the opponents is required for the real combat. In the same way, it is also a part of good training to expose the trainees to heterogeneous team mates and under command forces. It is also part of training to coordinate and cooperate with team mates that respond differently in a situation. It is also important for the trainees to be able to organize and make best use of under commands with different skills and varying knowledge levels and expertise.

In military simulations aimed at development and evaluation of new weapon systems and doctrines it is important to explore the extremes of all possible responses to a situation. For example, whilst designing a weapon system its response to an incorrect sequence in pressing a set of buttons may never come to light because it is quite unusual for anybody to do it. A combatant with a kind of variability in behaviour covering even some part of the incorrect behaviour space may expose this fault in the design. Variability across entities in a simulation may highlight the fact that a weapon system that is very effective against opponents with one type of behaviour may not be as effective against opponents with another type of behaviour.

1.10.2 Sources of variability in behaviour

Sources of variability are different for within-entity variability and across-entity variability. Across-entity variability is produced due to difference in knowledge, experience, personality, culture, religion, and emotional state. Within-entity variability is produced due to the variations in mental and physical conditions of the entity. Motivation, emotional state, fatigue, and adaptation due to more experience or knowledge are some of the factors that produce variability in behaviour within an entity. In computer models of human behaviour the variability across-entities may be produced by giving different knowledge, experiences, and personality to different entities. Producing variability within an entity is difficult to achieve.

An entity becomes unpredictable if randomness is introduced in its selection process of actions that produces behaviour. But this randomness produces undesirable behaviour which lacks coherence and salience of actions that should have been exhibited in the behaviour of an agent in pursuit of the assigned goals. This behaviour is not human-like. The requirement is to produce human-like variability in the behaviour to be unpredictable but not arbitrarily random. Humans have a tendency to select one course of action more often than other applicable ones. Therefore, if a population of behaviours created by repeatedly running the same episode for a single agent, is observed then that population should be able to represent the overall behaviour while a particular single episode may be different. Similarly, if a population of behaviours of same type of agents in an episode is observed then the population should be able to represent the over all behaviour while the individual behaviour of agents may be different. This is important for training because recognising a pattern is part of training that will be missing in case of variability in behaviour of agents produced by arbitrary randomness.

Behaviour validation seems to be at odds with behaviour variability as validating a changing behaviour for the same situation naturally looks more complex and difficult.

1.11 Learning

Learning is important for command agents or *HBM*s for many reasons. Learning represents expertise and experience. In military simulations, human-like command agents take the role of local commanders, subordinate commanders, opposition force commanders and even own or enemy single combatants in a loose command structure where they make individual decisions during an operation. A human-like agent in any possible role who encounters a situation for a second time is likely to behave differently in the light of the experience gained from the previous episode. Keeping in view the learning procedures and methods present in current military simulations, learning may be divided into two categories: first is off-line learning; and the second is learning during the simulation. Off-line learning is the method in which the agents are trained when they are not participating in a simulation and which means the agent is not adaptable during the simulation. Off-line learning may be with or without human intervention. The second method of learning refers to the learning methods that are applied during the life of an agent within a simulation and it is adapting its behaviour with in the simulation (Ritter, 2002).

Both of these types of learning can be used to assist the modeller in developing the agent but it's the second type which is significant in representing learning in humans that occur in a very short span of time (Pew and Mavor, 1998).

The procedures and rules used by *CGF*s in military simulations are usually very complicated and it is very difficult to extract this knowledge from subject matter experts (*SME*s). Therefore, one objective of learning is to automatically train an agent to have various levels of knowledge and skills. Moreover, it is very difficult to model the agents for each and every situation that may be encountered by the agent in a simulation and it is very helpful if some general information is given by the *SME* and the agent learns to handle similar situations automatically. It is relatively easier for the *SME*s to provide strategies or courses of action at a higher level and then give rules and general guidelines to implement lower level actions than giving details of all actions down to atomic level. Therefore, a learning method that may automatically decompose a higher level course of action and learn what to do with the help of some general rules is also required.

Learning from experience by observing the outcomes of previous decisions is also a requirement for command agents. This is reinforcement learning of the agent based on a reward signal from the environment. In case of computerized *HBM*s, if the reward is immediate then it is easy to relate the reward to a decision or an action but it becomes very difficult for delayed rewards. For example, in military operations on urban terrain (*MOUT*) a combatant agent fighting inside a building in the presence of enemy forces outside peeps out of the window and get shot on his helmet and learns from the reward of the action that its not very safe to peep out in that situation. Now if he is not shot at that moment in time rather his presence is revealed to the enemy and later after lapse of some time when the agent is filling its rifle's magazine is injured by a grenade that is whirled in from the window. What does the agent learn out of this episode? The agent needs to keep a record of all previous actions to take any advantage in learning from this episode. Or may be there is a requirement of some reasoning system with sufficient domain knowledge to take advantage of belated rewards. In cases where the actions are hierarchical and so is the associated reward then the problem reduces but only to an extent because belated rewards at the same level of hierarchy still remain a problem. Learning by observation is yet another approach in which the computer agents learn behaviours by observing an expert perform them (Stensrud, 2005).

The artificial neural network is one candidate technology which provides robust learning in noisy, dynamically changing, and uncertain environments. A well trained neural net requires large number of examples which is often a problem in military domain.

One suitable candidate for learning in military domain is explanation-based generalization (*EBL*) (Mitchell, 1997). *EBL* is a type of inductive learning in which learning augments the information provided by the historical examples using domain knowledge and deductive reasoning. This aids the learning process and substantially reduces the number of training examples required for adequate learning. Although, *EBL* has problems of its own such as over generalization, it is suitable in military domain because of its ability to learn using very few training examples. *EBL* is further discussed in Section 5.2.6.

1.12 Contribution of this research

This research contributes in the field of human behaviour representation for military simulations; specifically in proposing a command agent model incorporating flexibility in decision making strategies, variability in behaviour, and adaptability. The main features of the research are discussed as follows:

- Parts of the recognition primed decision making (*RPD*) model is successfully implemented in the *Soar* cognitive architecture in a way that is capable of mimicking some decisions made by military commanders in land battlefield settings.
- The model implements Level 1 *RPD*, when sufficient knowledge exists it recognizes a situation in a changing context. Level 2 *RPD* is partially implemented; information available in the environment is processed to make cues in order to recognize a situation. The story building part of Level 2 *RPD* is not implemented.
- Mental simulation forms the basis of Level 3 *RPD* model. Mental simulation has been implemented in this model with such an inherent flexibility to accommodate all types of requirements that are expected to be encountered while making decisions using *RPD* model.
- Flexibility in decision making strategies based on psychological theories is achieved. Decision making strategies are based on experience and extent of knowledge.
- Variability in behaviour across individuals is a desirable characteristic in human behaviour representation. Variability in behaviour across individuals is achieved based on the type of experiences in long term memory of similar agents.
- Variability in behaviour within individuals over different episodes of the same task is a very difficult phenomenon to model realistically. *Within-entity variability* is achieved in this model not through randomness which introduces undesirable behaviour but due to reasonable but sometimes sub-optimal choices made by the agent.
- The single command agent of the developed model exhibits adaptability across various episodes which adds the much desired dynamism to the simulation environment. The agent learns from its experience. The learning is based on the chunking phenomenon inherent in *Soar* which is a form of explanation-based generalization.

- The agents also exhibit transfer of knowledge from one task to the other in case of overlapping problem spaces within tasks.
- Due to the ability of the agents to mentally simulate courses of action it is possible for the agent to handle new situations very effectively. Which relieves the modeller from coding behaviours for all situations expected to be encountered in a simulation and this in turn reduces the development time of the agent.
- The strategies to form experiences in the long term memory of the agents are required only at a higher level with general rules to evaluate actions at lower levels which is easier for the subject matter expert to describe and less tasking for the knowledge engineer to elicit. This reduces the time and effort in the development of the agent. The ability to mentally simulate the candidate courses of action and adaptability inherent in the agent further improves its performance.
- To enhance the ability of the agent to handle new situations, a trained artificial neural network is integrated in the proposed architecture, which further reduces the labour of the modeller in coding behaviours for all expected situations.
- The research also developed a simple *RPDAgent* to operate in a simple simulation environment in order to explore the affects of realistic human decision making on the outcome of the battle simulations. The study concludes that the outcome of the constructive military simulations changes if realistic human behaviour is incorporated in these simulations, and the known mathematical and probabilistic solutions for combat modelling help in validating the start point or base line of simulations involving human behaviour.
- The following papers have been published based on the work in this thesis:
    - Raza, M. & Sastry, V. V. S. S. (2007) Command Agents with Human-Like Decision Making Strategies. *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence - (ICTAI 2007),* Vol. 2, pp. 71-74. IEEE Computer Society.

- Raza, M. & Sastry, V. V. S. S. (2008) Variability in Behavior of Command Agents with Human-Like Decision Making Strategies. *Tenth International Conference on Computer Modeling and Simulation (uksim 2008)*, pp. 562-567. Cambridge, England.

1.13 Organisation of the thesis

Chapter 1 of the thesis covers the motivation for research, some background knowledge about cognitive science, types of military simulations and intelligent software agents. The chapter also sets out the problem that is being addressed in the thesis. The context in which the problem is addressed is described in some detail and definitions of some terms that are used later in the thesis are given. The chapter also includes the requirement of various characteristics in synthetic commanders to include flexibility in decision making strategies, variability in behaviour, and learning. In the end of the chapter, the contributions of this research are presented and the organization of the thesis is given.

Chapter 2 briefly describes mission-planning process, presents Klein's comments on classical approach of decision making, and reviews existing computer techniques for representation and acquisition of information required for mission planning. And then briefly discusses human behaviour models and definition of related terms, and describes recognition primed decision making. In the end, it provides an overview of some of the most used existing cognitive architectures as models of human cognition to include *ACT-R*, *Soar*, and *belief*, *desire*, and *intentions* (*BDI*).

Chapter 3 provides the literature review on attempts at the computer implementation of recognition primed decision making model. The models discussed in this chapter are based on different technologies to include multiple trace memory model, physiological model, artificial neural network, fuzzy logic, rule based system, context-based reasoning, and multi agents based systems (*MAS*) such as *BDI* cognitive architecture and composite agents.

Chapter 4 describes the development details of a simple *RPDAgent* to operate in a simple simulation environment and discusses the related experiments and their results. The experiments are focused on the aim of the development of this *RPDAgent* which is to see the affects of intelligent like behaviour on the outcome of military simulations. The chapter also highlights the requirements on technology to implement a synthetic commander based on recognition primed decision making model.

Chapter 5 describes the parts of *Soar* cognitive architecture that are required to comprehend the implementation of the *RPD-Soar* agent discussed in the next chapter. The working memory of *Soar*, its reasoning cycle, conflicts and their resolution, truth maintenance system, and learning in *Soar* are discussed. Some applications of and improvements in *Soar* are also discussed.

Chapter 6 describes the implementation of recognition primed decision making model in *Soar* cognitive architecture and in the later part of the chapter the enhancement of the situation recognition ability of the agent by integrating a trained neural network in the architecture is discussed.

Chapter 7 contains the experiments that are conducted to elucidate the abilities of *RPD-Soar* agent. A total of five major experiments conducted in this chapter are aimed at demonstrating the flexibility in decision making, evaluating performance and behaviour of various types of *RPD-Soar* agents, demonstrating behaviour variability across agents, testing the ability of the agent to recognize a situation in a changing context, testing mental simulation capability of the agent for dynamic situations, demonstrating within agent behaviour variability, and adaptability of the agent. The last experiment is related to integration of a trained neural network in the architecture to enhance the situation recognition ability of the agent. The discussion on the results of these experiments is also included.

Chapter 8 provides the summary and conclusions of the research and also includes recommendations for future work.

2 BACKGROUND KNOWLEDGE

In this chapter, key aspects of decision making processes and the related terminologies are presented. First the mission planning process used in the military is described briefly and then Klein's comments are given on classical decision making approach used in this process. Definition of situation awareness with brief description is presented, and then some artificial intelligence techniques employed in military simulation is discussed. Human behaviour models in use in military simulations are presented, and recognition primed decision making model is discussed in detail. An overview of *ACT-R*, *Soar*, and *BDI* cognitive architectures is given with their comparison in the end.

2.1 Mission planning

In this chapter two types of military commander planning behaviours are discussed, one is *doctrinally correct* and the other is *observed* behaviour. First the former type of behaviour is discussed and the latter is discussed with *RPD*. Doctrinally specified planning process, detailed in U.S. Army Publication, Staff Organization and Operations Field Manual 101-5, has five stages:

- Mission analysis
- Intelligence preparation of the battlefield
- Development of courses of action
- Analysis of courses of action
- Decision and execution

The *mission analysis stage* begins with receipt of an operation order from the higher command and is based on the contents of the order (Pew and Mavor, 1998). The aims and objectives are analyzed with consideration to operational constraints also called limitations that will apply during the course of the operation. The process clearly defines the current situation and the mission objectives. This is a very elaborate process for the higher echelons of command but at lower level such as a platoon, this process is reduced to considering the factors like the mission, enemy, terrain, troops, time available, commonly known as "*METT-T Process*".

The *intelligence preparation of the battlefield* (*IPB*) is the next stage, which is the situation assessment process. This stage may be very complex for a divisional and larger sized force and is not discussed as part of this thesis. For more details on *IPB* interested readers are referred to Doctrine for Intelligence Support to Joint Operations, JP 2-0, dated 9 March 2000 and Intelligence Support to Joint Operations, JWP 2-00. In platoon level operations, the situation assessment is based on observation, cover and concealment, obstacles, key terrain, and avenues of approach (OCOKA) process, with consideration also given to the weather. Terrain analysis forms the major part of this process and has been described in "*FM* 5-33 *Terrain Analysis*, Headquarters Department of the U.S. Army, July 1990".

The *course of action development stage* is the stage of the planning process, in which several alternative courses of action are generated that can achieve the mission. At lower levels the number of plans is usually three but at higher levels such as brigade and higher there may be more alternative plans. Most of the times also at higher level the alternative plans are three and then there are variants of these plans. It is the requirement of army doctrine to generate several courses of action.

In the *course of action analysis stage* of the planning process the candidate courses of action are elaborated and pitched against each other and evaluated on multiple criteria according to the guidelines prescribed in the doctrine, however, there is scope for the commanders to keep their own evaluation criteria.

Course of action selection stage is the stage where decision is made for a plan and usually the highest-rated course of action is selected. Commander selects the plans and refines it, and generates the plans and orders for unit execution.

Monitoring and replanning is the process responsible for assessing the situation and any deviations from the plan, and then developing or calling up new plans to compensate for those deviations.

2.2 Klein's comments on classical decision making approaches

The military uses military decision making process (*MDMP*) which is based on multi-attribute utility analysis (*MAUA*) and decision analysis. *MAUA* is considered as a classical approach to decision making and has certain advantages such as it explains the reasons behind a decision which is a requirement where a decision needs to be justified. Moreover, *MAUA* is a systematic process and is suitable for new or less experienced decision makers. The experienced decision makers in military and other fields involving dynamic situations, high stakes and time pressures have been observed to make decisions according to the recognition primed decision making (*RPD*) model. *RPD* is a type of naturalistic decision making, described by Klein and associates after studying fire-ground commanders, nurses in intensive care units, and other experts for sustained periods in their natural settings (Klein, 1998). Klein while proposing naturalistic decision making evaluates classical decision making which are also called prescriptive or normative approaches to decision making. His comments are presented here in his own words, *"Classical approaches to decision making, such as multi-attribute utility analysis (MAUA) and decision analysis, prescribe analytic and systematic methods to weigh evidence and select an optimal course of action. MAUA decision makers are encouraged to generate a wide range of options, identify criteria for evaluating them, assign weights to the evaluation criteria, rate each option on each criterion, and tabulate the scores to find the best option. Decision analysis is a technique for constructing various branches of responses and counter-responses and postulating the probability and utility of each possible future state, to calculate maximum and minimum outcomes. …… On the surface these strategies may seem adequate, yet they fail to consider some important factors inherent in real-world decisions. Classical theories deteriorate with time pressure. They simply take too long. Under low time pressure, they still require extensive work and they lack flexibility for handling rapidly changing conditions. It is difficult to factor in ambiguity, vagueness, and inaccuracies when applying analytical methods"* (Klein and Klinger, 2000).

2.3 Situation awareness

"Situation awareness is the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in future" (Endsley, 1995). This definition reproduced in the words of Endsley, is the most comprehensive and widely accepted definition of situation awareness. It can be divided into three distinct components or levels to be more meaningful. First level is the identification of the key elements in the environment.

The second level is to elaborate, process, and explain the identified elements or events or a combination of both in order to comprehend their meaning. And the third level is to generate expectations or predict what is going to be the future values of these identified elements which define what is going to be the next situation or may be these identified elements do not remain the key elements in future at all or do not remain observable.

2.4 Terrain representation and estimation of situation in mission planning

Some of the methods of terrain representation and techniques used for deriving information for situation awareness in the process of mission planning are discussed in this section. This discussion provides information on the quality and form of inputs available in computer technology for synthetic military commanders.

2.4.1 Terrain representation

In military simulations, certain features of terrain need to be represented such as terrain surface, bathymetry, physical features to include vegetation, trees, roads, rivers, and building etc., and soil information to include mobility and water content. The terrain surface can be represented using a digital elevation model (*DEM*). A *DEM* is represented as raster (a grid of squares) commonly built using remote sensing. The terrain surface can also be represented as a triangulated irregular network (*TIN*). *TIN* is a vector based representation, made up of irregularly distributed nodes and lines with three dimensional coordinates that are arranged in a network of non-overlapping triangles. The fidelity of terrain representation is an important issue. High resolution is required for realism but this increases data storage and process costs. The *TIN* budget can be effectively managed, by identifying tactically significant and insignificant terrain and accordingly adjusting modelling at high or low resolution (Campbell et al., 1997).

Compact terrain database (*CTDB*) is a highly compact format for terrain representation that covers all features of terrain required in military simulations. *CTDB* is used in *ModSAF*, *JointSAF* and *OneSAF* testbed *CGF*s. *CTDB* represents the terrain surface, bathymetry, physical and abstract features, and contains a polygon attribution table (*PAT*). Elevation data to represent the terrain surface can be stored in elevation grid, *TIN*, or hybrid forms. Elevation grid is composed of elevation posts with elevation data. Abstract features are used by CGFs for path planning composed of arial feature boundaries such as tree canopies, lakes etc. The *PAT* is a global storage area for sets of object attribute values such as the mobility characteristics, water content, surface category and material category of the terrain.

2.4.2 Estimation of situation in mission planning

Gaining situation awareness includes performing assessments of terrain and weather, and enemy and friendly situations. The terrain is studied keeping in view our own mission and resources, and the intentions of the enemy and the size of its force.

2.4.2.1 Line of sight visibility

Inter-visibility between two points on the terrain surface is calculated to model the line of sight (*LOS*) visibility of entities in simulations. Clear line of sight visibility is a dominant factor in selecting defensive positions and also in siting weapons. A popular technique used is to calculate it along an appropriate number of equally spaced rays out to a certain distance from the observer location for each point in digital elevation model (*DEM*). The *LOS* calculations assume a certain target height above ground level (*AGL*) and a certain observer height *AGL*. These heights will vary depending on target types and observer types and also the type of operations. The visibility may also be varied as a function of distance depending upon visibility at that time for more realism in modelling. Some more factors effecting visibility, like forestation and cultivation that vary for different seasons of the year cannot be considered when using only *DEM*s. The problem with this technique arises when the spacing of the arrays is increased to reduce computations, then the distance between observed points at the far end of the array increases (Campbell et al., 1997).

2.4.2.2 Tactical use of terrain

Surface configuration is studied to determine mobility over an area and also to identify suitable areas used for various purposes in military operations. For example, if an area needs to be selected for physical occupation by dismounted infantry soldiers with the aim of defending that area, then the suitability of the area is based on observation and fields of fire towards the approaches leading to it, the size of the area for the deployment of the force, and the local slope changes in any direction. The process of surface configuration study from *DEM*s, is automated through similar techniques as that of edge detection in image processing (Campbell et al., 1997). For edge detection techniques see (Gonzalez and Woods, 2002).

Identifying possible concealed avenues of approach is important for planning all types of military operations. An attacking force will attempt to minimize its exposure both to observation and direct fire as it advances towards a defended location. The defending force would prefer to select positions that have highly observable approaches. Standard path-planning algorithms (LaValle, 2006) may be applied to the visibility scores acquired through the above-mentioned visibility calculating techniques in order to rate the availability of cover and concealment on a particular approach.

The information about surface configuration may be used in conjunction with probable avenues of approach to identify potential obstacle emplacements, pre-planned indirect fire locations, etc.

2.4.3 Spatial reasoning

Forbus, Jeffrey, and Chapmann (2004) have developed a technique called "*Qualitative spatial reasoning*". This technique involves reasoning that can be done on a computer model of a terrain at various levels of resolution, starting from very high resolution terrain representation down to a sketch map.

Fields of fire and observation are important factors considered in military planning and operations. Also cover from fire and observation are the same thing considered from opposite views. Terrain features, like mountains provide cover from fire and also from ground observation. Other kinds of terrain features such as forests block visibility, and thus provide concealment. Regions that satisfy these properties are critical for mission planning. Regions that must satisfy multiple constraints are computed by combining the regions constructed for each constraint.

Spatial reasoning is based on spatial relationships and is reasoned on topological and positional relationships. Topological relationship is based on the relationship of two entities if they are disjointed, touching, or inside one another.

Positional relationships provide qualitative position and orientation information with respect to a global coordinate frame. Compass directions are used to express positional relationships. For example a tank can be north of a small village.

Two entities may also be linked in a positional relationship based on a local coordinate system. For example, if two entities are on a route then it can be said about one entity that is ahead, behind, or at the same location along that path. Centroids of some objects can be used to indicate their position or location. Some entities have a distinct orientation like military units that have fronts, flanks, and rears.

2.5 Current models and simulations in use by the military

There are many models and simulations in use by the military, this discussion is not very exhaustive and only a few of them are discussed here to highlight the requirements of human behaviour representation.

2.5.1 JANUS

JANUS is a constructive high resolution combat model in which individual platforms and soldiers are modelled. Platforms have distinct properties such as dimension, weight, and carrying capacities. It is designed for the level of squad/team/crew to battalion task force but has been extended to brigade and division levels with some loss in fidelity. Engagement results are based on mathematical computations with stochastic distributions of probabilities of detection, based on the line of sight; kill, based on the lethality of the firer and protection level of the target; and hit, based on the ballistic characteristics of the weapons (Pew and Mavor, 1998).
Capabilities and locations of all weapon systems are required to be manually entered when setting up the simulation. Human participation is also required for certain other game decisions (Ilachinski, 2004).

2.5.2 Close combat tactical training (*CCTT*)

CCTT is family of virtual simulations and simulator developed by the U.S. Army and training and doctrine command (*TRADOC*). It simulates battalion sized task force by modelling M1 tank, Bradley infantry fighting vehicle, and AH64 attack helicopter (Pew and Mavor, 1998).

2.5.3 Corps battle simulation (*CBS*)

CBS is a constructive simulation to simulate divisional and corps level operations. It interfaces with other army, air force, naval, and logistic simulations in use by the military. It is used to train staff officers at the Army Command and General Staff College at Fort Leavenworth, Kansas. Staff officers at level of brigade, division, and corps set up the simulation giving inputs that establish unit locations, weapon system status, and intended plans (actions or manoeuvres). It executes approximately three hours of combat based on the player inputs. It computes battle losses and logistic consumptions down to the company and battalion level task forces. The reports and status is given to all levels of command and staff participating (Pew and Mavor, 1998).

2.5.4 Combined arms and support task force evaluation model (*CASTFOREM*)

CASTFOREM is currently the U.S. Army's highest resolution combined arms combat simulation model. This model is designed to simulate combats of task force and combined arms brigade level forces up to about one and a half hour of intense fire fight. The model uses mathematical formulae and stochastic distributions along with subroutines to execute some command and control implemented through a look-up table based on doctrinal tactics and manoeuvres. Model is used for simulating division level operations with some loss in fidelity. Main user of the model is *TRADOC* (Ilachinski, 2004).

2.6 Current *HBR* models

Some of the models representing human behaviour are discussed here; the range, flexibility, and realism vary in these models.

2.6.1 ModSAF

ModSAF is the successor of simulator networking (*SIMNET*) semi-automated forces (*SAF*) developed by U.S. Army's Simulation, Training, and Instrumentation Command (*STRICOM*). The *ModSAF* is designed for training and runs in real time for combat simulations up to battalion level. It is an interactive, high resolution, entity level simulation linked to the terrain database. The user with the help of graphical user interface (*GUI*) can create and control entities. The user also with the help of *GUI* creates, loads, and runs scenarios to simulate a battlefield situation. It provides a credible representation of the battlefield including physical and environmental models. Human behaviour models cover basic activities like movement, sensing, shooting, communication, and situation awareness. These behaviours are hard wired into the model based on the finite state machine model which is restricted to a limited number of states. The finite state machine includes a list of states and commands that are accepted in each state, a list of actions for each command and a list of conditions in a state required to trigger an action (Ilachinski, 2004).

In *ModSAF*, the behaviour is restricted to these actions and as such there is no underlying human behaviour model and human behaviour representation need to be coded into finite state machine. *ModSAF* is used to model individual soldiers, and vehicle and weapon system platforms and the coordinated move of platoons and squads and their tactical actions while unit operations are planned and executed by a human controller (Pew and Mavor, 1998).

ModSAF developed by the U.S. Army has been adopted by the other services. In Synthetic Theatre of War 1997 (*STOW*-97) exercise, four types for *ModSAF* were used. Now a new version named *OneSAF* is being developed that is reported to be more capable.

2.6.2 Intelligent forces (*IFOR*)

IFOR model has been developed to represent the combat behaviour of fixed and rotary wing pilots in combat and reconnaissance missions. These models are based on Soar architecture that has been discussed in detail in Chapter 5. The soar architecture is a rule based system to model human cognition. Soar uses production rules as the basic unit of long-term knowledge. With a view to develop general purpose *IFOR* in future, first a specific context of fixed and rotary wing air operations is used to develop fixed wing attack (*FWA*)-*Soar* and rotary wing attack (*RWA*)-*Soar* pilots for air operations are developed.

2.6.2.1 Fixed-wing attack-*Soar* (*FWA-Soar*)

This project is also known as *"The TacAir-Soar System"*. The system is capable of executing most of the airborne missions that the United States military flies in fixed-wing aircraft. It accomplishes this by integrating a wide variety of intelligent capabilities, including real-time hierarchical execution of complex goals and plans, communication and coordination with humans and simulated entities, maintenance of situational; awareness, and the ability to accept and respond to new orders in flight.

TacAir-*Soar* consists of over 5,200 rules. It uses task decomposition to carry out orders given by the higher command. Its most dramatic use was in *STOW*-97 (Jones et al., 1999).

2.6.2.2 Rotary-wing attack-*Soar* (*RWA-Soar*)

Hill and associates developed *RWA-Soar* (Hill et al., 1997), the system is based on *Soar* architecture and has also added new techniques to facilitate teamwork (Tambe, 1997). The system consists of a team of agents that perform the tasks of an attack helicopter company for a synthetic battlefield environment used for running large-scale military exercises. This system has an approach to teamwork that enables the pilot agents to coordinate their activities in accomplishing the goals of the company.

2.6.3 Synthetic adversaries for urban combat training

Wray and associates have developed synthetic adversaries to train four-person fire teams of US Marines for military operations on urban terrain (*MOUT*) scenarios (Wray et al., 2005). The agents are built using Soar cognitive architecture.

Best and associates have developed similar implementation of synthetic opponents for *MOUT* in 2002 using *ACT-R* cognitive architecture (Best et al., 2002). *ACT-R* architecture will be discussed in detail separately.

2.6.4 Synthetic G staff for headquarters

Mason and Moffat have developed a multi-agent based system to simulate the behaviours of staff officers in military headquarters. Their work is focused on representing G2 and G3 processes of data fusion, decision-making and planning (Mason and Moffat, 2001).

2.6.5 Smart whole air mission model (*SWARMM*)

Air Operations Division of the Australian Defence and Science Technology Organisation developed *SWARMM*, in conjunction with the Australian Artificial Intelligence Institute (*AAII*). It is used to simulate fighter aircraft operations; each pilot in the system is an agent, programmed with *dMARS*, a *BDI*-based cognitive architecture. *BDI* architecture is discussed in detail separately. The agents receive data

from the physical models equivalent to the information a real pilot would receive from his/her vision and instruments (Lucas and Goss, 1999).
SWARMM models squadrons of fighter pilots, with a heavy emphasis on teamwork. It is used to test new equipment and tactics and has proved to be useful for this purpose.

2.6.6 Irreducible semi-autonomous adaptive combat (*ISAAC*)

Introduced in 1997, *ISAAC* is an agent-based simulation of small unit combat. It served as a proof-of-concept that the theretofore-speculative proposition that using swarms of software agents obeying simple rules may reproduce real combat behaviours could be turned into a practical reality. *ISAAC* is developed for DOS-based computers, and its source code is written in ANSI C. The basic element of *ISAAC* is agent, which loosely represents a primitive combat unit (infantryman, tank, transport vehicle, etc.) that is equipped with doctrine, mission, situational awareness, and reaction. Doctrine is default local-rule set specifying how to act in a generic environment; mission is a set of goals directing an agent's behaviour; situational awareness is based on sensors generating an internal map of an agent's local environment; and reaction is a set of rules that determine how an agent behaves in a given context (Ilachinski, 2004).

2.6.6.1 Enhanced ISSAC neural simulation toolkit (EINSTein)

EINSTein was introduced in 1999. It is based on *ISAAC*, but uses entirely new source code and decision algorithms and contains a vastly richer landscape of user-defined primitive functions. The underlying dynamics is patterned after mobile cellular automata rules. It has been programmed in C++, using a windows *GUI* front-end. It uses a genetic algorithm toolkit to tailor agent's rules to desired force level behaviour (Ilachinski, 2004).
EINSTein is used to run simulations for a variety of purposes, land and marine combat, command and control evaluation, and social modelling involving riots and unrest control.

2.6.7 Map aware non-uniform automata (*MANA*)

MANA is an agent-based combat model developed by New Zealand's Defence Technology Agency. *MANA* shares some concepts with either *ISAAC* or EINSTein.

MANA like *ISAAC* and EINSTein uses numerical weights to motivate agent's behaviours and parameter setting for sensor range, and fire range through dialog. Nonetheless, there are certain unique characteristics and advance features in *MANA* including the ability to develop and maintain a mental map in every agent of the locations of previously sensed enemies. Therefore, agent's actions at any time are based on a combination of information from both what they currently perceive and what they remember in their mental maps. This internal picture of the environment is built as the simulation progresses (Ilachinski, 2004).

2.7 Comparison of EINSTein with JANUS

Klingaman and Carlton in 2002 at United States West Point Military academy's Operation Research Center for Excellence compared EINSTein and JANUS to establish the combat effectiveness of EINSTein's agents executing National Training Center (NTC) type-scenario (cited in Ilachinski, 2004). In the scenario, own force consists of an armoured company of fourteen tanks, and enemy force is also of similar size consisting of fourteen main battle tanks. There are two sets of EINSTein agents, one set learns using EINSTein's built-in learning capability based on genetic algorithm and the other set does not learn. Combat results of both set are recorded. These observed actions are then programmed into JANUS and for each case; the combat effectiveness resulting from JANUS is compared to the outcome in EINSTein. Problems are observed in translating agent and environmental characteristics from one model to the other due to model specific constraints and conceptual differences. To conclude the report Klingaman and Carlton offer suggestions for both types of model to make them more compatible: 1) that multi-agent-based models (*ABM*s) need increased fidelity in terms of terrain and weapon systems; 2) *ABM*-like personality traits and realistic decision making algorithms should be incorporated in traditional models, such as JANUS; 3) traditional models should incorporate some mechanism to allow learning.

2.8 Recognition primed decision making

Recognition primed decision-making (*RPD*) is a promising model of naturalistic decision making (*NDM*) (Klein, 1998) and (Lipshitz et al., 2001). *RPD* posits that humans rarely generate a large number of options and then evaluate all of them in parallel on various abstract dimensions to maximize the expected utility. On the contrary, an experienced decision maker recognizes a situation and a course of action as first one to consider.

Earlier Rasmussen (Rasmussen, 1985) used code of behaviour and critical incident interviews to study nuclear power plant operators. He proposed a three-stage typology of skills: *sensorimotor, rule-based, and knowledge based.* His three-stage typology highlights how a person with different level of expertise uses different strategies in decision making. Gladwell (2005) also narrates incidents of correct blink of an eye – snapshot decisions. He acknowledges the work of Klein, supported overall by Paul Van Riper who was president of the Marine Corps University in 1989 and director of Marine air-ground training and education centre, MCCDC, in 1990, in giving this spontaneity a structure. He is of the opinion that it is very difficult to bring out the correct cues and processes that resulted in a correct decision in the blink of an eye. But he also cites the work of Gottman related to reducing complex problems into simple elements and proving that even the most complicated relationships and problems have underlying patterns that can be identified. He also cites the work of Lee Goldman who proves that in picking up these pattern more information than needed or information overload increases the level of difficulty in the process because then one is required to identify the pattern in more clutter (Goldman et al., 1996). When the decision maker thin-slices a situation, recognizes patterns and make a snapshot decision he is unconsciously editing the information.

During the study, by Klein and his associates, of decision makers in various domains under time stress, high stakes, and uncertain environments with multiple players in field settings it is observed that for an expert the recognizable cues feeding the decision process are so overwhelmingly important that only a single option is considered before making a decision. One such study involved experienced naval officers make decisions in the combat information centre of AEGIS cruisers (Kaempf et al., 1996). It is observed in the study that out of 103 cases of situation awareness, 87% are recognized through feature matching, 12% are developed through story building, and only 1% are not explained. In 2003, the Fort Leavenworth Battle Command Laboratory conducted experiments involving a group of serving and retired officers to evaluate *RPD*, and the validating comment was "Yes, that's what we usually do". From the preliminary results it was found that *RPD* took 30% less time than *MDMP* (Ross et al., 2004).

Klein's *RPD* model is arguably one of the best-known models in naturalistic decision-making. Elements of this model have been appearing in the literature previously but Klein and associates integrated all elements and produced a wholesome model (Klein, 1998). This model is characterized with the absence of parallel evaluation of more than one option. It is believed that experienced decision makers identify a plausible course of action as the first one to consider rather than to generate and evaluate a large set of options. Option evaluation is performed serially by mentally simulating action and finding out its weaknesses. Problem solving and judgment is a part of decision making. The *RPD* model consists of three Levels as shown in Figure 2.1.

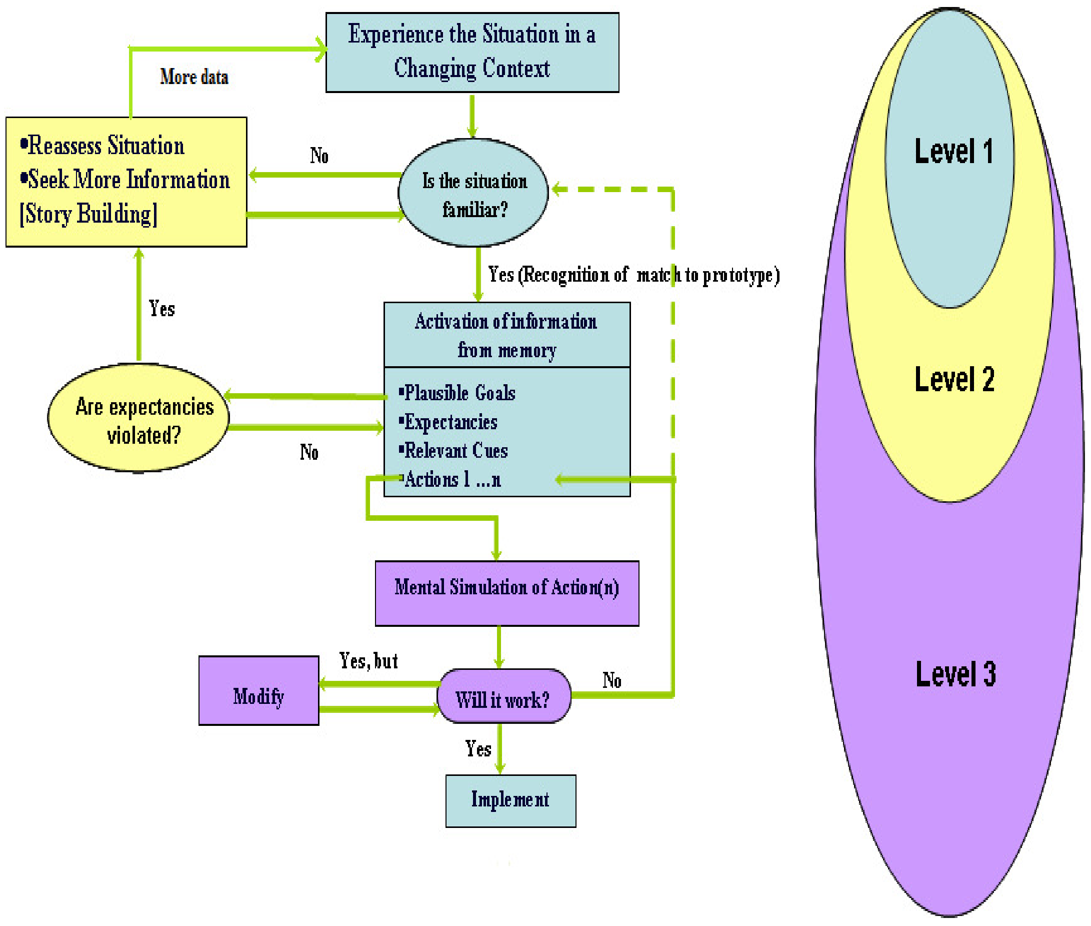

Experience the Situation in a Changing Context
More data
•Reassess Situation
•Seek More Information
[Story Building]
No
Is the situation familiar?
Yes (Recognition of match to prototype)
Yes
Activation of information from memory
•Plausible Goals
•Expectancies
•Relevant Cues
•Actions 1 ...n
Are expectancies violated?
No
Mental Simulation of Action(n)
Yes, but
Modify
Will it work?
No
Yes
Implement
Level 1
Level 2
Level 3

Figure 2.1 Klein's *RPD* model [adapted from (Klein, 1998)]

The simplest and probably the most common case for experts within the *RPD* model is Level 1 (Figure 2.1), where a decision-maker sizes up a situation, forms expectancies about what's going to happen next, determines the cues that are most relevant, recognizes the reasonable goals to pursue in the situation, recognizes a typical course of action that is likely to succeed and carries it out.

Level 2 is a more difficult case, in which the decision-maker isn't certain about the nature of the situation. Perhaps some anomaly arises that violates expectancies and forces the decision-maker to question whether the situation is different from what it seems, or perhaps uncertainty might be present from the beginning. Here, decision-makers do deliberate about what's happening.

Level 3 of *RPD* model is the case in which decision maker arrives at an understanding of a situation and recognizes a typical course of action and then evaluates it by mentally simulating what will happen when it is carried out. In this way, if he spots weaknesses in the plan, he can repair it and improve the plan, or throw it away and evaluate the next plausible action (Klein, 1998).

The model has been tested in variety of applications including fireground command, battle planning, critical care nursing, corporate information management, and chess tournament play (Klein and Klinger, 2000).

2.9 Set effects

Humans have a tendency to set the mind and, at a lower level, the perception in a certain way. The ***Mental set*** called ***Einstellung*** is a tendency of humans to set the mind in a certain framework and to adopt a certain strategy, or procedure. For an example of mental set see (Luchins, 1942). The perceptual set is a similar bias in the way that problems and their solutions are perceived, e.g., nine dots problem (Kershaw and Ohlsson, 2001). Another example of perceptual set is from the work of Coren, Porac and Ward (1978) where they find gender differences in interpretation using ambiguous doodle-like black-and-white figures (Figure 2.2).

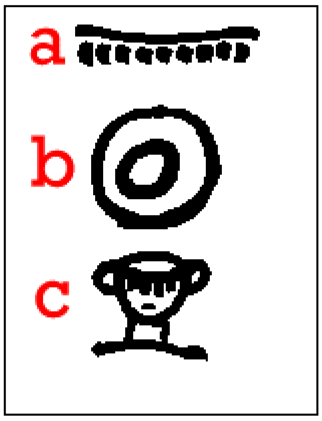


Figure 2.2 Doodle-like black and white figures

A figure which in more cases was viewed as a brush or a centipede by males was viewed in more cases as a comb or teeth by females. Another figure viewed as a target mostly by males was in more cases viewed by females as a dinner plate. And a third figure which was viewed mostly by men as a head was viewed by most females as a cup.

2.9.1 Negative set

Successful problem solving with a particular mental set biases people toward reusing the same set in similar situations. “Negative set” refers to instances in which the “set” leads to a non-productive solution, e.g., Luchin’s Water Jug Problem (Luchins, 1942). Negative set is a phenomenon that may cause a problem in making decisions with the help of *RPD* model. But mental simulation in which the course of action is played out to check for its progress towards the goal helps in avoiding this negative mental set.

2.10 Cognitive architectures

Human cognition is modelled with the help of cognitive architectures. These architectures provide a set of tools and theoretical constraints that help the cognitive modeller. Different architectures make different theoretical assumptions which influence the nature of cognitive models supported by them (Johnson, 1997).

Sensing and perception, and motor behaviour have been added to most of the cognitive architectures. Sensing and perception transform representation of external stimulus into internal representations that are fed to the cognitive process. Cognition encompasses processes such as situation awareness, planning, decision making and learning. Motor behaviour models the functions performed by the neuromuscular system to carry out the physical actions selected by the cognitive processes. Cognitive processes are based on a memory system and an inference engine. Memory system is composed of two types of memories: long term memory (*LTM*) and short term memory (*STM*) which is also called working memory (*WM*) in some of the cognitive models. *LTM* is responsible for holding large amount of information for long periods of time whereas, *STM* holds information temporarily for cognitive processing. *LTM* consists of two types of memories: procedural and propositional memories. Procedural also called operational memory consists of procedural-motor skills i.e., know how of doing a task. Propositional memory consists of a huge variety of knowledge that can be represented and expressed symbolically. Propositional memory is further subdivided into *episodic* and *semantic* memories. *Episodic memory* is involved with recording and subsequent retrieval of unique and concrete experiences of a person with some sense of time attached to them. Whereas, *semantic memory* is concerned with a person's abstract, timeless knowledge of the world that is independent of a person's identity (Tulving, 1983). More discussion on other theories about *episodic* and *semantic* memories is in Chapter 3.

ACT-R is a cognitive architecture that is aimed at simulating and understanding human cognition (Anderson, 1993). *Soar* is a cognitive architecture that exhibits intelligent behaviour (Laird et al., 1987). The beliefs, desires, and intentions (*BDI*) model is based on the theory of human practical reasoning developed by philosopher Michael Bratman (1987). He developed this theory in the mid 1980s. *BDI* is proposed at a higher level of abstraction and researchers make different theoretical assumptions to produce practical *BDI* models.

2.10.1 Adaptive control of thought – rational (*ACT-R*)

ACT-R is a hybrid cognitive architecture based on the experimental knowledge in cognitive psychology and human cognition. It has a symbolic production system which is coupled to a connectionistic sub-symbolic layer like a neural network. *ACT-R* is a parallel matching, serial firing production system. Conflict resolution strategy is psychologically motivated (Anderson, 1993) and (Anderson and Lebiere, 1998). *ACT-R* is focused on higher level cognition but perception motor module is added in *ACT-R* in 2004 and now it is called *ACT-R*/PM (Anderson et al., 2004).

ACT-R has two types of knowledge – declarative and procedural. Declarative Knowledge is the facts we are aware of and which we can describe to others, for example, "Cross country movement for wheeled vehicles becomes difficult after rain fall" and "the visibility is poor in foggy weather". Procedural Knowledge (or know-how) is the knowledge of how to perform some task. We display this knowledge in our behaviour, for example, taking turns at a junction while driving a car. Declarative knowledge is represented in the form of chunks. These chunks consist of *isa* pointers. One *isa* pointer specifies the category and additional pointers describe the contents. Procedural knowledge is in the form of production rules, which are condition action pairs. Both declarative and procedural knowledge are stored in the long term memory. For a production rule to apply, its conditions or antecedents are required to match the chunks in the working memory. The working memory is the active part of the declarative knowledge. The action on the right-hand side specifies some actions to take. These actions can modify the declarative memory.

The chunks are activated on the basis of activation value which is a sum of base-level activation and associative activation. Base-level activation is a value representing the usefulness of the chunk in the past while associative activation reflects the relevance of the chunk to the current context.

Multiple rules may match the pattern of chunks in the working memory and may apply. But as mentioned earlier *ACT-R* is a serial rule firing system and therefore, only one rule is required to be selected to fire. This conflict is resolved by selecting the production with the highest utility. The utility of the production is the estimated cost to achieve the goal subtracted from the product of the probability of achieving the goal if this production is selected and the value of the current goal.

Associations between declarative memory elements (*DME*s) can be tuned through experience this is associative learning and can automatically adjust the strength of association between *DME*s. New productions (procedural knowledge) can be learned through analogy to old procedural knowledge through inductive inferences from existing procedural knowledge and also through worked examples. Production rules are tuned through learning strengths and updating of the estimates of success probability and cost parameters.

ACT-R is applied to model intelligent opponents in military urban terrain operation (*MOUT*) simulation (Best et al., 2002).

2.10.2 Soar

Soar is a symbolic cognitive architecture for general intelligence (Laird et al., 1987) and (Newell, 1990). It has been used for creating intelligent forces for large and small scale military simulations (Hill et al., 1997), (Jones et al., 1999) and (Wray et al., 2005). *Soar* is a forward chaining parallel rule matching and parallel rule firing system. Both the declarative and procedural knowledge are represented as production rules. The production rules are condition-action pairs. The long term memory (*LTM*) is composed of production rules while the short term memory (*STM*) contains only declarative knowledge. *STM* in *Soar* is also the Working Memory (WM) that holds all the dynamic data structures. Impasse in *Soar* is the architecturally detected lack of available knowledge. *Soar*'s basic reasoning cycle is as follows:

- Input
- State elaboration
- Proposing operators
- Comparing and evaluating operators
- Selecting the correct operator
- Applying operator
- Output.

Learning in *Soar* is called chunking which is a form of explanation based generalization. *Soar* has been evaluated extensively as a cognitive architecture against human behaviour in a wide variety of tasks. Some examples of these models are natural-language comprehension, concept acquisition, use of help system etc (Pew and Mavor, 1998). *Soar* is validated for very large-scale military simulation in the TacAir-*Soar* for *STOW* '97, which included 722 individual sorties to be flown by US Air Force. It demonstrated *Soar*'s ability to generate autonomous, real-time, high fidelity behaviour for a large-scale simulation of a complete theatre battle. The current version of TacAir-*Soar* contains over 5200 production rules, organized into about 450 operators and 130 goals (Jones et al., 1999). *Soar* is further discussed in detail in Chapter 5.

2.10.3 Belief, desire, and intentions (*BDI*)

Philosopher Michael Bratman (1987) developed the theory of human practical reasoning, the origins of the *BDI* model lie in this theory. In *BDI* approach the behaviour of the individual agent is shaped by its Beliefs, Desires, and Intentions. Belief is the agent's perception of the environment that may or may not be true, Desires are the states of the world it seeks to bring, and Intentions are the committed plans. A number of researchers have proposed their preferred axiomatizations capturing the relationships between beliefs, desires, and intentions that resulted in various theoretical frameworks for *BDI* architecture (Rao and Georgeff, 1995) and (Georgeff and Rao, 1996). The *BDI* approach was further developed from theoretical frameworks to practical systems, through application of abstractions, and static and dynamic constraints. And this process resulted in a number of successful implementations (Lucas and Goss, 1999); most notable are the procedural reasoning system (*PRS*) (Georgeff and Ingrand, 1990), and its successor "distributed Multi-Agent Reasoning System" (*dMARS*) developed in C++ (d'Inverno et al., 1998).

In computational terms, Beliefs is representation of the state of world, be it in the forms of values of variables or symbolic expressions in predicate calculus. Goals may also be expressed in any of the forms described for Beliefs above, however in what ever way goals are represented, the representation should reflect the desire and not tasks as used in usual computer programs. Computationally, Intentions may be a group of threads being executed in a process. From a theoretical perspective, Intentions are committed plans which are liable to change after observing a change in the external environment. The agent is supposed to create these plans every time it finds a new situation but because of the resource bound, it caches the plan for reuse. Some researchers consider these stored plans to be the part of Belief (Georgeff et al., 1999) while others consider it as a fourth data structure and formally name it as the plan library (d'Inverno et al., 1998).

The *BDI* agent senses the environment, reasons about beliefs, desires and intentions and then performs a series of actions. After sensing the environment the beliefs of the agent may change, changes in beliefs may change some goals and therefore the intentions. Change in intentions will bring new partial plans or recipes into play to achieve goals. If multiple plans are available to achieve a goal then the agent uses rational choice to select a plan which means it will evaluate all options and select the best one.

The Australian Artificial Intelligence Institute (*AAII*) has developed *Oasis* (*O*ptical *a*ircraft *s*equencing using *i*ntelligent *s*cheduling) an agent based air traffic control system and NASA's Space Shuttle Monitoring and Control System, using SRI's *PRS* (Georgeff and Ingrand, 1990). *AAII* in conjunction with Australia's Defence Science and Technology Organisation (*DSTO*) is developing *SWARMM*, a *dMARS* agent based simulation system to simulate dynamics and pilot reasoning of air missions, and provide visualisation.

2.10.4 Summary of cognitive architectures

All of the cognitive architectures have some distinct advantages and some limitations. The first problem in comparing cognitive architectures is that they are universal Turing machines and therefore, it is very difficult to prove that an architecture can not model some phenomena. A Turing machine that is able to simulate any other is called a universal Turing machine or simply a universal machine. A Turing machine is a kind of *state machine*. At any time the machine is in any one of a finite number of states. Instructions for a Turing machine consist in specified conditions under which the machine will transition between one state and another. A Turing machine has an infinite one-dimensional *tape* divided into cells. Each cell is able to contain one symbol, either '0' or '1'. The machine has a *read-write head*, which at any time scans a single cell on the tape. This read-write head can move along the tape to scan successive cells. The action of a Turing machine is determined completely by (1) the current state of the machine (2) the symbol in the cell currently being scanned by the head and (3) a table of transition rules, which serve as the "program" for the machine. The actions available to a Turing machine are either to write a symbol on the tape in the current cell or to move the head one cell (Turing, 1936-7).

The second problem in comparing cognitive architectures is that there are virtual-architectures within the architecture. The problem domain, for which a model or an agent is being developed, is the most important factor in deciding the architecture.

Even within a domain, the specific aspect that needs to be modelled is also a very important deciding factor. For command agents, *ACT-R* has some advantages over others in modelling human psychology, flexibility in learning, probabilistic behaviour, and decision making based on knowledge as well as Bayesian networks. *Soar* is scalable as has been demonstrated in large scale implementations in military simulations and war-games, and it has also been successfully deployed to model teamwork due to STEAM. *BDI* architectures offer proactive planning and teamwork in the basic architecture.

2.11 Chapter summary

The decision making process in use in the military is called Military decision making process (*MDMP*) and is based on multi-attribute utility analysis (*MAUA*) in which the decision makers are encouraged to generate a number of candidate courses of action and evaluate them in parallel on multiple attributes. Decision making in most military simulations are represented by *MDMP*. Klein and associates proposed the recognition primed decision making (*RPD*) model that posits that humans rarely generate a large number of options; on the contrary an experienced decision maker recognizes a situation and a course of action as first one to consider. The courses of action are evaluated serially by the decision maker by mentally simulating them one after the other. For modelling human behaviour, representing realistic human decision making behaviour is imperative, and situation awareness and problem solving is a part of decision making. Human behaviour representation without an underlying psychological theory based on cognitive processes results in brittle models. Human cognition is represented by cognitive architectures such as *ACT-R*, Soar, and *BDI*. Artificial intelligence techniques that are not a complete and unified model of human cognition have also been used to implement *RPD* to represent realistic human behaviour. Some attempts at computer implementation of *RPD* available in the literature are discussed in the next chapter.

3 LITERATURE REVIEW

Computer implementation of any conceptual or theoretical model is a challenge in its own right. The intensity of the challenge further increases for models of psychological theories in general and psychological models in field settings that are outside the controlled environment of the laboratory in particular. *RPD* is one such model of human decision making which involves human cognitive processes such as gathering, storing, retrieving, and assessing information, setting goals, and *sub-goals*, developing and monitoring expectations, performing mental simulation and making decisions. Many attempts have been made at implementing *RPD* agents using various technologies to include multiple trace memory models, physiological models, artificial neural networks, fuzzy logic, rule based systems, context-based reasoning, and multi agents based systems like *BDI* cognitive architecture and composite agents (CA). Relevant literature using the above technologies is reviewed in this chapter

3.1 Multiple-trace memory model

Warwick et al. (2001) developed a computational model of *RPD* (Klein, 1998), based on the decision maker's long term memory (*LTM*) on the lines of Hintzman's multiple-trace memory model (1986). Hintzman (1984) claims that there is only one memory system and that stores episodic traces. The abstract knowledge is not stored but is derived from these traces of experience at the time of retrieval. Multiple trace theories assume that each experience is stored in memory as a separate trace and does not strengthen or modify a prior representation. The new and old traces of even similar experiences coexist in the memory.

The alternate theory assumes that the effect of repetition is mediated by a mechanism different from the one involved in episodic memory tasks. According to this view, the repeated exposure to exemplars of a category produces traces of individual events in episodic memory but also produces an abstract representation of the category in a functionally separate generic memory system (Tulving, 1983)**.**

The decision maker's *LTM* is represented by a two-dimensional array. Each row represents a situation that prompts recognition and the by-products that follow recognition. The *RPD* experience consists of cues, goals, expectations and a course of action, however, in this model only expectations and courses of action are being represented. A probe that is the snapshot of the current situation is sent to the *LTM* and a similarity value for each row is obtained. Each row in *LTM* contributes to this similarity value according to its contents. The situation is recognized if the similarity has a value more than a set threshold. This threshold is set by estimating a value from worst case scenarios where associations between situations and by-products are entirely random.

There are two ways of characterizing situations. One method uses a rich structure of cues, inferences and judgements to identify situations in *LTM*; the other characterises situations simply in terms of cue values. *Flat situation awareness* is a routine whereby unprocessed cue values are stored in the situation awareness array and the unprocessed cues from the environment is straight away used to recognize a situation. The cue values in the driving environment of the implementation under discussion (Warwick et al., 2001) e.g., light colour, the presence or absence of trailing traffic, perceived distance to the intersection, the presence or absence of a police officer, etc. are used to form judgements the driver makes about the situation. These cues are made available in the environment and are straightaway used to recognize the situation stored as such in the *LTM*. But in most real world environments the cues do not wear meaning on their sleeves and need interpretation before this can be used to form judgement. Taking the example of the same authors in their next computer implementation of *RPD* for conflict resolution in an enroute air traffic control (*ATC*) environment (Warwick et al., 2001), where the cues need processing before a meaning can be extracted out of them. The cues in the airspace model are positions, altitudes and headings. But for conflict resolution the air traffic controller must know whether the planes are climbing or descending. In this case the data from the environment drives inferences which in turn form the basis of higher level judgements about the situation.

There are two short comings in this implementation, firstly, the situation has a completely flat structure and this may not always be the case in the real world where complex problems are deep and hierarchical in nature, secondly, mental simulation has not been implemented.

3.2 Human emulation model

Forsythe and Wenner (2000) proposed an 'organic model' to predict the behaviour of engineered system that results from human involvement in these systems. The human cognition emulation model results from this 'organic model'.

Forsythe and Xavier in their work with Schoenwald (Schoenwald et al., 2002) developed a computational model for Level 1 *RPD* (Klein, 1998). In this work they developed a computational model to generate episodic memory for use in human cognition emulation. The application is based on eight embodied-agents in the form of vehicles in a large building trying to move through hallways avoiding collision from the walls and also amongst each other to place a member at the maximum smoke concentration in the building. During the operation they are supposed to keep their wireless communication intact which is based on physical limitations; more distance for line of sight and less distance through walls. For simplicity of computations 15 out of 32 dimensions, and 800 out of 48,000 observations are selected as traces of episodic memory. First the observations are classified using two types of cluster analysis techniques and then these clusters are interpreted using a classification tree model.

Cluster analysis is a form of unsupervised learning and the nature of no-supervision is that there is no knowledge of data structure. Two clustering methods are used: K-means clustering (Forgy, 1965) and (Hastie et al., 2001) and DIvisive ANAlysis (*DIANA*) (Kaufman and Rousseeuw, 1990).

For both the K-means and *DIANA*, the authors considered the number of clusters that can range from one to ten. For the K-means algorithm, five clusters provided adequate partitioning whilst the *DIANA* algorithm requires six clusters. The classification trees derived from K-means and *DIANA* clusters were partitioned on different dimensions. The partition rules developed through this process are applied to all 48,000 observations, i.e., each observation is associated with one of the states developed by each algorithm. This work demonstrates the ability to identify behaviour through schema abstraction using K-means and *DIANA* clustering algorithms and a classification tree analysis.

Forsythe and Xavier adopted a two-tiered approach to develop a human emulator (Forsythe and Xavier, 2002). In this model, the knowledge is represented using a psychological model and a physiological-based model drives this psychological model. The psychological model is the Recognition Primed Decision making (*RPD*) model. In the initial model the decision maker is a perfect decision maker and the agent exhibits no individual differences based on cultural factors or individual experiences. Factors like fear, arousal, stress, etc., collectively termed as ‘organic factors’ (Forsythe and Wenner, 2000), are also not represented. As the knowledge is not represented in the neural model thus this design distinguishes itself from neural net and connectionist approaches.

In this “human emulator”, as the authors have called it, memory has been modelled on the processes proposed by Klimesch (1996). In this memory model, there are neural units operated by dictation from low-level neural processes, e.g., transmitter-receptor interactions, metabolic properties, etc. Neural assemblies are formed by collecting individual neural units.

Episodic memory is modelled by a single distributed neural assembly. Processing demands lead to increased synchronization.

Semantic knowledge is represented by a semantic network. This network consists of nodes, each node represents a concept. Associated nodes are connected to each other; the strength of links varies with the degree of association. The activation of concept nodes is dependent on the activation of its neural assembly.

The pattern recognition process that monitors activation of assemblies associated with individual elements and responds when specified patterns of activation occur. This amounts to matching current conditions to a known situation schema. This pattern recognition process in episodic memory is dependent on a single neural assembly. Rows of the template in episodic memory represent known situation schema and columns correspond to concepts in the semantic memory. A binary number represents activation of a concept. Binary number has a value equal to ‘1’ when a concept is activated and a ‘0’ otherwise. Recognition occurs incrementally in accordance with a race model and when a threshold is exceeded there is activation of the situation schema.

This improved model sets the stage for implementing mental simulation, which is an essential ingredient for Levels 2 and 3 *RPD*; however, as yet only level 1 *RPD* has been implemented. This computational model is resource intensive and as declared by the authors already on the upper limits of response time when it is working on a high end desktop with very little knowledge i.e., only 30 concepts in its semantic network.

The time taken in statistical analysis on experience traces to produce episodic memory has not been mentioned most probably it is an offline process. The episodic memory thus produced is said to be created without any contribution from domain knowledge.

3.3 Artificial neural network (*ANN*) model

Liang et al. (2001) developed a part of the *RPD* model using artificial neural network (*ANN*). Neural networks, also known as connectionistic networks are inspired by principles of neuroscience. A neural network consists of simple processors called neurons or units and these neurons are connected with the help of communication channels called connections. The channels carry numerical data and the neurons are non-linear processors. Neurons process the local data and the data that they receive through their connections. There are several types of neural network architectures (Gurney, 1997) and (Russell and Norvig, 2003), however, Liang has used a feed-forward network with back propagation as learning algorithm. Neural networks support both types of learning: supervised and unsupervised. Most learning algorithms in neural networks are based on the phenomenon of adjusting weights of the connections or links (Russell and Norvig, 2003). Back propagation is a supervised learning algorithm and adjusts the weights of the connections.

A feed-forward network represents a function of its current input, and the only internal state is the weights themselves. Feed-forward networks are structured on layers. The structure may be based on single or multiple layer(s). In a multilayer feed-forward neural network, the first layer is known as input layer, the last as output layer and in between there may be one or more hidden layers (Figure 3.1).

The neural network used, by Liang et al. (2001), is a simple multi-layer feed forward network consisting of an input layer of four nodes, three hidden layers of 12 nodes each, and an output layer of eight nodes. The back-propagation algorithm is used for learning. This back-propagation algorithm implements a gradient descent in parameter space to minimize the output error. The error on the output is the difference between the current output and desired output from a set of training examples. After the net is trained by adjusting the weights on the connections, then the resulting net is used to recognize patterns or classify the input patterns.

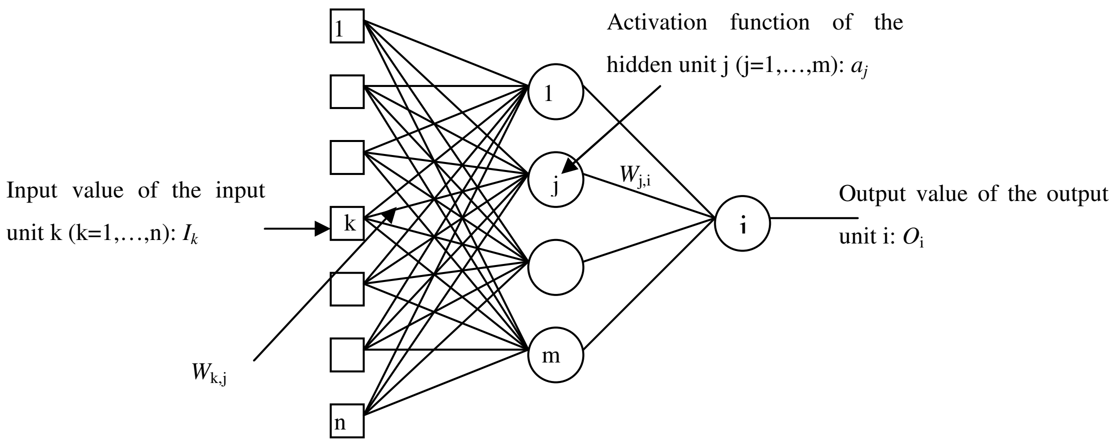

Activation function of the hidden unit j (j=1,…,m): a_j
Input value of the input unit k (k=1,…,n): I_k
Output value of the output unit i: O_i
$W_{j,i}$
$W_{k,j}$
1
k
n
1
j
m
i

Figure 3.1 Artificial neural network

In the work of Liang et al, The environment used is simple and there are 0, 1, or 2 hills of equal size in the battle field. The enemy is a single tank and is supposed to fight from a fixed position until he wins or loses the battle. Own force consists of three tanks and has a three to one advantage in the battle. The plans are based on the options to attack with or without a firebase. The variations in plans within these two options are generated by locating the fire base and the final assault group at different places, by selecting different routes to these locations, and also by developing different combinations of firebase and assault groups by changing the strength in each. When there is no fire base then all the tanks go into their final assault positions using a route and there is no route and position for the fire base. The routes are described by giving only one point in between the starting position and the destination in both cases. The inputs to the net are the normalized Cartesian coordinates of hills. The output is Cartesian coordinates of the final assault group and the firebase positions and one point along the route for both of them. In the opinion of the authors (Liang et al., 2001); for better results more data sets are required, training is suggested to be carried out only for one set of solutions to every scenario, and some other technique be used after the neural network for better solutions. The recommendation by the authors about this hybrid system is that the neural network should be used in reducing the number of plans and then some other technique be used to finalize the plan from this reduced set.

For the scenarios in the test set, some of the solutions generated by the trained neural net are not directly executable plans because they are tactically infeasible. Therefore, there is a definite requirement within the system to evaluate the generated option and modify or reject the generated plan if it is tactically infeasible. But it is difficult to determine from this experiment whether the neural network is not able to generate tactically feasible plans or is it because of the training set provided as the solution or both are contributing factors. Because, in some of the plans given as solutions for the training set, the final position of the fire base is located very close to the enemy tank. Locating fire base so close to the enemy positions is not a usual practice in the tactical situations presented in this paper. In some of the cases in the training set, the final position of the assault group is on the enemy position itself and in same cases approximately three grids south, south west, or south east of the enemy position. This difference in the final positions of the assault group, in our opinion, is not a different strategy but a different representation of the same plan, because in some representation of the attack plans the plan is marked up to the *forming up place* (*FUP*). An *FUP* is a place where the attacking forces form up in attacking formations and from this point on it is usually a direct run to the *objective*. An *objective* is a place that needs to be captured or neutralized in an attack. But the neural net while training considers these as two different strategies. When the complete training data set is used after doubling the data by taking advantage of symmetry and then further doubling it by changing the order of the hill in the data set the network converged only after the error criterion was increased by two decimal points. In this thesis, we have integrated an artificial neural network with *RPD-Soar* agent architecture motivated from this work. Apart from modifying some plans that are not tactically feasible and adding some new plans we have done a major change and that change is in the purpose for which the artificial neural network is used. We have used the artificial neural net for pattern recognition only and not for plan generation, as Liang et al. (2001) also realizes and comments that the option to generate plan directly from the trained neural net did not prove to be successful.

3.4 Fuzzy logic model

Ji et al. (2007) have developed a computational model of *RPD* based on fuzzy logic. Fuzzy logic is reasoning with approximate values rather than precise values as used in predicate logic. Fuzzy logic is derived from fuzzy set theory. Fuzzy sets are sets consisting of elements with fuzzy membership associations that is to say a membership of an element may have any value ranging from 0 to 1. In classical set theory an element of a set may have a membership value of either 0 or 1, in fuzzy sets an element may be part of two sets with different membership association values. Fuzzy set theory provides a method of formalizing imprecise premises and of inferences from them; it is applying logic to language. The age of a person is numerically precise. However, relating a particular age to young can be difficult and confusing. Fuzziness is deterministic and not random, as the nature of the above question is deterministic to a particular person. AND, OR, and NOT operators are fundamental to fuzzy sets. The elements of resultant set of AND, OR, and NOT operations are also partial memberships because the membership being operated is partial and not full. If – then rules are means to inference in fuzzy set theory, e.g., if x is small then y is fast. Fuzzy if – then rules are used in fuzzy modelling and control systems. In this model imprecise cues are represented by fuzzy sets and higher level cues are abstracted out of elementary data using fuzzy reasoning. After developing all the cues for a situation, similarity is measured between the present situation and a prior experience. The module to measure similarity can handle different types of cues involving nominal values, quantitative data, and fuzzy numbers/sets. It is assumed that the prior experience and the present situation have the same set of cues. Local similarity is measured between the experience and the situation separately for each cue. Then a global similarity is computed as the normalized weighted sum of the local similarities. If the global similarity value is above a threshold chosen by the user then the situation is said to be recognized and if there are more than one experience above the threshold then the one with the highest similarity value is selected. There is also an action evaluation procedure which is a form of mental simulation but it has two short comings. First, it does not have a proper mental model where an action is implemented and its effects are observed. Instead a predefined effect is stored with the action. If there is no mental model then an action can not change the mental world and then the agent can not observe the change in cues to determine whether the selected action is taking the agent to the goal or not. Second, human intervention is required to modify the plan instead of architecturally supported decomposition of the course of action to atomic actions where these atomic actions may be put in a new sequence to develop a new plan.

3.5 Context-based reasoning model

Context based reasoning is used to represent intelligent behaviour in training simulations by Gonzalez and Ahlers (1998). Gonzalez states that context-based reasoning is based on the concept of *Scripts* developed by Shank (Schank and Abelson, 1977), for representing knowledge to understand natural language (Gonzalez and Ahlers, 1998). The context-based reasoning paradigm posits that the identification of the future situation is simplified due to the present situation itself as the present situation can only lead to a limited the number of situations. And also, that the context defines a set of actions appropriate to address the present situation.

The concept of Scripts is extended to represent intelligent behaviour in autonomous agents in military simulations (Gonzalez and Ahlers, 1998). Tactical knowledge representation of these autonomous agents is context-based. The contexts are hierarchically divided into the *Mission-context*, *Major-contexts*, and *Sub-contexts.* The *Mission-context* consists of *Major-contexts* which can be sequentially activated to achieve the assigned mission. *Major-contexts* contain the knowledge to perform major tasks and also the knowledge to control its deactivation and activation of another *Major-context. Major-contexts* are mutually exclusive. The *Sub-contexts* are the lower level actions needed to implement a *Major-context*. *Sub-contexts* are mutually exclusive but may be associated with more than one *Major-context.* The Context-based reasoning paradigm encapsulates knowledge about suitable actions for specific situations and compatible new situations into hierarchically organized contexts. That means all the behavioural knowledge is stored in the context base which is the collection of all contexts (Fernlund et al., 2006).

The Context-based reasoning paradigm is comparable to *Soar* cognitive architecture with regards hierarchical goal decomposition. As discussed above the contexts exist to partition the behaviour space and the same can be done in *Soar* by creating a *sub-goal.* In context-based reasoning in order to activate a context, context-transition logic is used which exists to select an appropriate active context at each time step. Whereas, in *Soar* the same is achieved by firing productions to propose an abstract operator which in turn creates a *sub-state* through an operator no change impasse.

The Context-based reasoning also resembles Level 1 and Level 2 *RPD* when its selection of context for activation procedure is analysed which is based on situations. The context is deselected if it is found by the rules controlling the selection of contexts that the premises are not met (Stensrud, 2005). But the similarities end here as there is no concept of mental simulation in context-based reasoning.

3.6 Event Predictor - Mental simulation model

Kunde and Darken (2005) implemented an event predictor to model the mental simulation part of *RPD*. They have applied and tested the model on a scenario built in a simulation environment Combat XXI (Kunde and Darken, 2006). In this model, the agent decides to fire or hold fire depending on the prediction from the mental simulation part of the model as to how many red tanks will be observed by the blue tank commander in the next observation and when this next event is expected. The mental simulation component is based on a Markov Chain. A Markov Chain is a stochastic state machine with the property that the transition to the next state is dependent only on the present state and not on the previous states. The transition probability from state *i* to state *j* is the frequency of transition from state *i* to *j* in the observations up to the current observation. These transition probabilities are normalized so that the sum of all the transition probabilities that any state can transition to equals 1. A state is defined as the number of enemy entities detected in an observation. The agent stays in a state until it observes a change in the number of enemy entities. At this point in time the agent changes its state and the transition probabilities and mean dwell times are updated.

The straight forward method in this state machine may be to predict events with the highest transition probability. However, a state machine based on this approach will always select the most likely transitions and the states with less likely transition probabilities will never be reached. In order to also predict events with low transition probabilities, a Monte Carlo simulation was used for sampling the values from the probability distributions as estimates. The agent decides to fire or hold fire based on the prediction of the next event. It decides to fire if it is predicted that the next state will either have less entities or the mean transition time exceeds a preset threshold. One hundred Monte Carlo simulations are run for three transitions ahead at each decision point. The mode (most common single outcome) of the 100 runs is selected as the sequence ahead.

This is one of the only two known attempts at implementing the mental simulation part of *RPD*. This event predictor is not flexible enough to accommodate all kinds of activities that may be carried out in a mental model in *RPD*. In this model, the designer of the agent has to know all the states that the system can transit to and from, that may be too many in a complex situation as the number of state explodes with increasing complexity. Not only the states but the transition probabilities of states must be known from the beginning, some transition probabilities may be 0 or 1 but others will have to be determined prior to the design of the agent. The authors declared that the learning may be done while in active use but for these experiments learning has been done off line.

3.7 Bratman's belief, desire, and intensions (*BDI*) cognitive architecture model

Norling (2000) discusses three approaches of implementing the *RPD* model based on *BDI* architecture. First, is a 'Naïve' approach, in which the agent recognizes all possible situations and identifies an individual plan for each situation without having to choose from multiple options. The agent is assumed to identify the subtleties in situations. This approach has similarities with case-based reasoning (*CBR*) (Kolodner, 1993); it may work for simple problems with limited situations but not for complex problems. Second, it is a preference-based approach, in which the plans are weighted and the plan with the highest weight is selected, in case some choices have equal weight then one out of them is randomly selected. Initially, all the plans are equally weighted. If a plan succeeds its weighting is increased and if the plan fails the weighting is decreased. This approach is a form of reinforcement learning. The third approach is context-based, in which the agent adapts plan context. It refines the context until the overlaps are removed. This method requires the agent to record the state each time a plan is used and then use reflection to work out what caused the plan to fail. Then change the context conditions accordingly. For this method to work, contextual difference need to be recognized at appropriate level of abstraction, otherwise to make the right adjustment in the context conditions very large number of experiments will be needed which seems impractical.

The approach for an ideally expert *RPD* agent can straightaway be implemented in existing *BDI* architecture. The preference-based approach can also be implemented using meta-level reasoning capability of JACK agent, by keeping the record of success and failure of plans and ranking the plans accordingly. JACK is an agent development environment produced by Agent Oriented Software Group, Melbourne, Australia. JACK, through its appropriate concepts in the JACK Agent Language supports BDI architecture and helps define beliefs, plans, external and internal events, and capabilities (Agent Oriented Software Ltd., 2008). Whereas, the context-based approach need major modifications in the architecture. Norling (2001) gives preliminary ideas about the methods that can be employed for this type of adaptability in agents.

JACK selects plans on the basis of Boolean tests of context conditions written at the time the agent is designed. These context conditions can not be updated during the run time. To enhance the *BDI* agents to select plans on the basis of preferences, reinforcement learning is introduced. Reinforcement learning is unsupervised learning and that is a requirement in these agents. In reinforcement learning the agent is rewarded or penalised after reaching a state. Q-learning algorithm is selected because of its simplicity. *RPD* enhanced preference-based *BDI* agents are evaluated in simple environments and is not considered sufficiently rich environments for proper evaluation (Norling and Sonenberg, 2002). First, Norling and Sonenberg (2004) plug *BDI* agents, with the help of an interface, to the 'deathmatch' version of the first-person shooting video game "Quake II" which they have previously recommended as a testbed for evaluation of agents having sufficiently rich environments. Then, they develop an enhanced *BDI* agent capable of reinforcement learning using Q-Learning algorithm and interfaced it to the Quake II and found two major problems (Norling, 2004). The first problem is regarding recognition of features of the environment in the game and the second is the unfeasibly large state space. The map in Quake II is represented in a polygon-based structure. Elements of these data structures have been used to render objects on the user's screen. It proved to be very difficult to recognize features of the landscape. State space becomes very large if the raw data out of the game engine is straight away used. Position variable, which is one of many, considered alone increases the state space incredibly, as the position is in three dimensions and each dimension is expressed in real numbers. Although the expert plans elicited out of *SME*'s knowledge may be easily implemented in the agent, the agent does not properly recognize the features of the landscape being used by the expert therefore the expert plans are never selected. Thus evaluation of enhanced agent is not successfully carried out.

Apart from these problems of very large state space making it difficult to adapt the agent using reinforcement learning and the problem peculiar to evaluation there is one basic short coming in *BDI* paradigm in implementing *RPD* model and that is mental simulation. Norling herself writes "The concept of mental simulation has no obvious equivalent in *BDI*, unless one argues that plans themselves do it".

3.8 Composite agent model

Sokolowski (2002) in his early work described composite agent (*CA*) and its similarities with that of the *RPD* model and discussed the ability of the *CA* to implement *RPD* model based on these similarities. Hiles et al. (2002) developed the *CA* as a result of their work aimed at computer generated autonomy. A *CA* is a multi-agent system (*MAS*) based on the concept that human decision making which is a complex phenomenon may be modelled by numerous interactive agents representing various activities involved in a human mind. A *CA* is composed of symbolic agents called symbolic constructor agents (*SCA*) and reactive agents (*RA*). *SCA* observes the external environment and creates an internal picture of the external environment. The reactive agent (*RA*) generates actions for the composite agents driven by the inner environment created by *SCA*. There are multiple *SCA*s and RAs in one *CA*. Each *RA* represents a specific behaviour of the *CA*. Each *RA* is striving to achieve one or more goals assigned to it. These goals are driving the behaviour of the *CA*. In an *RA*, to further these goals there are associative sets of actions. A *CA* has an over all goal. Multiple RAs interact with their own set of actions, and the selection is based on the degree to which these actions achieve the overall goal. The *CA* continues to observe the environment and if the situation changes then a different set of actions is selected.

Sokolowski describes the similarities between *RPD* model and CA. Like an *RPD* model, the *CA* through its *SCA* also senses the external environment, produces an internal representation of the situation, and periodically samples the environment. *CA* is also goal driven. Various goals compete for satisfaction and a dominant set of actions is selected based on the overall goal. The *RPD* model knows what to expect next as the situation unfolds and the decision is implemented. *CA* accomplishes the same by periodically monitoring the external situation as it changes caused by either external effects or as a result of its own actions. About mental simulation he writes "… A *CA* partially accomplishes mental simulation as it performs its goal management process to select the set of actions that it will carry out. However, there is no clear mechanism within the *CA* to modify its existing experiences to provide a better solution. The mental simulation process will most likely need to be enhanced to better replicate role of mental simulation within *RPD*…." (Sokolowski, 2002).

Sokolowski in his later work implemented the *RPD* model based on *CA* (Sokolowski, 2003a), (Sokolowski, 2003b) and (Sokolowski, 2003c). More agents namely Main Agent, Recognition Agent and Decision Agent are introduced in this model. *RPD*Agent's experiences are stored in Minsky's frames. Minsky identified frames as a data structure to hold information about a person's environment. Each frame holds a single *RPD*Agent experience.

F = (C*, G*, A*)

where, F is a frame, C* is a structure containing cues, G* contains goals, and A* contains actions for an experience.

Cues are formed by aggregating the environmental variables associated with that cue. Once the values of all cues have been calculated then they are transformed into fuzzy values. Each case has three fuzzy sets, an unsatisfactory, a marginal, and a satisfactory fuzzy set. Triangular-shaped fuzzy sets have been used. Higher values are more likely to fall in the satisfactory set.

The Main Agent manages the overall system and holds the *RPD*Agent's experience database. The decision process is conducted mostly by the Decision Agent. On receiving a decision request, the existing experience is matched via a look up table. In a case where there is no matching experience then the *RPD*Agent does not have the experience to make a decision. In case a where a match is found then the related information is given to the concerned agent and *SCA* is informed of a pending decision request. *SCA* generates an internal representation of the environment and then instantiates a Decision Agent to manage the decision process. The decision agent making use of its encoded experience proposes a potential decision according to the internal representation of the situation. The most favourable action is the action with the highest 'action value'. The action value of an action is the sum of all cue values associated with an action. A Decision agent (DA) instantiates reactive agent (*RA*) for each goal that *RPD*Agent is supposed to achieve. *RA* evaluates the potential decision with respect to the goal for which it has been instantiated. If potential decision satisfies all goals then it is selected for implementation otherwise *RPD*Agent gets into a negotiation function conducted under the control of decision agent (*DA*) by *RA*. If a negotiation is successful and a compromise above a threshold is reached the decision is rendered otherwise the next potential decision is evaluated. If no decisions adequately satisfy the goals then *RPD*Agent renders a default decision appropriate for the situation.

In this implementation the cues have been developed by aggregating the environmental variables and the same cues have been used for evaluating the potential decision. This method of developing cues for evaluation produces good results for operational level decisions like selecting an approach of attack, deciding on the line and bias of defence, and of course the selection of a site for amphibious landing. The agent has been developed for the same purpose and for an agent with more general tasks further methods will have to be added in generating cues from the environments. The mental simulation in this case is based on the evaluation of potential decision to the degree that it satisfies the main goals. And the degree of satisfaction of a goal is the weighted sum of all the cues associated with that goal. Mental simulation in this implementation is not flexible enough to accommodate all aspects of the mental simulation required of an *RPD* agent.

3.9 *RPD* enabled collaborative agents for simulating teamwork (*R-CAST*)

Yen, Fan, and Sun and others have developed an *RPD* enabled collaborative agent architecture to support human decision making teams (Fan et al., 2005) and (Yen et al., 2006). The architecture for collaborative agents which forms the base on which this *RPD* process is integrated to enhance the decision making ability is known as collaborative agent for simulating team behaviour (*CAST*) (Yen et al., 2001). The decision to communicate between team members is based on decision-theoretic strategy. That means the cost of communicating and the possibility of requirement of the message is considered in calculating expected utility of communicating and also in the same way the expected utility of not communicating is calculated. The decision is made by the agents for the choice with higher expected utility. For *RPD* process the experience is divided amongst agents and the one with the requisite experience or knowledge agrees to make the decision. All *R-CAST* agents maintain a mental picture of the world according to their own beliefs in the knowledge base. The *R-CAST* agent making the decision checks whether the preconditions of the plan that is selected satisfy the knowledge base and if so the agent asserts the plan to see whether the relevant goals are met. This part of the *RPD* process is the mental simulation. It is not clear as to what happens to the beliefs of the agent which is based on the state of knowledge base if the plan does not meet the goals and is rejected. Does the knowledge base go back to the previous stage before the plan is asserted? The knowledge base is stated to be proof preserving and in our opinion it should store the previous state and revert back to it if the plan is rejected for the sake of truth maintenance in the agent's mental model.

3.10 Summary

Computer implementations of *RPD* discussed in this chapter are based on human cognitive models at various levels of abstraction developed on different physiological and psychological theories to include multiple trace memory model, artificial neural network and belief, desire, and intention (*BDI*) cognitive architecture. *RPD* implemented on hybrid models whereby knowledge represented in a psychological model is driven by a physiological model based on neural units is discussed. Two implementations based on multiple agent system and one each on fuzzy logic, context based reasoning and Markov chain models are also discussed. Mental simulation which forms the major part of Level 3 *RPD* is implemented in fuzzy logic, Markov chain, and both of the multiple agent system models. But the scope of the mental simulation is limited and does not cover the complete range of requirements of *RPD* model.

Having reviewed some of the work implementing *RPD* agent we propose a methodology that embeds *RPD* in Soar cognitive architecture. As a first step we develop a simple *RPD* agent to identify essential components that are required for a complete implementation of *RPD* model. This is illustrated in the next chapter with the help of a simple example.

4 A SIMPLE RECOGNITION PRIMED DECISION MAKING AGENT

The aim of the experiments discussed in this chapter is to realize that realistic modelling of human behaviour changes the results of simulations and *wargames* and gives us the opportunity to draw more accurate results from military simulations. The aim is also to learn more about, and to probe the ability of the *RPD* model to provide the decision making model required for the intended command agent to be used in military simulations. In the end of the chapter the features required in a system to implement *RPD* model are also discussed.

The simulations based on analytical methods developed in this chapter also serve the purpose of validating the base line or start point of simulations involving *HBR*.

4.1 Tank battle simulation (3-on-1 combat involving a hidden defender)

For this experiment we have selected a very popular and very well analysed case of three-on-one combat (McNaught, 2002) and (Kress and Talmor, 1999). The basic idea of this vignette is taken from the work of Kunde and Darken (2005). The blue and red forces tactics, information on battle drills and capabilities of weapons and equipment is based on the interviews with the subject matter expert (*SME*) from the OA, Modelling and Simulation Group of Defence Academy, United Kingdom and personal knowledge of the author on the subject.

4.2 Vignette

Foxland and *Blueland* are two neighbouring states, relations have been strained due to territorial disputes and now the hostilities are imminent. *Foxland* is likely to start probing the border defensive positions of *Blueland* and launch a major offensive operation against *Blueland*.

4.2.1 Enemy situation

An enemy troop of tanks, consisting of three red tanks, is advancing as the forward reconnaissance element of the advancing force on the selected avenue of approach.

4.2.2 Friendly situation

There is one blue tank, in hull-down position, on the most likely approach to the *Blueland* main defensive positions. This blue tank is waiting for the advancing enemy tanks.

4.2.3 Mission

Delay the enemy by causing maximum attrition on enemy forward reconnaissance elements.

4.2.4 Description

It is expected that the forward reconnaissance elements consist of a minimum of three red tanks. The defensive position adopted by the blue tank will make it difficult for the red tanks to detect and engage it. Whereas, the red tanks are moving and the blue tank also knows their general direction of approach, therefore, it will have an advantage in detecting and later on engaging red tanks. The blue tank also has the advantage of surprise.

4.3 Characteristics of entities and terrain

As the blue tank is in hull-down position, therefore, the probability of its detection is relatively small. In a situation where enemy tanks are coming up or around a hill they appear and are detected one after the other. For simplicity, in this simulation the terrain has been abstracted to two dimensions and the same effect of tanks coming up or around the hill has been created using the sensor range of the blue tank. The red tank is detected by the blue tank only when the red tank comes within the sensor range of the blue tank. The sensor range of the blue tank is depicting the edge of the hill where the red tanks are appearing and then they remain visible to the blue tank. To give the effect of the red tanks coming up a hill within the firing range of the blue tank, the firing range of the blue tank is also kept equal to the sensor range. Sensor and firing ranges of blue tank are kept at 1200 meters.

4.4 The problem in existing computer generated forces

In modular semi-automated forces (*ModSAF*), and other existing computer generated forces (*CGF*), the behaviour of the tank commander is not very realistic. When operating without human intervention the very first action of the simulated tank commander after detecting an enemy tank, within the firing range of own tank, is to engage it. In *ModSAF,* the usual setting for most of the operations is "shoot on sight". However, for the vignette described in Section 4.2 there is another option that may be selected i.e., "no fire until ordered" but to use this option in a *ModSAF* simulation, human intervention is necessary.

4.5 Factors considered by a human tank commander in defence

Existing *CGF*s and semi-automated forces (*SAF*s) in simulations start to shoot on sighting enemy tanks. Some of them also check their firing ranges before deciding to engage the enemy tank. Whereas, an experienced human tank commander may or may not engage an enemy tank on its detection even though that may be within the firing range of his tank. Many questions immediately cross his mind on an event of enemy tank sighting. Following are some example questions that will immediately pop up in the mind of the tank commander when his tank is deployed in a defensive position:-

- Is this one the only enemy tank?
- Are there any more tanks following it?
- Is it the most advantageous time to engage them?
- Are they going to detect me?
- What will be the reaction of other enemy tanks after I engage the first tank?
- Is it feasible to engage them at all?
- Do I have to delay them?
- Will I be reinforced?
- Do I have sufficient ammunition to take on the forthcoming battle?

In this particular situation, a real tank commander having seen one enemy tank, would expect additional tanks and would therefore probably wait longer to begin surprise fire than would a simulated commander. If he fires before the other tanks round the corner (or come up the hill), they will be warned and may try to outflank him, seek cover, use artillery fire, choose a different path, etc.

4.6 Analytical models

In order to verify the basic simulation we compared our results with analytical models. Mainly the Lanchester models of attrition have been used, although, the Markovian model has not been used but it has been very briefly discussed in order to show that as the battle is realistically modelled with more details then the outcome of the battle reduces its dependence from numbers and fire power to other factors like use of terrain and battle tactics etcetera.

British Engineer F. W. Lanchester in 1914, published a paper describing a model of attrition process in battle (Lanchester, 1916). The attrition process in combat in this model is based on a pair of linked differential equations. The Lanchester equations are based on the assumption that the attrition suffered by either side in battle is a function of the numerical strengths of the opposing forces involved and the efficiency of their respective weapons. These deterministic Lanchester equations assume that each unit on each side is within the weapons range of all units on the other side, each firing unit is well aware of the location and condition of all enemy units so that the fire is immediately shifted to a new target when the previous target is killed, and the fire is uniformly distributed over all surviving units. There are two basic Lanchester laws: one is for attrition of forces in a direct fire battle called the deterministic Lanchester square law and the other for the attrition of forces in an indirect-fire battle called Lanchester linear law. There are modern variations from the original model that are in use in present combat models including exponential stochastic Lanchester model. In this chapter the deterministic Lanchester square law and stochastic Lanchester model are discussed.

4.6.1 Deterministic Lanchester (DL) square law

The two sides are designated blue and red.

b, r = number of surviving units on the blue side and red side respectively at time t.

B, R = initial number of units at time $t = 0$.

β = the rate at which single blue unit can kill red units.

ρ = the rate at which a single red unit can kill blue units.

The Lanchester equations are as follows:

$$\frac{db}{dt} = -\rho r \qquad \text{and} \qquad \frac{dr}{dt} = -\beta b$$

These equations may be solved with respect to time to give the number of surviving units on each side at time *t* after the start of the battle. However, the more usual form of the Lanchester direct-fire model called Lanchester Square Law of attrition for direct-fire battle is the solution of these equations with time eliminated as follows:

$$\beta(B^2 - b^2) = \rho(R^2 - r^2) \qquad \text{Equation 4.1}$$

In order for the firefight to be at parity in *DL square law*, the following condition must remain valid during the battle:

$$\frac{b}{B} = \frac{r}{R}$$

Substituting the above condition in $\beta(B^2 - b^2) = \rho(R^2 - r^2)$

Equation 4.1 yields the following requirement for parity:

$$\beta B^2 = \rho R^2$$

Thus, from the above equation, the effectiveness (kill rate) of the single blue combatant for parity in direct-fire battle must be given by,

$$\beta = \rho R^2 \qquad \text{Equation 4.2}$$

Therefore, for parity in three-on-one battle, blue is required to be nine times more effective.

4.6.2 Exponential stochastic Lanchester (ESL)

In stochastic Lanchester model combatants on both sides assume to have exponentially distributed interfering times. Taking the same notation as that of Section

4.6.1, the probability of red and blue kill and the mean time distribution to next kill is defined as follows:

- Blue forces kill red at rate = βb
- Red forces kill blue at rate = ρr
- Probability of blue killing red = $\frac{\beta b}{\beta b + \rho r}$
- Probability of red killing blue = $\frac{\rho r}{\beta b + \rho r}$
- Time to next kill has a negative exponential distribution with a mean = $\frac{1}{\beta b + \rho r}$

When two sides are at parity in *DL* square law model the above equation predicts mutual annihilation, whereas, parity in a stochastic model would imply an even chance of victory for either side.

McNaught (2002) suggests that in ESL model a different square law exists, and for the two forces to be at parity in this model requires the following equation to be satisfied:

$$\beta(B^2+B) = \rho(R^2+R) \qquad \text{Equation 4.3}$$

Therefore, for parity in three-on-one battle McNaught (2002) suggests that blue is required to be six times more effective but in fact this ratio is higher and parity exists approximately at an effectiveness ratio of 7.5 (Wand and Bathe, 2008).

4.6.3 Markovian model

The Markovian model (McNaught, 2002) takes into account the detection process and gives first shot advantage to the hidden defender, shows that in order to have parity in three-on-one combat involving a hidden defender the blue has to be four times more effective than the red.

4.7 The simulation

A Simulation has been developed in Java programming language based on the exponential stochastic Lanchester (*ESL*) model (Figure 4.1). This simulation is designed to investigate the effect of the introduction of intelligent-like-behaviour in the combatants on the outcome of the battle. The components of the simulation are discussed in the succeeding paragraphs.

To make the comparison simpler, we have assumed the mean inter-firing times to be equal for both red and blue forces and the switching time from one target to another is assumed to be zero which simply means that the effectiveness ratio can be taken as the ratio of the single shot kill probabilities (*SSKP*).

4.7.1 Blue tank commander (*BTC*)

The *BTC* makes decisions for the blue tank. It has a long term memory (*LTM*) that contains experiences which consists of the situational elements, courses of action and expectations. The *BTC* develops *present situation* which is a set of values of situational elements from the information available in the environment. The *present situation* contains information about the red tanks such as their status i.e. whether dead or alive, their distance from the blue tank, whether moving or static, and whether firing or not firing.

The basic idea, of the structure of this *LTM* consisting of experiences, is taken from the work of Warwick *et al.* (2001). These experiences are developed with the help of a subject matter expert (*SME*). *SME* is asked to give the most suitable course of action and expectation(s) for a given present situation. All possible situations that may arise in this scenario are included and every set of values of situational elements called *present situation* in this thesis is associated to a course of action and the expectation(s) and is stored in the *LTM* as an experience.

BTC gives the *present situation* to the *LTM* whose experiences are indexed to the elements of the *present situation*. Based on the *present situation* an experience is retrieved. For simplicity it has been assumed that each set of values of situation elements retrieves a single experience which corresponds to a single course of action. In this model only courses of action and expectations are retrieved from the memory, goals and cues have not been considered.

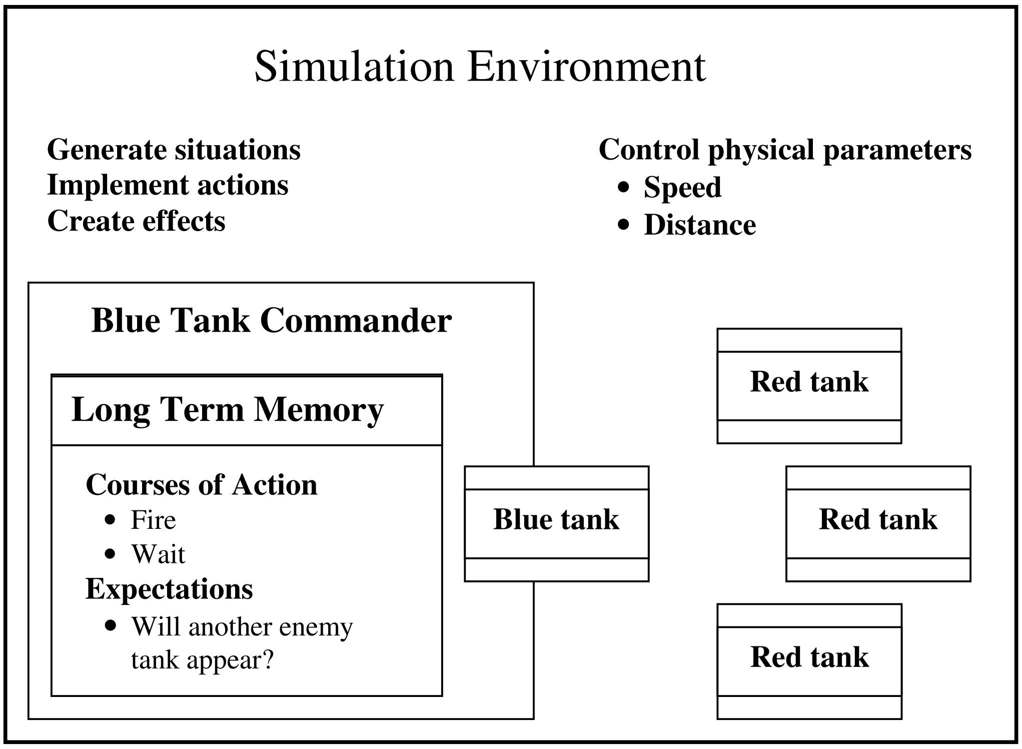


Figure 4.1 Three-on-one tank battle simulation

4.7.2 Simulation environment

The Simulation environment contains all entities, generates situations, implement actions taken by each entity and creates the effects of actions of all the entities present in the environment (Banks, 2005). It controls the physical parameters like time to engagement and the time taken by the red tanks to travel some distance based on their speed. It also decides whether a tank is killed or otherwise when fired at, based on the *SSKP* of the shooting tank. If a tank consumes all its ammunition then the environment does not allow the tank to fire any more rounds. However, in the experiments discussed in this chapter this limitation on tank ammunition is not imposed and it can fire as many rounds as required to end the battle.

4.7.3 Time to engagement

When a tank engages a new target there is a certain time required for detecting, identifying, aiming and firing and also there is travel time of the projectile that it takes to reach the target. And when it reaches the target it either hits or misses the target. Time to engagements may be modelled with the help of variety of probability distributions. In this experiment we have used exponential, triangular and rectangular distributions. The exponential distribution has been used to verify the results of the simulation with that of the analytical results and the triangular and rectangular distributions have been used to observe the change in results for distributions other than the exponential distribution. We have defined two types of time to engagement in this experiment. The first is the time to initial engagement and the second is time to next engagement. The time to next engagement need to be defined because during the course of a battle due to a variety of reasons a target that was previously fired at is engaged again.

4.7.3.1 Time to initial engagement

Time to initial engagement is defined as the time taken by a tank when it engages an enemy tank for the first time or a second time only if it disappears in an area that provides cover from observation and then reappears at a different location more than three hundred metres away. Time to initial engagement is longer than the time to next engagement. It has been modelled with the help of two types of distribution, the triangular and the exponential distributions (Lecture notes, ESD, 2004).

First the time to initial engagement based on the ***triangular distribution*** is discussed and we assume a probability distribution as shown in Figure 4.2.

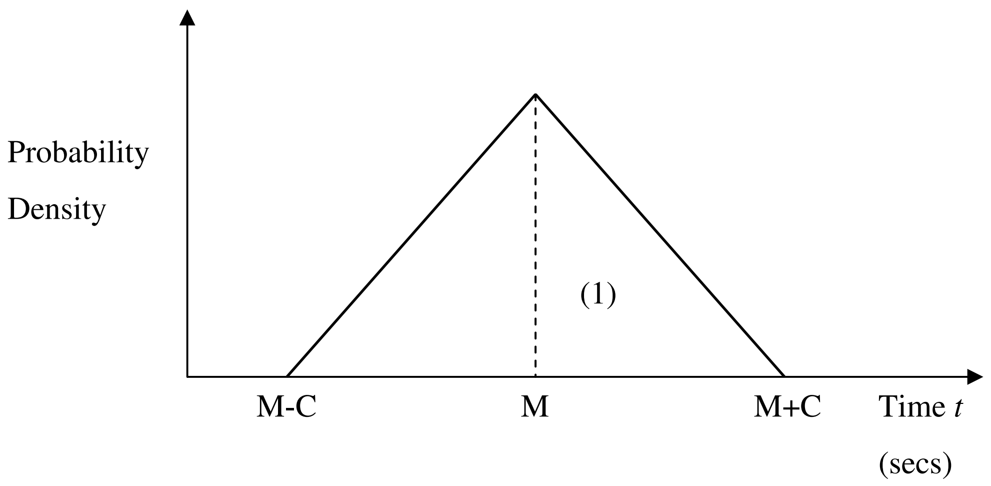


Figure 4.2 Time to initial engagement – Triangular distribution

The theory of probability requires that the area under the probability density function curve must be unity. The mathematical expression for this probability density function is:

$$f_1(t) = \begin{cases} \dfrac{1}{C} + \dfrac{(t-M)}{C^2} & M - C < t < M \\ \dfrac{1}{C} + \dfrac{(M-t)}{C^2} & M < t < M + C \\ 0 & elsewhere \end{cases} \qquad \text{Equation 4.4}$$

Where M is the mean time to engagement and *C* is the spread of time to engagement from the mean. The mathematical expression for related cumulative distribution functions is:

$$F_1(t) = \begin{cases} \dfrac{t}{C} + \dfrac{(t^2 - 2Mt)}{2C^2} + \dfrac{(C-M)^2}{2C^2} & M - C < t < M \\ \dfrac{t}{C} + \dfrac{(2Mt - t^2)}{2C^2} - \dfrac{M(2C+1)}{2C^2} + \dfrac{1}{2} & M < t < M + C \\ 0 & elsewhere \end{cases} \qquad \text{Equation 4.5}$$

In the above distribution function the expression is in such a form that an analytical expression can be formed from which for any generated random number, the variate time *t* may be calculated directly (Rubinstein, 1981), using the following equation:

$$t = \begin{cases} M - C \pm C\sqrt{2R} & M - C < t < M \\ M + C \pm C\sqrt{2(1-R)} & M < t < M + C \end{cases} \qquad \text{Equation 4.6}$$

Where *R* is the random number generated by the random number generator of the system simulator.

Next time to initial engagement using exponential distribution is discussed and we assume the probability density function to be a negative exponential distribution with a mean of $\frac{1}{A}$, as shown in Figure 4.3. The mathematical function is follows:

$$f(t) = Ae^{-At} \qquad t > 0 \qquad \text{Equation 4.7}$$

The mathematical expression for cumulative density function is as follows:

$$F(t) = 1 - e^{-At} \qquad \text{Equation 4.8}$$

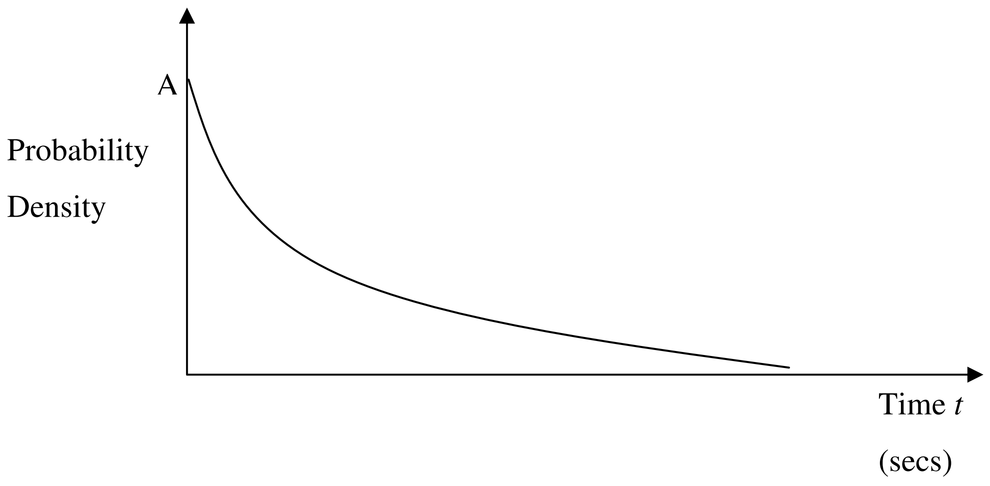


Figure 4.3 Time to initial engagement – Exponential distribution

For generated random numbers, *R*, the variate t may be calculated directly using the following expression:

$$t = \frac{1}{A}\log_e \frac{1}{1-R} \qquad \text{Equation 4.9}$$

4.7.3.2 Time to next engagement

When it is comparatively easier to detect an enemy tank then less time is required to engage a target, therefore, time to next engagement is shorter than time to initial engagement. This situation arises when either a tank fails to defeat its target and thus it fires again against the same target or the second target is close to the first target. In this simulation if the next target is within 300 meters of the first target then it is considered as close. Time to next engagement is modelled with the help of rectangular and exponential probability distributions.

In case of the ***rectangular distribution*****,** we have assumed the time to next engagement to have a probability density function as shown in Figure 4.4.

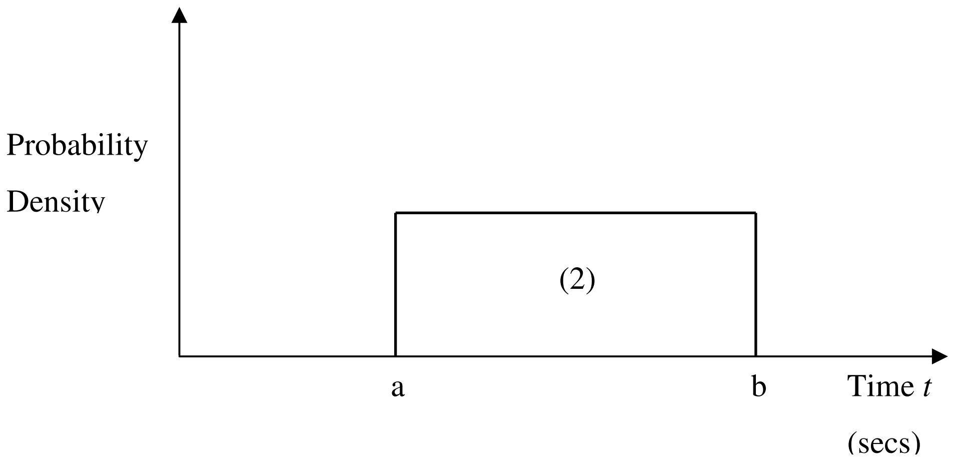


Figure 4.4 Time to next engagement of the same target – Rectangular distribution

Keeping the area under the rectangle unity, in accordance with probability theory, the mathematical expression for this probability density function is as follows:

$$f_2(t) = \frac{1}{b-a} \qquad a < t < b \qquad \text{Equation 4.10}$$

Where *a* is the minimum time a re-engagement takes and *b* is the maximum time.

$$F_2(t) = \frac{(t-a)}{b-a} \qquad a < t < b \qquad \text{Equation 4.11}$$

In the above distribution function also the expression is in such a form that an analytical expression can be formed from which for any generated random number, the variate time *t* may be calculated directly, using the following equation:

$$t = R(b-a) + a \qquad a < t < b \qquad \text{Equation 4.12}$$

Where *R* is the random number generated by the random number generator of the system simulator.

The time to next engagement for the ***exponential distribution*** is calculated using Equation 4.9; only the mean time in this case is half of the mean time for time to initial engagement.

The above distribution functions for time to next engagement are also used for engaging new tanks with in 300 meters distance of the previously engaged tank. If the distance of a new detected tank is more than 300 meters in that case the time to engagement is determined from the distribution function of time to initial engagement.

4.7.3.3 Speed of red tanks

It has been assumed that the tanks are moving at a speed of 36 Km per hour and thus cover a distance of ten metres in one second.

4.7.3.4 Decision whether a tank is killed or not

The decision whether a tank is killed or not when fired at is based on the *SSKP* of the tank that is firing. Whenever a tank fires, a random number '*R'* ranging from 0 to 1 is generated based on uniformly distributed probability function. If this '*R'* is less than the *SSKP* of the tank that is firing then the tank is declared killed. It is not killed otherwise.

4.7.4 Validation

To validate the simulation, the simulation is first tested on one-on-one battle. We know from the analytic solution in case of ESL (Section 4.6.2 is referred) the stochastic parity exists for one-on-one battle if the effectiveness of the combatants is the same. In one-on-one battle there is no switching time as the battle terminates when one of the two combatants is destroyed. Therefore, the effectiveness depends only on inter-firing times and *SSKP* of the combatants and for equal inter-firing times the effectiveness only depends on *SSKP*. For both red and blue tanks, the inter-firing times, firing and sensor ranges, and *SSKP*s are kept equal. The inter-firing time has a negative exponential distribution with mean at ten seconds and is calculated from Equation 4.9. The inter-fire time is kept the same for both initial and subsequent engagements throughout this battle. The SSKP and firing range are 0.5 and 1200 meters respectively. First 100, 500, 1000, and 10000 simulations are run for exponential probability distribution to validate the simulation with respect to the analytical solution for ESL. The results of simulations are shown in Figure 4.5, which clearly demonstrate that the stochastic parity exists as suggested.

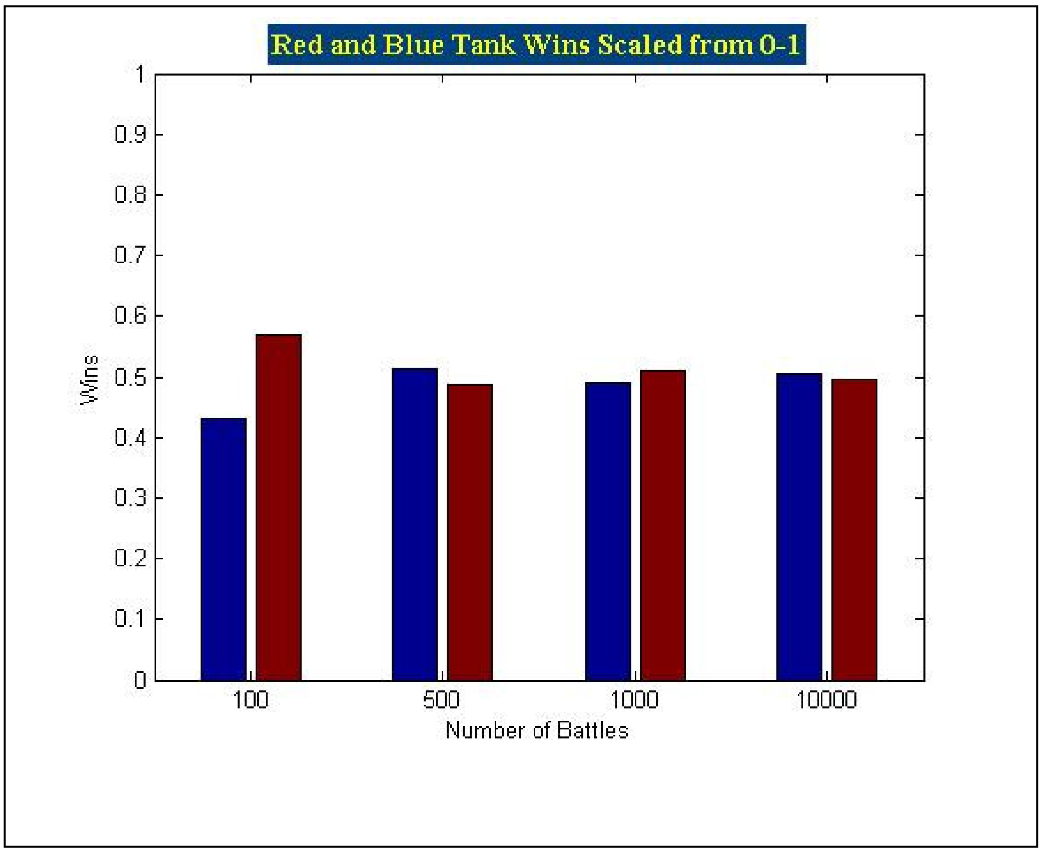


Figure 4.5 Blue and red wins for one-on-one battle

The next set of simulations is run to compare the triangular and exponential probability distributions in order to validate the model with triangular distribution. Characteristics of the combatants and the probability distributions of time to engagement are shown in Table 4.1. The values used for this simulation are the same as that of the first simulation only the upper and lower limits in case of triangular distribution is specified in addition to the mean value which is the same as that of the exponential distribution. Similar to the previous simulation, the inter-firing time is kept the same for both initial and subsequent engagements throughout the battle.

Table 4.1 Characteristics of combatants for one-on-one simulation

		ESL		Triangular	
		Red	Blue	Red	Blue
Number of tanks		1	1	1	1
Interfering times	Mean	10	10	10	10
	Spread			±4	±4
SSKP		0.5	0.5	0.5	0.5
Firing range	Metres	1200	1200	1200	1200

Five sets of 100 simulations each for both probability distributions are run and the results are shown in Table 4.2.

Table 4.2 Comparison of exponential and triangular inter-firing time distributions

Exponential		Triangular	
Blue wins	Red wins	Blue wins	Red wins
46	54	57	43
55	45	44	56
48	52	51	49
45	55	46	54
50	50	42	58
Mean = 48.5	Mean = 51.2	Mean = 48	Mean = 52

This test is aimed at validating triangular distribution as an alternative to exponential distribution because triangular distribution is comparatively easier to handle in computer simulations than exponential distribution. The exponential distribution has an infinite tail that causes problems in developing the simulations. A cut-off time is required to be set for exponential distribution in order to get a finite inter-fire time.

It is evident from the results that the triangular distribution can be used as an alternate to the exponential probability distribution.

The third test on the simulation is aimed at validating the simulation on Equation 4.3 for three-on-one battle. The validation is done by keeping the equal inter-firing times and equal sensor and firing ranges but the *SSKP* of blue tank is 7.5 times more than the red tank for stochastic parity in case *stochastic exponential Lanchester* (*ESL*) as suggested by (Wand and Bathe, 2008) discussed in Section 4.6.2. Similar to the previous simulation firing range of both tanks is 1200 meters and inter-firing time is based on negative exponential distribution with mean at ten seconds. In this simulation the *SSKP* of combatants represent their effectiveness as we have assumed the switching time to be equal to zero and only the inter-firing time is considered even when the targets are switched. We ran 4000 simulations and the total number of blue and red wins turned out to be 1868 and 2132 respectively and the corresponding probabilities of win are 0.47 and 0.53.

After validating the simulation with the help of the analytical solution, we changed the conduct of battle in the simulation and now the red tanks move 100 metres and then stop to engage the blue tank. Red tanks can not fire during move. We ran 100 simulations for exponential probability distribution and found that stochastic parity results when the single tank is six times more effective than each of the three attacking tanks. We ran another 100 simulations with the same settings and only changed the distribution from exponential to triangular and found out that in this case also the stochastic parity exists when the single tank is approximately six times more effective.

The simulation is now run based on our vignette for both exponential and triangular inter-firing time distributions. These are run for a combination of forces with intelligent-like and unintelligent-like behaviours opposing each other. One hundred simulations are run for each case and the results are analyzed. This is done in order to highlight the concept that intelligent-like behaviour can make a difference in the outcome of a battle simulation given the same terrain, forces, equipment and situation.

4.7.5 If both red and blue sides do not have intelligent-like behaviour

The three red tanks start moving towards the blue tank. The first red tank enters the sensor range of blue tank, which is also the firing range as described earlier, and the blue tank detects it. At this moment the *BTC* is faced with a decision point. *BTC* looks at the situation and sends a probe based on the *present situation* to his memory and finds out that he has encountered this situation before and the best course of action is to fire, *BTC* does so because it does not have intelligent-like behaviour.

We ran 100 simulations each for exponential and triangular probability distributions. The time to initial and next engagements are calculated based on Equation 4.9 for exponential probability distribution, mean times for initial engagement and next engagement are 10 and 5 seconds respectively. Time to engagement for the simulation set associated with triangular distribution is calculated using Equation 4.6 and Equation 4.12. Time to initial engagement, in this case, is calculated using triangular distribution whereas; time to next engagement is calculated using rectangular distribution. Mean inter-firing time is 10 and the spread is ±4 seconds for triangular distribution. And the maximum and minimum inter-firing times are 3 and 1 second(s) respectively for rectangular distribution. Switching time for targets within a distance of 300 meters is assumed to be zero and the inter-fire time in this case is equal to time to next engagement. For switching targets with a distance of more than 300 meters the inter-fire time is increased and is equal to time to initial engagement.

In this simulation the red tanks are appearing one after the other with a gap of 50 meters in between them. *BTC* shoots on sighting the first red tank but the red tanks keep moving in the same direction even after realizing that the tank ahead of them is engaged (that is the present state of simulations, e.g., *ModSAF*). As the red tanks are appearing one after the other, therefore, it gives an advantage to the blue combatant. For the same effectiveness ratio the probability of winning the battle for blue improves from parity to 0.62 and 0.61 for exponential and triangular distributions respectively.

4.7.6 If the red side has intelligent-like behaviour and blue does not

BTC shoots again on sighting the first red tank. But in this case, the rest of the two red tanks after realizing that the tank ahead of them is being engaged try to manoeuvre and attack from the flanks without getting into the blue tanks killing area if they hear the first blue shot fired in time. If the red tanks are successful in coming from the flanks they compromise the advantage of blue's defilade position and reduce their detection times. Again this situation was run for two choices of probability distributions.

The time to initial and next engagements are calculated based on Equation 4.9 for exponential probability distribution, mean times for initial engagement and next engagement are 10 and 5 seconds respectively. Time to engagement for the simulation set associated with triangular distribution is calculated using Equation 4.6 and Equation 4.12. Time to initial engagement, in this case, is calculated using triangular distribution whereas; time to next engagement is calculated using rectangular distribution. Mean inter-firing time is 10 and the spread is ±4 seconds for triangular distribution. And the maximum and minimum inter-firing times are 3 and 1 second(s) respectively for rectangular distribution. Switching time for targets within a distance of 300 meters is assumed to be zero and the inter-fire time in this case is equal to time to next engagement. For switching targets with a distance of more than 300 meters the inter-fire time is increased and is equal to time to initial engagement.

Because of this advantage to the red and disadvantage to the blue combatant, the blue reduces its probability of winning the battle to 0.41 and 0.35 for exponential and triangular distributions respectively.

4.7.7 If both red and blue sides have intelligent-like behaviour

The three red tanks start moving towards the blue tank. The first red tank enters the sensor range of blue tank, which is also the firing range as described earlier, and the blue tank detects it. At this moment the *BTC* is faced with a decision point. *BTC* looks at the situation and sends a probe based on the *present situation* to his memory and finds out that he has encountered this situation before and the best course of action is to hold fire and the expectation is another red tank appearing after this tank in a few seconds. Therefore, *BTC* holds fire and waits for another tank. At this point in time the blue tank may be engaged by the red tank as it is also in its firing range. Therefore the decision to hold fire and wait depends on the personality of this particular *BTC*. If he has been acting bravely in the past and of course considering his hull-down position in *present situation*, he would have experiences in his memory of holding fire for greater advantage of trapping more red tanks. But if the *BTC* has been risk averse then the course of action in his memory would be to engage the very first red tank. In this case, we take him to be a risk-taking commander and he decides to hold fire and expects another red tank.

In this simulation, the red tanks are also intelligent and they will try to manoeuvre and outflank the blue tank if they detect the blue tank either by observation or if blue tank fires. But the blue tank holds fire and it is very difficult for them to detect the blue tank due to its hull-down position. Therefore, after few seconds, another red tank appears within the firing range of blue tank. *BTC* sends a probe based on this situation to the *LTM* and finds it to be a typical situation, with a course of action to hold fire and expectation of another tank appearing after few seconds. When the third red tank also gets in the firing range of blue tank, *BTC* engages the first red tank, which is at the shortest threatening distance.

The time to initial and next engagements are calculated based on Equation 4.9 for exponential probability distribution, mean times for initial engagement and next engagement are 10 and 5 seconds respectively. Time to engagement for the simulation set associated with triangular distribution is calculated using Equation 4.6 and Equation 4.12. Time to initial engagement, in this case, is calculated using triangular distribution whereas; time to next engagement is calculated using rectangular distribution. Mean inter-firing time is 10 and the spread is ±4 seconds for triangular distribution. And the maximum and minimum inter-firing times are 3 and 1 second(s) respectively for rectangular distribution. Switching time for targets within a distance of 300 meters is assumed to be zero and the inter-fire time in this case is equal to time to next engagement. For switching targets with a distance of more than 300 meters the inter-fire time is increased and is equal to time to initial engagement.

Red tanks after realizing that they are being engaged try to manoeuvre and attack from the flanks, but this may not be possible as they are already in the *killing area* of the blue tank and they can not disengage as they did in the previous cases. The other two red tanks are within 300 meters of each other so the blue tank will engage them with shorter inter-firing, i.e. ‘time to next engagement’.

BTC after each change in situation keeps probing his memory for recognition of situation and related courses of action and expectancies and keep testing the expectations to find an anomaly and then get back to his memory for recognition of a new situation. Red tanks engage the blue tank as they detect it. The time to engagement and firing procedure for red tanks is exactly the same as explained above for blue tank.

Because in this case the red combatants can not avail this advantage of outflanking the blue combatant hence blue increases its probability of winning the battle from parity (i.e., 0.50) to 0.57 and 0.56 for exponential and triangular distributions respectively. The results are summarised in Table 4.3. It is evident from the results that different attacking formations and different battle strategies change the out come of the battle.

Table 4.3 Summary of simulation results of *simple RPDAgent*

	Exponential distribution		Triangular distribution	
Intelligent	Blue wins	Red wins	Blue wins	Red wins
None	62	38	61	39
Red	41	59	35	65
Both	57	43	56	44
*Note: Stochastic parity exists (which means P(win) = 0.50 for each force) in 3-on-1 battle with single combatant suggested to be six times more effective when the three tanks move 100 metres towards the single				

This experiment also includes the case when blue side has intelligent-like behaviour while the red side does not. Because of the better decision making of blue commander all the red tanks get into the killing area and in this situation no matter what is the behaviour of red tanks the results will not be different. Because the red tanks can not disengage themselves from the blue once they are inside the killing area.

4.8 Conclusions

The outcomes of constructive military simulations are likely to change if realistic human behaviour is incorporated in these simulations. This computer implementation of *RPD* model works for simple problems and need to be developed and experimented for complex problems.

As the main focus of this research is human behaviour representation therefore, more emphasis should be laid on realistic modelling of human cognition, decision making and learning and less on modelling physical parameters in order to cover more aspects of the central topic of research with sufficient depth. Therefore, the physical parameters may be abstracted and should only be sufficiently modelled to provide proper context to the command agent.

A long term memory and a few ‘if then’ rules will not suffice for an *RPD*Agent to operate successfully in a complex environment, and the agent would require a proper cognitive architecture. The decision making in complex situations demands that the agent is able to recognize the context of the situation, keep more than one goal in mind and make an effort to select an action to satisfy all of them at the same time it is also necessary to derive cues from the elements of the situation presented to the agent and to have a truth maintenance system in the short term memory to keep a valid picture of the whole situation at all times, a long term memory as before to keep all the rules applicable to the problem domain, an inference engine and an architecture to develop a mental model to evaluate proposed actions. The *Soar* cognitive architecture offers most of the required features and is discussed in the next chapter.

5 SOAR

Soar is a symbolic cognitive architecture for general intelligence (Laird et al., 1987). It has also been proposed by Newell as a suitable candidate for unified theories of cognition (UTC) in the series of *The William James Lectures* in 1987 (Newell, 1990). *Soar* is a forward chaining, parallel rule matching and parallel rule firing production system. *Soar* uses an associative mechanism to identify knowledge relevant to the current problems with the help of an extremely efficient symbolic matcher. *Soar* employs a computationally inexpensive truth maintenance algorithm to update its beliefs about the world. Automatic *sub-goaling* gives *Soar* agents a meta-level reasoning capability and enables task decomposition. *Sub-goals* are created due to *impasses*. *Impasse* in *Soar* is the architecturally detected lack of available knowledge. All types of learning in *Soar* are through a single phenomenon called chunking. Chunking is a form of explanation-based generalization. Chunks are the cached results of *sub-goals. Soar* is capable of building autonomous intelligent agents that interact with complex environments inhabited by other intelligent agents and humans. *Soar* has been used to develop intelligent agents for small as well as large scale military simulations (Hill et al., 1997), (Jones et al., 1999) and (Wray et al., 2005).

In this chapter the basics of the underlying concepts and functioning of various mechanisms in *Soar* are discussed that are intended to be used in the implementation of the model. After giving an overview of *Soar*, the architecture, applications, and improvements in Soar are discussed. The discussion on architecture includes working memory, reasoning cycle, conflict resolution and learning in *Soar*. Most of the material in this Chapter is taken from "*The Soar 8 Tutorials 1* – 8" (Laird, 2006a) and *Soar User's Manual Version 8.6 Edition 1* dated 18 May 2006 (Laird, 2006b).

5.1 An overview of *Soar*

Soar is based on the Problem space hypothesis. In a problem space, there is an initial state; there are operators that change the current state, and a desired state. Every task is accomplished by attaining a goal. The goal is to reach the desired state. Thus, every task is achieved through a search in the problem space for the desired state by selecting and applying operators. When there is sufficient knowledge available to exactly know which operator to select at each step then the routine behaviour emerges.

This routine behaviour is usually represented procedurally, but not in *Soar*, where all problems are represented in problem spaces.

Decisions in *Soar* are taken to search in the problem space, e.g., selection of operator, selection of state, etc. If sufficient knowledge exists and can be immediately brought to bear then a decision is straight away taken. Otherwise, a *sub-goal* is created to make a decision. If there are three proposed operators and the knowledge to select one out of them is not immediately available then a *sub-goal* to select an operator is set up. *Sub-goals* in *Soar* can be setup for any decision for which sufficient knowledge is not available. A *sub-goal* is setup to search for information in order to make the required decision. Further *sub-goals* can also be setup from one *sub-goal*, thereby, forming a tree of *sub-goals* and problem spaces.

The long term memory (*LTM*) containing long term knowledge is organized as a production system. Both the task implementation knowledge and the search control knowledge are stored in *LTM* as production rules. Production rules are condition-action pairs. The declarative knowledge which is examined by the productions is available in the working memory (*WM*). In *Soar*, *WM* is the same as short term memory (*STM*). WM of Soar has been explained with the help of an example in Section 5.2.1. However, these data structures that take the form of declarative knowledge are also stored in the *LTM* as production rules. When productions fire the actions of these production rules produce these data structures in *WM*. The data structures in *WM* are formed with the help of working memory elements (*WME*). A *WME* is an identifier, attribute, and value triplet.

Unlike other production systems, *Soar* fires all production rules that are satisfied without any conflict resolution. A production is satisfied when its antecedents match the declarative knowledge, available in the form of *WME*s, in *WM*. Productions can only add *WME*s. Modification and removal of *WME*s is carried out by the architecture itself.

The search control knowledge is transferred to *WM* from *LTM* through firing of production rules containing preferences. The preferences give the behaviour to *Soar* in its current situation. The situation is defined by problem space, a current goal, state and operator. The preferences take one of these forms: acceptable, reject, better, best, worse, worst, and indifferent. Better and worse preference represent comparison between two items. The decision procedure is independent of domain knowledge and it interprets these preferences to select the next action.

Impasses in *Soar* are architecturally detected lack of available knowledge to continue problem solving. Thus an impasse stops problem solving as *Soar* does not know what to do next and creates a *sub-goal* to overcome an impasse. All *sub-goals* in *Soar* are created and maintained by the architecture and therefore, this process is named as automatic *sub-goaling*. Automatic *sub-goaling* is an important feature of *Soar* as it forms the basis for many other useful features of *Soar*. If *Soar* can not accumulate sufficient knowledge to proceed with problem solving it stops.

The architecture maintains a goal stack and keeps monitoring all the active goals in the goal hierarchy and it immediately detects the termination of a goal. After detection of termination *Soar* proceeds from termination point, that will be a level higher from where the *sub-goal* is set up. When the goal terminates all the working memory elements related to it are automatically removed.

As the Soar is proposed as a cognitive architecture for general intelligence therefore Soar realizes all weak methods. Weak methods are general-purpose search mechanisms trying to string together elementary reasoning steps to find complete solutions. Such problem solving approaches are called weak methods because, although general, they do not scale up to large or difficult problems (Russell and Norvig, 2003). The alternative to weak methods is to use the more powerful, domain specific knowledge that allows large reasoning steps. The *Soar* realizes all weak methods e.g., hill climbing, means-ends-analysis, etc., through productions provided for search control. Due to the structure of *Soar*, it's not necessary to procedurally represent the method to employ any of the weak methods. If knowledge exists for evaluation of operators, and better operators are given larger numerical values or better preference symbolically then *Soar* automatically exhibits a form of hill climbing.

Soar learns by caching the results of its *sub-goals* as productions and the process is named chunking analogous to human cognition. Chunking is a form of explanation-based generalization.

5.2 Architecture

The behaviour is an integration of architecture and content. The content consists of the knowledge of task implementation and search control. The architecture performs the functions of; goal creation, goal maintenance, goal termination, decision making, memory management and learning. A higher level view of *Soar* is shown in Figure 5.1.

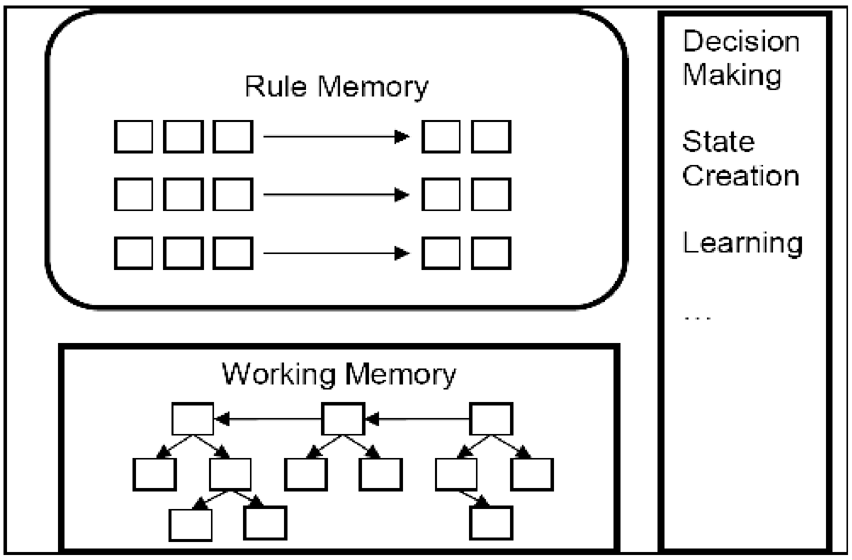


Figure 5.1 A higher level view of *Soar* Architecture [(Laird, 2006a) with permission]

5.2.1 Working memory

The working memory holds the complete processing state for problem solving in *Soar*, to include sensor data, intermediate calculations, objects in the state, goals and operators. The graph in Figure 5.2 represents the working memory of a *Soar* agent that has three objects in its world. A block named 'A' on top of another block named 'B' on top of a table named 'Table'.

The structure of working memory is in the form of a connected graph, consisting of nodes, e.g., *S1, B1, T1, and blue*, and edges or links, e.g., *ontop, name, colour*, and *type* (see Figure 5.2). There are two types of nodes in this graph. One type of nodes is called *identifier* and they have links emanating from them and are non-terminal nodes such as *S1, B1,* and *T1.* While the other type of nodes is called *constant* and they are terminal nodes with no further links emanating from them such as *blue*. The edges or links are called *attributes*. The working memory is in the form of an identifier, attribute and value triplet called working memory elements (*WME*). The value in a *WME* may also be an identifier connecting to another attribute. Every *WME* is either directly or indirectly connected to a state symbol, in Figure 5.2 the state identifier is *S1* and all the *WME*s are eventually connected to it.

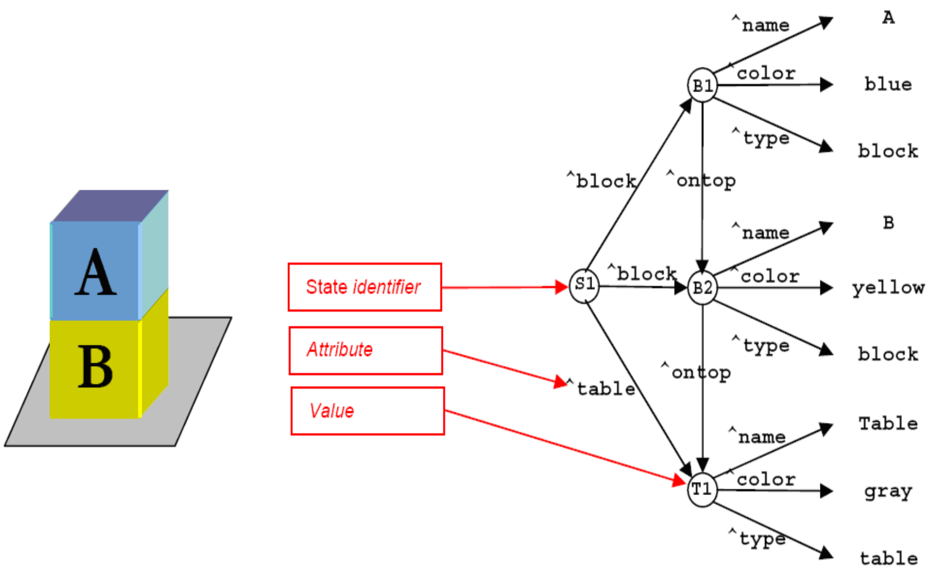


Figure 5.2 Structure of working memory [(Laird, 2006a) with permission]

An *object* in *Soar* is defined as a collection of *WMEs* that share the same first identifier. The *object* in working memory is usually a representation of a physical object in the world of the agent. In Figure 5.2, the identifiers *S1* and *B1* are objects. One object may contain other objects as in the case of *S1* and *B1*. The identifier *B1* is an object of type *block*, name *A*, and colour *blue*, and is *ontop* of another object *B2*. There are some working memory structures as shown in Figure 5.3 that *Soar* creates automatically. Although, this part of the memory is not shown in Figure 5.2 but the agent will have this structure also. The attribute *io* pointing to identifier *I1* in state *S1* appears only on the top state i.e., the first state that *Soar* agent creates. Whereas, the attributes, *super-state* and *type*, appear in all states that are created by *Soar*. The output-link *I2* and input-link *I3* are both identifiers as they may have further augmentations connected to them later.

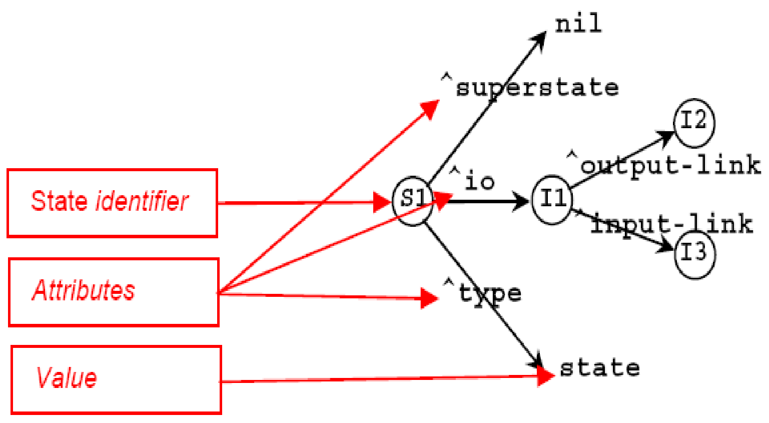


Figure 5.3 Working memory input – output link [(Laird, 2006a) with permission]

Working memory is modified by; productions, the decision procedure, and the working memory manager. Productions add augmentations in working memory, the decision procedure modifies the context stack, and the working memory manager removes irrelevant contexts and objects from working memory.

5.2.2 Reasoning cycle of *Soar*

Soar's basic reasoning cycle is shown in Figure 5.4 is as follows:

- Input
- State elaboration
- Proposing operators
- Comparing and evaluating operators
- Selecting the correct operator
- Applying operator
- Output

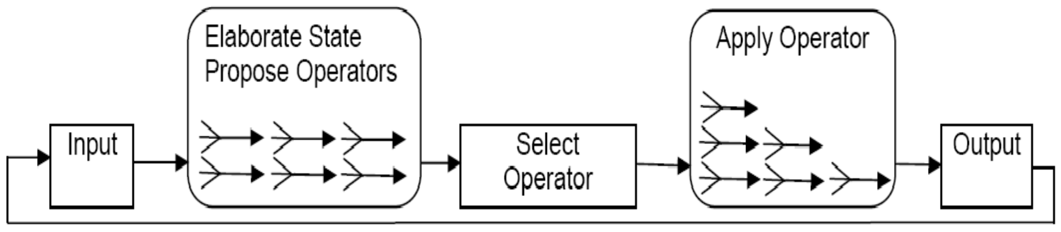


Figure 5.4 Reasoning Cycle of *Soar* [(Laird, 2006a) with permission]

In ***input*** phase, new sensory data comes into the working memory through the input-link. This new data is interpreted during ***elaboration*** phase which is next. The elaboration phase elaborates the state, proposes operators, and collects preferences. The working memory is examined by the productions in the long term memory and new objects, augmentation in old objects, and preferences are added. The productions that satisfy their conjunction of conditions with a consistent binding of variables by matching it with the contents of the working memory are successfully instantiated. A production can have a number of concurrently successful instantiations. The elaboration phase is monotonic. All successfully instantiated productions fire in parallel without any conflict resolution. The only type of conflict resolution in the elaboration phase is refractory inhibition which means an instantiation of a production is fired only once. Although in a serial machines, productions fire one after the other, this is only a limitation of the machine and is at a lower level and does not affect the simulated parallelism of *Soar* production firing. More importantly, the consequences of rule firing are accounted for, and 'simulated parallelism' does not affect the veracity of the system. The process of successful instantiation and firing of productions takes place in phases. When a production fires, the action part of a production modifies or adds *WME*s in the working memory that in turn satisfies the conditions of other productions. Eventually all productions that satisfy their conditions fire and there are no more productions to fire, at this stage the system is said to reach *quiescence*. Operators are proposed during elaboration phase and the preferences related to the proposed operators are also added in the working memory. After elaboration phase the ***decision procedure*** starts. The process of selection of an operator is based on the preferences for the operators. The preferences have three basic concepts: *acceptability, rejection,* and *desirability.* ***Acceptability*** is a choice to be considered. ***Rejection*** means a choice is not to be made. ***Desirability*** means a choice is *better than, worse than*, or *indifferent to* another choice. A choice can also be *best* or *worst*. A choice with best preference means that the choice is selected until either it is rejected or there is another choice better than it. A choice with worst preference is selected only when there are no other alternatives. The decision procedure interprets the semantics of the preference concepts to select an operator to be applied. After the operator is selected the rules that apply the operator fire which is followed by firing and retracting of state elaboration and operator proposal rules. These rules may also fire during the application phase. After reaching quiescence, output and then input are processed. Then elaboration phase described above starts again.

5.2.3 Conflicts in *Soar*

In *Soar*, the conflict resolution is not at the level of production rules rather it is at the level of problem solving. Because of independence and incompleteness of knowledge it is possible for the decision process to fail to select an operator to apply, in which case an impasse occurs that needs to be resolved to proceed further with the problem solving. In elaboration phase, individual productions expressing independent source of knowledge fire independently and contribute to the selection process. It is possible for an operator to be both better and worse than another, and thus create conflict of desirability between choices. The incompleteness of knowledge is due to the reason that the elaboration phase delivers some collection of preferences and these can be silent on any particular fact. *Soar* can at any time be in any state of incomplete knowledge.

Due to conflicting or insufficient knowledge impasse occurs. When multiple operators are proposed and there is not sufficient knowledge to distinguish them in order to select one out of them, then it is called an ***operator-tie-impasse***. When multiple operators are proposed but their preferences conflict then it is called ***operator-conflict-impasse***. A ***state-no-change-impasse*** occurs when there are no acceptable preferences to propose operators for the current state or all the acceptable values have been rejected. When a new operator is selected in the decision phase but no further productions fire in the application phase then an ***operator-no-change-impasse*** occurs.

5.2.4 Conflict resolution in *Soar*

Soar always creates a new state to resolve a conflict or impasse as called by the *Soar* designers. The goal of the new state is to resolve the impasse. As it is a new state created while solving a problem in the higher state and it is created to achieve a goal which is part of the main goal therefore it is interchangeably called *sub-goal* and *sub-state*. The new state is initialized with the information from the higher state and it carries a link to the higher state named as super-state. The value in this attribute points to the higher state. The new state also contains the complete description of why the impasse was created e.g., *tie* describes it was an *operator-tie-impasse* and with it the information about all the operators that tied is also given. In the new state operators are proposed and selected to further the problem solving but an impasse may occur in this *sub-state* creating another state therefore it is possible for *Soar* to have a stack of *sub-goals*. An impasse in *Soar* is not considered to be a problem rather problem solving in *sub-states* is a way of decomposing complex problems into smaller parts and *sub-states* provide a context to deliberate about which operator to select. The ***tie-impasse*** is resolved by productions that provide preferences for one choice to be distinguished from others or making all the choices indifferent. The ***conflict-impasse*** is resolved by the productions that create preferences to require one choice or eliminate the alternatives. ***State-no-change-impasse*** is resolved by productions that propose operators for the current state. And ***operator-no-change-impasse*** is resolved by productions that apply the selected operator, make changes in the state so that the proposal for the current operator no more matches, or new operators are proposed and preferred.

All states in *Soar* are active at all times and the processing goes on in all levels of states. An impasse is resolved when the knowledge becomes available in a state which created the impasse. When the impasse is resolved, *Soar* architecture removes the *sub-state* with all its *WME*s and preferences from the working memory as it has served its purpose and is no longer required. But the results that are created in the super-state are kept. The *sub-states* at all the lower levels are removed if an impasse at a higher level is resolved and the problem solving in a higher state progresses. The impasses may also become irrelevant when something in the outside world change causing productions to fire that create knowledge to resolve the impasse e.g., preferences to select an operator when the impasse is a tie.

The functioning of *Soar* starting from instantiation of productions in the *LTM* to the impasses and creation of *sub-states* or *sub-goals* is represented graphically in Figure 5.5. The process of new results writing new productions in the *LTM* is the learning process of *Soar* and is discussed in Section 5.2.6.

5.2.5 Truth maintenance system

Soar has a truth maintenance system which retracts results created by a rule from working memory when the concerned rule no longer matches. *Soar* has a support system for the facts in the working memory based on two types of supports; ***i-support*** and ***o-support***.

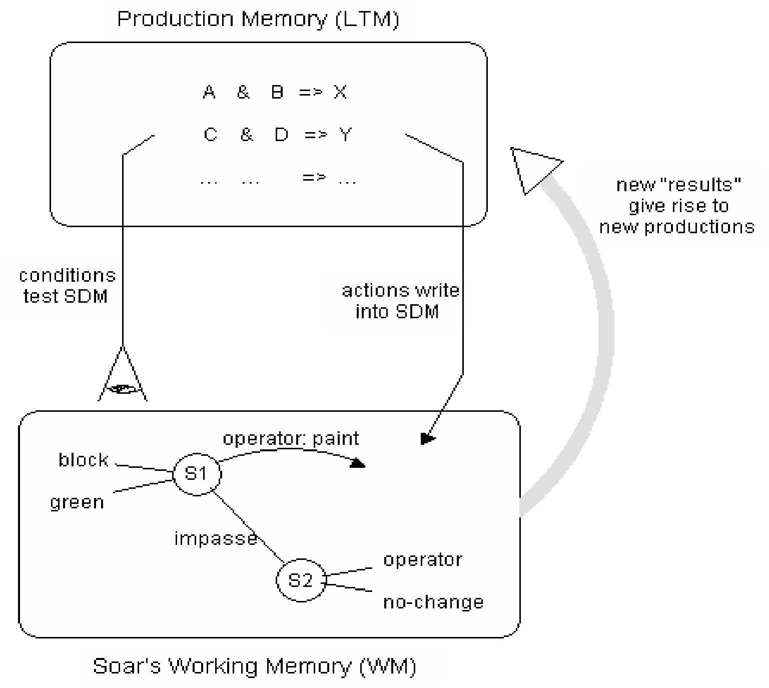


Figure 5.5 *Soar*: a functional diagram [(Ritter, 2007) with permission]

The *Soar* architecture classifies rules on the basis of their being part of operator application or not. If any antecedent of a rule tests the current operator and changes the state the result is classified to have *operator-support* or *o-support*. These *WME*s are persistent and may only be removed by other operator applications or if they get disconnected from the state. The results created from all other rules, to include rules that propose an operator, elaborate state, elaborate operators, or compare operators are said to have *instantiation-support* or *i-support*. The *WME*S that have *i-support* persist as long as the rule instantiation that created them matches. To explain the *i-* and *o-support*, the working memory of world with one blue and one yellow blocks on the table presented in Figure 5.2 is considered. There is a production rule that checks the *colour* attribute of the blocks and adds a *WME* *'^blue-block-present yes'* on the state object in the working memory if there is any block with the value *blue* of attribute *colour*. This fact in the working memory has *i-support*. This fact retracts if the condition of having a blue block is not satisfied any more. Now consider what happens if there is a selected *paint* operator that paints this blue block green and changes the value of colour attribute of the blue block to green. Now this change in the working memory has *o-support* and will remain there until explicitly removed. Automatic retraction of unsupported facts from working memory is a special feature of *Soar* and distinguishes it from other rule-based systems.

Determining the persistence of results from *sub-goals* is complicated because of the fact that the rules that created these results are removed from the working memory with the *sub-goals*. Thus the question arises how we can determine persistence of results when the rules that created the results have been removed. It is done by a rule created by *Soar* architecture called ***justification***. The condition part of the justification is the *WME*s that exist in the super-state and are tested by the productions that created the result. It is done by collecting all the *WME*s tested by the production rule that created the result and then removing the ones tested from the *sub-state*. The action part of the *justification* is the result of the *sub-goal*. The *justification* is tested as though it is the rule responsible for creation of the result kept in a state from the *sub-goal*. The conditions of the *justification* determine the persistency based on the fact that whether any condition tests an operator in this state or otherwise.

5.2.6 Learning

The learning mechanism of *Soar* is a form of explanation-based generalization. Automatic *sub-goaling* in all aspects of problem solving is the basis of learning in *Soar*. When a *sub-goal* is created and this new sub-space brings the required knowledge to solve the problem due to which the impasse is resolved then the *Soar* architecture creates a chunk production that later controls the search. And next time when this particular *sub-goal* needs to be created this chunk production fires and the problem solving proceeds without the impasse. As discussed earlier, the impasse that creates *sub-goals* is an architecturally detected lack of available knowledge and is that part of problem solving where *Soar* needs to learn. The *sub-goal* is created to find that knowledge if it is available in the form of productions in the *LTM* and the chunk is created to store this knowledge in the form of a *Soar* production in its *LTM* from where it can be straight away used when needed. Because of this additional availability of knowledge *Soar* improves its performance via a reduction in the amount of search. If all sub-spaces are exhausted that means all possible *sub-goals* in a problem space are created to make either a search control decision or perform a task implementation function then what is left is an efficient algorithm of the task. The efficiency of this algorithm depends on the quality of evaluation of the alternatives and the task-implementation methods used in the sub-spaces.

5.2.6.1 The mechanism of chunking

The chunk production is just like any other *Soar* production rule. The condition part of the chunk is the *WME*s in the state that allow through some chain of production firing to resolve the impasse. The action part is the result of the *sub-goal* which is the change made in the *sub-state* that terminated the impasse. The conditions of the chunk are based on a dependency analysis of traces of the productions that are fired in the *sub-state*. The traces keep a record of all the *WME*s that the production matched and *WME*s that it generated. The procedure of dependency analysis for chunking is explained in Figure 5.6. *WME*s are represented by circles both bold and otherwise. The *WME*s that form the condition part of the chunk are identified as nodes with bold circles before the impasse, i.e., before the first vertical line. The arrows going into nodes are rules that fire to add it. The arcs joining the arrows mean conjunction of the conditions at the tail of these arrows. The first vertical line indicates the start of the impasse and creation of a *sub-goal*, and the next vertical line indicates the resolution of impasse. Node R that resolved the impasse is created by the production rule that tested nodes 3 and 4 as its conditions. Node 4 is created after testing nodes 3 and B as its antecedents. Node 3 is created after testing node 1, while node 1 is created after testing the nodes D and A as its antecedents. The result node R depends on nodes 3, 4, and 1 and in turn they depend on nodes A, B, and D from the super-state. It is evident that node R can be created directly by testing *WME*s A, B, and D before the impasse occurs without creating any of the nodes 1, 2, 3, and 4, only if the dependence of result R on the *WME*s from the super-state is known. Thus if there is a production in the *LTM* that tests nodes A, B, and D in the current state and directly creates node R in the same state then there is no requirement of generating a *sub-goal*.

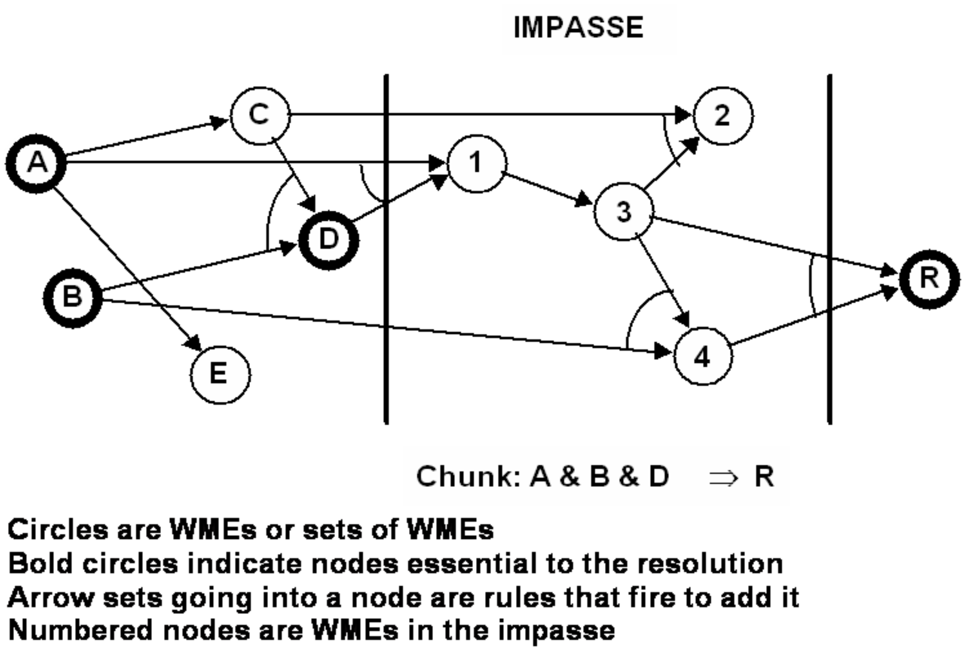

IMPASSE
A
B
C
D
E
1
2
3
4
R
Chunk: A & B & D ⇒ R
Circles are WMEs or sets of WMEs
Bold circles indicate nodes essential to the resolution
Arrow sets going into a node are rules that fire to add it
Numbered nodes are WMEs in the impasse

Figure 5.6 Chunking – the learning mechanism in *Soar* [(Ritter, 2007) with permission]

A generalization process is applied to the chunks to make them able to match a situation of similar description. This generalization process consists of changing the identifiers in the *WME*s by variables. The identifiers are used in *Soar* to tie together the augmentations of an object in the working memory – they carry no meanings and serves as a pointer to the object. A new identifier is generated every time an object is created. All instances of the same identifier are replaced by the same variable. Different identifiers are replaced by different variables and are forced to match different variables. To describe this generalization procedure the example of two blocks on a table shown in Figure 5.2 is considered again; if the conditions in the chunk are based on these *WME*s, *<s1> ^block <B1>, <B1> ^name A, <B1> ^colour blue, <B1> ^type block*, and the action is to give the best preference to the *paint* operator that changes the colour of the blue block to green. In order to generalize this chunk, the specific identifier *B1* is replaced with a variable that matches to any block with the attributes and values as shown in the conditions of the chunk above. All instantiations of identifier *B1* are replaced with the same variable in a chunk including the action side of the chunk. In this case it is the block that is being painted green. Chunks are further discussed with practical examples in Chapter 7.

Justification and chunk are similar in many ways both are in the form of productions with conditions and action parts, and the backtracing process of chunking and justifications is also the same. However, their similarities end here; the justifications are removed as soon as the *WME*s or the preferences that they support are removed, whereas, chunks are stored in the *LTM* with other productions. Chunks have variables in its conditions to match similar situations while justifications have identifiers; similar to an instantiated chunk.

Learning in *Soar* can be turned on or off. When learning is turned off the chunks are not produced.

5.3 Applications of *Soar*

The problem solving behaviour of *Soar* has been studied on a range of tasks and methods. *Soar* has been used to solve standard *AI* toy problems such as towers of Hanoi, missionaries and cannibals, eight-puzzle etc (Laird, 2006b). These tasks elicit knowledge lean, goal oriented behaviour. *Soar* has also been used to solve routine, algorithmic problems such as searching roots of a quadratic equation, doing elementary syllogisms, etc. *Soar* has also been run on knowledge intensive tasks which are the far end of the range of cognitive tasks and are used in current expert systems. *Soar* has been used to develop a system that performed the same task as that of an expert system named "R1" which used to configure VAX and PDP-11 computers at Digital Equipment Corporation. One quarter of the functionality of R1 was developed using *Soar* to show that it could completely replace the system if the effort warranted. *Soar* has been able to realize all the familiar weak methods (Laird and Newell, 1983). In larger and complex tasks, different weak methods solve different subparts of the task. *Soar* has also been used for creating intelligent forces for large and small scale military simulations (Hill et al., 1997), (Jones et al., 1999) and (Wray et al., 2005) such as synthetic theatre of war 1997 (*STOW-97*) in which *TacAir-Soar* flew all U.S. fixed wing aircrafts (Jones et al., 1999).

5.4 Improvements in *Soar*

Tambe (1997) developed a general model of team work and called it shell for teamwork (*STEAM*). The main model of *STEAM* is built on the *joint intentions* (Levesque et al., 1990) and the teamwork is modelled on the hierarchy of *joint* *intentions* based on the *shared plans* of Grosz and Kraus (1996). *STEAM* provides the ability to *Soar* to model team behaviour. *STEAM* has about 50 domain independent production rules to facilitate the modelling of team behaviour. Sun et al. (2004) developed a model on the lines of *STEAM* called *Team-Soar* and compared it with collaborative agents for simulating teamwork (*CAST*) model (Yen et al., 2001). Both of the teams are given the same task and similar procedural and declarative domain knowledge. *Team-Soar* contains 22 production rules encoded as communication knowledge, whereas, *CAST* has an elaborate communication mechanism embedded in the architecture. Sun et al. (2004) found out that: some of the behaviours of both teams is similar; although, *CAST* has an efficient communication mechanism, as it is embedded in the architecture, compared to *Team-Soar* but it communicated quite frequently compared to the team members of *Team-Soar*. In *Soar*, implementing teamwork models such as *STEAM* or *Team-Soar* requires writing *Soar* rules to incorporate collaboration and communication.

To introduce variability in the behaviour of *Soar* agents as a requirement of *HBR*, Wray and Laird (2003) modified the *Soar*'s knowledge representation and modified the decision making process to support the change in knowledge representation. The decision making process now also takes into account numerical values associated with operators in the absence of symbolic preference. Symbolic preference has priority over numerical value. As is true for all knowledge in *Soar*, the rules giving numeric values for candidate operators are context sensitive. Thus, there may be any number of rules that give numeric values for an operator. There exist many potential choices to use these multiple numeric value for selection of an option. One choice may be averaging the values and then a random choice made from the normalized probability distribution of the averaged values. The second choice may be to sum them up and then randomly select one from the normalized probability distribution of the summed values. The selected method is to sum up all proposed numeric preferences for an operator O_i into a total score $Sum(O_i)$. The winning operator is selected probabilistically according to the ***Boltzmann*** distribution as per Equation 5.1.

$$P(O_i) = \frac{e^{Sum(O_i)/Temperature}}{\sum_i e^{Sum(O_i)/Temperature}} \qquad \text{Equation 5.1}$$

The parameter *Temperature* is used to round the peak of the probability distribution. Nason and Laird (2005) used the acquired capability in *Soar*, of selecting options based on numerical preferences, to extend the architecture to add reinforcement learning. The reinforcement learning is a type of learning in which the task is to learn how to act in a given environment so as to maximize a reward signal. This is a credit-assignment problem of determining what was responsible for the reward or punishment. In most reinforcement learning approaches, the agent learns a value function, which is an estimation of expected sum of future rewards for taking an action in a particular state. In *Soar-RL*, the numeric preferences represent a state-operator value function. And the reinforcement learning task is to adjust the numeric values as the agent encounters rewards in the world. *Soar* is extended to receive the rewards as one of the inputs from the external environment. The environment rewards the successful operators with a positive value corresponding to the level of success and a negative value to represent punishment. The learning therefore, is as good as the measure of success for reward signal. The reinforcement learning in *Soar* (*Soar-RL*) updates the numeric preference in the next cycle and stores only the immediate history. The updating procedure of *Soar-RL* is similar to the procedure used in state-action-reward-state-action (*SARSA*) algorithm (Rummery and Niranjan, 1994). The state-operator value function is distributed over a number of rules generating numeric preferences for an operator for a particular set of features in the working memory, and the numeric preferences are summed up to form the expected value of reward signal used for the selection of an operator. Thus, the update in the value function due to the recent reward signal is also distributed equally over all such rules.

Nuxoll and Laird (2007) integrated episodic memory with *Soar* in order to extend case-based reasoning (*CBR*) paradigm. *Soar* architecture is extended to incorporate a working memory activation system (Nuxoll et al., 2004) on the lines of the activation scheme in *ACT-R* (Anderson and Lebiere, 1998). Previous episodes are stored in the episodic memory and are utilized to remember locations of required items during search, and also to learn other actions e.g., dodging enemy fire. This work is an improvement on an earlier work of the authors (Nuxoll and Laird, 2004) in which the learning through episodic memory is tested on relatively simpler tasks.

5.5 Summary

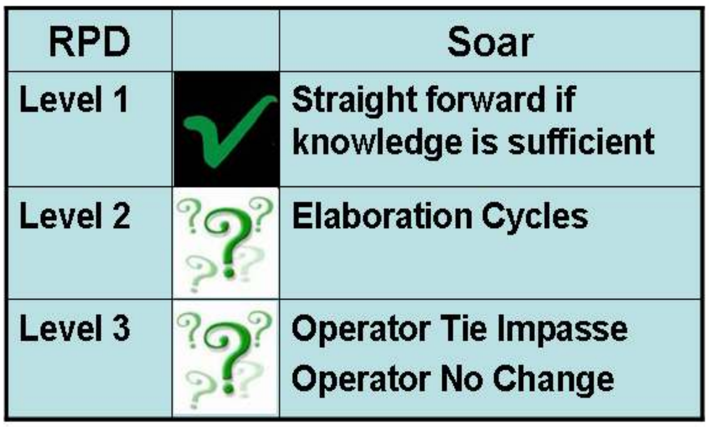
Soar is a cognitive architecture which has long term and short term memories, an elaborate truth maintenance system, an architecturally supported goal stack, automatic creation of sub-goals and *sub-states* due to impasses, and a learning mechanism that produces new production rules that can be straight away utilized. *Soar* has been used to produce intelligent forces for large scale military simulations and *wargames*, and the architecture is continuously improved to match future requirements. Soar provides a convenient framework to model all aspects of *RPD* model which facilitates the implementation of *RPD* in *Soar*.

6 THE RPD-SOAR AGENT

In this chapter, the implementation of the *RPD* model in the *Soar* cognitive architecture is discussed. The similarities between *Soar* and *RPD* that assist in implementation of the model are highlighted first. Then different components of the architecture are briefly discussed and then the interface built on *Soar mark-up language* (*SML*) is discussed. The different processes involved in the working of the *RPD-Soar* agent are discussed with the help of a vignette of an advance to contact military land operation. The behaviour of the agent is directed by its experiences or previously encountered situations that are stored along with their by-products of goals, cues, expectations, and courses of action, in the *LTM*. These experiences are required to be translated into *Soar* production rules for a *Soar* agent to understand them and behave accordingly. These experiences along with related *Soar* production rules are discussed. In the end of the chapter, the integration of a neural network in the over all architecture is discussed. Generally in the thesis and particularly in this chapter, the words situation and experience have been used interchangeably when situations are mentioned as memory contents that are being recognised. Because it's these situations that the agent has faced in the past are remembered now as his experiences.

6.1 Similarities between *Soar* and *RPD*

Soar has many similarities with the *RPD* model that may be used to our advantage in developing the *RPD* model. The first advantage of using *Soar* to model an *RPD* agent is that recognizing a pattern at the input and proposing relevant operators according to the situation is already a part of the architecture. The second advantage is that the state elaboration phase may be used to process information and reason with it to recognize the situation for Level 2 *RPD*. The third advantage is that if sufficient knowledge in the *LTM* exists then *Soar* behaves like Level 1 *RPD* model. And the fourth advantage is that the basic structure in *Soar* is problem space based, and with the help of impasses sub-spaces can be created for mental simulation (Raza and Sastry, 2007). These similarities are tabulated in Figure 6.1.



RPD		Soar
Level 1	✓	Straight forward if knowledge is sufficient
Level 2	?	Elaboration Cycles
Level 3	?	Operator Tie Impasse Operator No Change

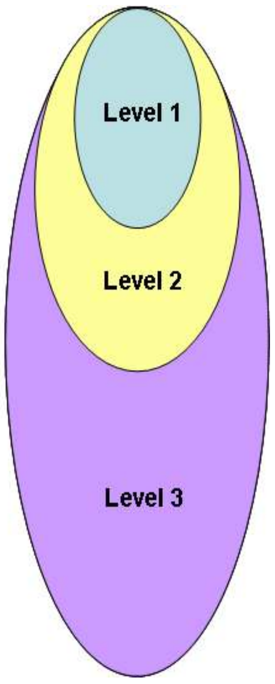


Figure 6.1 Similarities between *Soar* and *RPD*

6.2 The architecture

The external environment or the world is developed using the Java programming language and the agent is developed using the *Soar* Cognitive architecture. The *Soar* agent and the external environment are interfaced using *Soar* mark-up language (*SML*). Different environments based on maps for different scenarios can be loaded into the system. Agents with different behaviours may be loaded into the system as production rules in *Soar* files (Raza and Sastry, 2007). The architecture of the agent is shown in Figure 6.2. In the *RPD* model it is the experience of the agent that guides its behaviour. As recognition primed decision making is modelled within the *Soar* cognitive architecture, therefore, experiences of the *RPD* model consisting of goals, courses of action, cues, and expectations are transformed into appropriate *Soar* production rules. And these *Soar*-production rules are stored in the agent's *LTM*.

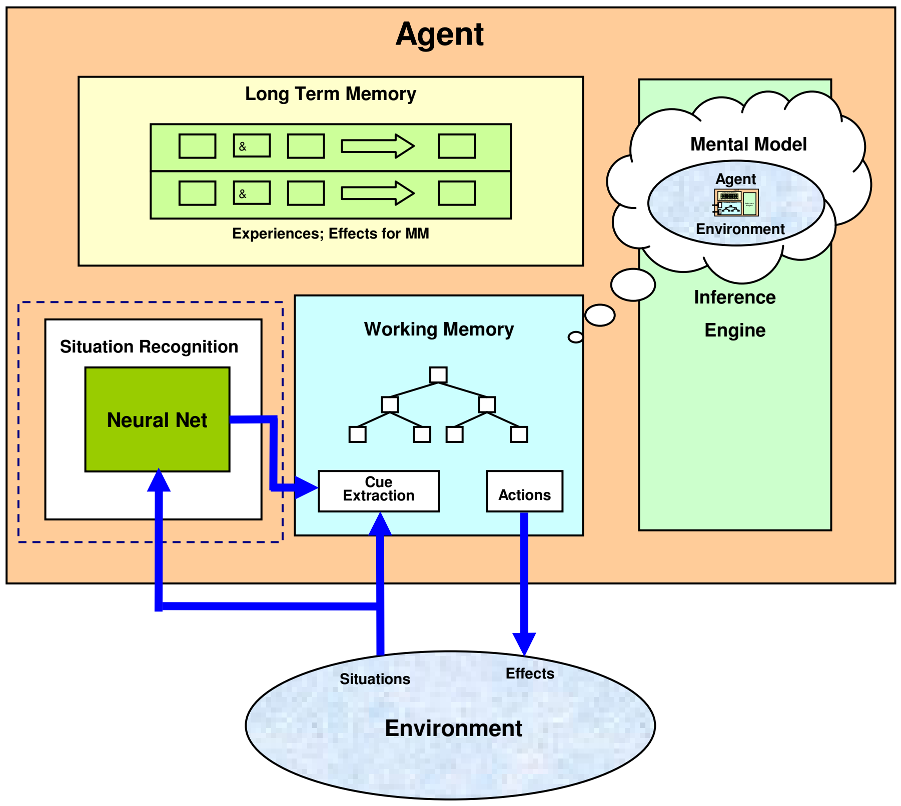


Figure 6.2 Agent architecture

At the start of each simulation step the situation present in the environment is given as input to the agent at its input-link. The agent examines the elements of the situation present at the input-link and the information available in its own working memory, and if sufficient knowledge is available, it recognizes straight away that a situation is typical. This is Level 1 *RPD*. Some situations are complex and the decision maker has to devote more attention to diagnosing the situation. In some of these situations, the information from the environment is required to be processed and combined with other available knowledge in order to recognize a situation as typical. This is a chain reaction and therefore, based on these processed cues and information available in the working memory more production rules stored in the *LTM* fire to process other associated information in order to understand the situation better. This part of Level 2 *RPD* is implemented with the help of the elaboration phase in the *Soar* agent's decision cycle. Level 2 *RPD* for very complex situations warrants story building to account for some of the inconsistencies. The story building part of Level 2 *RPD* is not implemented in this model.

The situation or experience once recognized is transferred to the working memory when the rules containing this experience are fired. The experience appears in the working memory in the form of *WME*s proposing courses of action applicable to this situation, setting goal(s) to accomplish, indicating expectation(s) and indicating important cues to monitor. All these elements are present in the *LTM* as part of the experience of the agent in the form of *Soar* production rules. Based on the available knowledge, the inference engine either takes a decision to select one course of action to implement or forms a mental model to mentally simulate one or more courses of action in order to select one to implement. The environment may be modified by the action of the agent or actions by other entities in the environment. When an agent takes a decision that needs to change the external world the information is put on the output-link of the agent in the output phase of the *Soar* decision cycle. The external world is waiting for any information on the output-link and changes itself accordingly as and when any information becomes available. As soon as the world changes, it provides this information at the input-link of the agent which picks up the information on each of its input phases.

A trained neural network is used to help the agent in recognizing the situation. As the broken outline around the neural network suggests, the neural net is not used in all cases. The reason for integration of an artificial neural network in the architecture and its functioning as the integrated part of the architecture is discussed in Section 6.8.

The implementation is aimed at producing an agent mimicking the decision-making behaviour of humans. Therefore, the model of the physical world and entities in it have been restricted to represent actions and effects of decisions taken and do not include the representation to implement motor actions and its effects at a higher level of resolution. For example, the reasoning and action selection is restricted in a situation to the point where a tank commander selects the action ‘turn’. The action ‘turn’ has not been further decomposed into the motor actions of braking, turning the steering, etc.

6.3 Mental simulation

In the computer implementation of mental simulation, as in the case of humans, a model of the external environment is created in the agent’s head. For mental models and related errors see (Burns, 2000). All the objects present in the environment are modelled and the effects each action creates on these objects are also modelled in the same way. The mental model of an agent has only the information that is available to the agent at the time the mental picture is created. For example, the agent can see only one cell around itself and while mentally simulating an action it moves two cells ahead but it will know only that much information about this new area that it has when it starts the mental simulation. While creating the mental model and replicating the world the restrictions mentioned above are kept in mind. Moreover, the environment is modelled in such a way that there is no link between the objects in the mental world and the same objects in the real world. This need to be carefully done in *Soar* and is taken care of in *Selection space* production rules developed by *Soar* group (Laird, 2006a). Due to this isolation, during the mental simulation when the agent selects an action and applies it, only the mental world changes and the outside environment remains un-changed. As the world is modelled in the agent's head and actions are also implemented in this model, therefore, the effect of each action on the mental world is required to be the same as that of the real world, which also needs to be modelled. This is done with the help of separate production rules taking the same action as that of the real world but applicable only to the mental world. Their applicability only to the mental world is ensured by keeping appropriate antecedents to check the absence of input- output-link on the states representing the world. The agent then looks at this changed mental world and then merits the action numerically after considering the progress made in achieving the goal. Past experience in the form of production rules help the agent in preferring an operator to be evaluated first and in judging the usefulness of this action in achieving the goal. In this experiment only one step mental simulation has been implemented. In *Soar*, the mental model for simulation is developed by creating problem sub-spaces using operator-tie and operator-no change impasses (Raza and Sastry, 2008). After an applicable course of action is evaluated then the agent dissolves its mental model either to go ahead and apply the selected operator to the real world or to make another mental model to evaluate the next candidate course of action (Raza and Sastry, 2007).

A similar mental model using *Soar* is developed by Johnson (1994a) and (1994b) for a different reason and that is to explain the actions taken by intelligent tactical air agents in *TacAir-Soar*, the *Soar-IFOR* project discussed in Chapter 2.

This mental simulation is a part of *Level 3 RPD* model and helps the agent select a course of action to implement. Unlike *Level 1 RPD* where the agent is very sure as to which course of action should be implemented in a given situation, *Level 3 RPD* model is useful for situations where the agent lacks sufficient experience to exactly know how a course of action will play out, and therefore, the agent mentally simulates courses of action to see how it unfolds. Based on its experience the agent knows which course of action should be mentally simulated first. The agent simulates the preferred course of action and if it satisfices then the agent implements it (see Section 7.1.4). If this course of action is not suitable then the agent selects the next course of action in line for mental simulation. The prioritization of these courses of action is based on the experience of the agent where it remembers as to how successful a course of action was when implemented in this situation previously. When an agent faces a completely new situation, it suffers from lack of experience and degenerates to traditional decision making. Due to its capability of mental simulation it evaluates all courses of action serially, selects the most suitable course of action and implements it and due to its adaptability remembers it for the next time as an experience. Although in this implementation mental simulation is run for a single step, it can be run for as many steps as required. If more than one courses of action are equally promising then the agent selects one at random. This method is further discussed in Chapter 7, where evaluation based on mental simulation generates variability in the behaviour of the agent. Variability in behaviour within an agent across episodes for the same situation and task is a major requirement in human behaviour representation, and mental simulation in *RPD-Soar* introduces this variability in the behaviour of the agent. This ability of the *RPD-Soar* agent is an advantage it has over other implementations.

Before discussing the working of the model in detail with the help of a vignette, the interface of the simulation environment with the *Soar* kernel is discussed.

6.4 The interface

The simulation environment is interfaced to the *Soar* kernel with the help of *Soar* mark-up language (*SML*), as shown in Figure 6.3.

The simulation environment consists of objects or ‘entities’ as usually called in simulations and some of these entities are *Soar* agents. The *Soar* kernel is capable of developing and maintaining multiple agents and each can have its individual behaviour based on the *Soar* production rules loaded in that agent. *SML* was developed by the *Soar* group (Threepenny, 2005) to provide an interface into *Soar*. The client can send and receive *Soar* XML packets through a socket maintained by *Soar*, which is port 12121 by default. Client*SML* is available in C++, Java, and Tcl. We have developed the simulation environment in Java and for a client implemented in Java, Java_sml_ClientInterface.dll, *Soar*Kernel*SML*.dll, and ElementXML.dll dynamically loaded libraries are required.

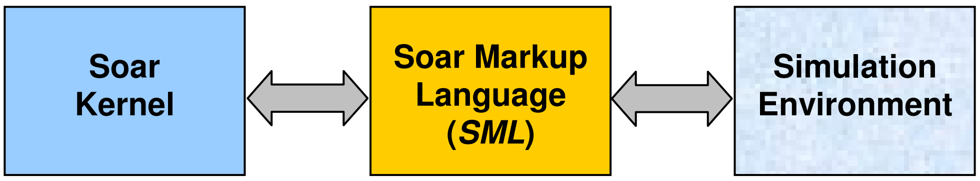


Figure 6.3 The interface

The entities present in the environment are represented as objects in the working memory of the agent therefore; it is natural, logical and more compatible to model the world using object oriented software. Java is an object oriented language and therefore it can be used to build synthetic environment (Sommerville, 2004). The other option is to develop the environment in C++ which is also an object oriented language and can be interfaced with *Soar* using *SML*. But in this implementation Java is preferred over C++ in order to remain in line with the *Soar* group.
The complete code for implementation is available in the attached CD (see Appendix C) and the key elements of the *SML* code are given in Appendix D.

6.4.1 Creating *Soar* kernel and agents

A *Soar* kernel can either be created in a new thread or in the same thread. In this implementation, the *Soar* Kernel is created in a new thread so that the simulation is run independent of the *Soar* Kernel. Multiple agents can be created in one kernel with different behaviours. The behaviour of each agent is controlled by the production rules loaded in it.

6.4.2 Input - perception

The "input-link" of the *Soar* agent, as explained in Chapter 5, is the link of the agent to receive the information about the outside world. This information is picked up by the agent during the input phase of the next decision cycle. The client needs to acquire the identifier of the input-link in order to give all the information depicting the present situation of the world to the agent.

The identifier *WME*s are used to create objects at the input-link. String and integer *WME*s are created either directly on the input-link or as part of the object represented by an identifier at the input-link. It has been discussed in Chapter 5 that a *WME* is an identifier, attribute, and value triplet. The value is either a constant or an identifier. The value is an identifier if it is not a terminal node and one or more branches are emanating out of this node. The '*bluetank*' is created as an object in the working memory at the input-link representing an entity present in the simulation environment (Figure 6.4).

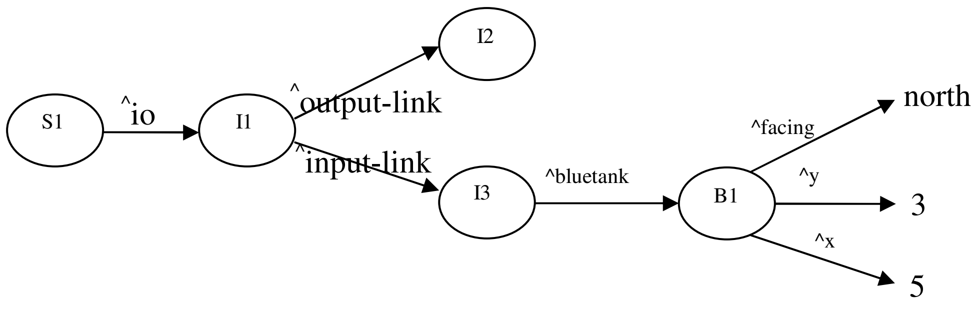


Figure 6.4 Objects on the input-link

The object '*bluetank*' has three attributes; two of them give its location in the Cartesian coordinates and third indicates the direction that the '*bluetank*' is facing. The *X* and *Y* coordinates are represented with the *WME*s of type integer and the direction that the tank is facing is represented with a *WME* of type 'string'. All the information about the environment and entities present in it that are required by the agent to reason for situational awareness to make decisions is provided to the working memory of the agent through the input-link.

The environment in this model is grid based. Each cell in the grid is surrounded by its neighbouring cells. Each cell has at least three and at most eight cells as its neighbours. These cells are represented as objects in the working memory of the agent because each cell has two attributes representing its location in Cartesian coordinates. These attributes have integer values and can be represented with the help of techniques discussed above. But consider an example of *Cell 5* (Figure 6.5) that has a neighbouring cell *Cell 2* which is just above *Cell 5*. To represent the relative position of these cells in this environment, a *WME* need to be created, this has the identifier of *Cell 5* as its identifier with an attribute north and the value of this attribute being the identifier of the cell in the north the *Cell 2*. This is a case where graphical representation is required instead of a simple tree structure. In order to develop a graph in working memory of the agent new identifier *WME* with the same value as that of an identifier of an existing object need to be created called 'S*hared identifier WME*'. An agent created in the *Soar* Kernel can create this type of WME using an inherent method for the purpose.

Cell 1	Cell 2	Cell 3
Cell 4	Cell 5	Cell 6
Cell 7	Cell 8	Cell 9

Figure 6.5 Example of graph structure in WM developed from shared identifier *WME*

Cell is developed as a class in Java. The agent instantiates the cell object to create the nine cell graph structure. The agent sits in the centre in Cell 5 in Figure 6.5. This template of nine cells moves over the map and the values of the x and y attributes representing Cartesian coordinates of the location of the cell on the map and the value of the content giving the name of the object present on the location where the cell is located now keep changing accordingly.

6.4.3 Output – command/action

If the agent produces a command then it is put at the output-link after the output phase. One or more commands present at the output-link are picked up for implementation in the environment.

After acquiring all the information related to the command from the output-link the agent is informed that the commands have been picked up. This information is used by the agent to remove the implemented commands from the output link.

6.4.4 Event handling

In this model event handling is required to update the user interface in the environment and to connect the environment to the 'Java debugger'. The Java debugger can connect to the remote *Soar* kernel given an internet protocol (*IP*) address and a port number. The *IP* address is not required if the *Soar* kernel is running on the same machine. When the environment is updated, the world represented in the user interface along with the buttons in the bottom of *GUI* are also updated. Stop, start, and update events are registered with the environment and they trigger actions wherever required.

6.5 Graphical user interface (*GUI*)

The interface has four buttons Run, Stop, Step, and Reset to control the simulation (Figure 6.6). The Run button when pressed runs the agents forever until either the Stop button is pressed or the agent achieves its goal. All the buttons are enabled and disabled appropriately. The *GUI* is updated whenever the agent makes a decision to take an action in the world. The simulation and the *GUI* are running in separate threads and therefore the *GUI* is updated independently of the simulation.

6.6 The Environment

The environment is grid based (Figure 6.6). The perimeter has obstacles and the agent's world is restricted to these boundaries. There is a *Map* class which contains the location of obstacle and initial location of the red tank, and is responsible to place the appropriate map for the task. The agent is a tank commander who is commanding a single tank. There are two types of sensors in the tank, one is a visual sensor that looks only one adjacent cell around itself, and the other is a radar sensor that can see up to five cells in the direction that the tank is facing. The radar sensor can not see beyond any obstacle. Past observations from the radar are retained in the memory of the agent and it can use this information in decision making. This environment is more or less common in all the experiments but the changes, if any, are mentioned in the experiments.

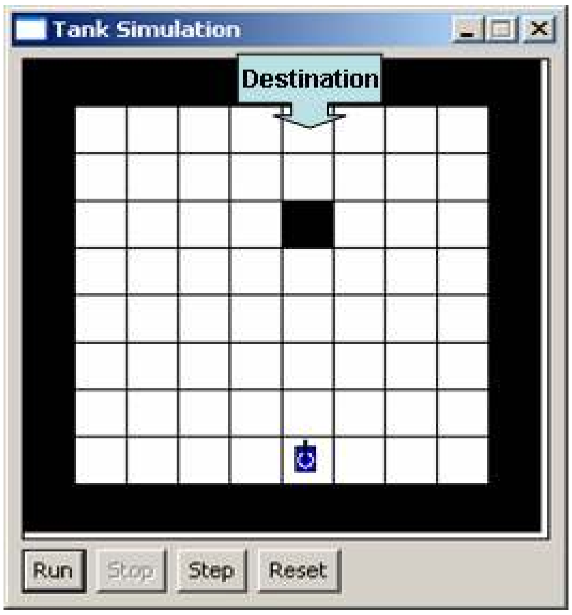


Figure 6.6 The Environment

6.7 Working of *RPD-Soar* agent

The implementation and working of the *RPD-Soar* agent is explained with the help of a vignette. The context is an advance-to-contact military land operation. In a 10 x 10, grid based environment (Figure 6.6), the tank has to start from the south and advance towards north to reach the destination. The environment has only one obstacle which is a hill that gives protection from observation and fire. The agent has radar and visual sensors as described in Section 6.6. The agent has been given the location of the destination cell and has been tasked to advance to that location. Enemy tanks are expected on the route to delay the advance. The firing range of an enemy tank is three kilometres while, that of the agent is four kilometres. In this experiment one cell represents one kilometre. In this thesis the scales for representation of terrain, if required, are mentioned with the experiment.

Most tasks are performed within a larger context that includes higher-level goals. In this case the main context is an advance-to-contact military land operation. There are three high level contexts in this experiment and each is represented with an experience. The experience has goals, cues, expectations, and a course of action. These high level contexts are mutually exclusive and the agent at one time is in any one of them. These experiences are shown in Figure 6.7, Figure 6.8, and Figure 6.9.

Experience: Advance

- **Goal**
  - **Reach the destination**
- **Cues**
  - **High ground: not visible**
  - **Incoming missile: none**
  - **Enemy tank: none visible**
  - **Distance to the destination**
- **Expectations**
  - **No incoming missile**
  - **No enemy tank visible**
  - **No high ground within four kilometres**
- **Course of Action**
  - **Move towards destination**

Figure 6.7 Experience – advance

The *goal* is the state of affairs that is intended to be achieved and may also be defined as the end state to which all efforts are directed. The *cue* is the perception of a set of patterns that gives the dynamics of the situation, and makes distinctions in these patterns. This pattern is formed by the features of a situation or elements in an environment. The *expectation* is the belief of the agent that an event will or will not occur in a given situation. The *course of action* is the strategy or plan that the agent intends to implement.

Recognition of a situation not only means recognizing a typical response but also indicating what goals make sense, what cues are important and what is expected next. During advance an important cue is *high ground*. The agent expects to see no high ground within four kilometres of it. Now if the agent finds high ground within four kilometres then this expectation is violated and a fresh evaluation of the situation is necessary. If the agent finds high ground within four kilometres of itself and is facing north, which is the direction of its destination, then it recognizes this situation and changes its state to *manoeuvre*. During *manoeuvre* the agent does not expect to see an enemy tank. If it sees a tank an expectation is violated and the situation is evaluated again.

Experience: Manoeuvre

- **Goals**
  - **Expose the enemy tank at the longest range**
  - **Do not expose own tank to enemy within enemy tank's firing range**
- **Cues**
  - **High ground: at a distance <= 4 kilometres**
  - **Direction of own tank: facing destination (north)**
  - **Incoming missile: none**
  - **Enemy tank: none visible**
- **Expectations**
  - **No incoming missile**
  - **No enemy tank visible**
  - **Enemy tank behind high ground on completion of manoeuvre**
- **Course of Action**
  - **While taking cover from the high ground, move to a location four kilometres east of expected enemy tank**

Figure 6.8 Experience – manoeuvre

Experience: Attack

- **Goal**
  - **Destroy the enemy**
- **Cues**
  - **Enemy tank: visible**
- **Expectations**
  - **Enemy tank remains visible**
- **Course of Action**
  - **Engage the enemy tank with fire**

Figure 6.9 Experience - attack

A brief description of the agent's behaviour will be given here, a fuller explanation together with the code generated is given in Appendix D.

If we set up the simulation with the map representing the environment displayed in Figure 6.6, load the agent with the behaviour required to accomplish the mission for *advance-to-contact* operation, connect it with *Soar* Java debugger and then run it for a single step then the agent will start to develop working memory contents. Running the simulation one step also makes the agent run through one decision cycle. The information generated by the radar and the visual sensors is put in the working memory through the input-link of the agent. The agent is facing north and is five cells south of the high ground therefore the radar sensor of the agent sees an obstacle at location represented in Cartesian coordinates as (5, 3). The visual sensor as we know can see only one cell around itself and therefore, sees three obstacles in the south, south-west, and south-east of the agent. The rest of the five cells around the agent are empty and are displaying their contents as empty in the working memory.

The objects in the environment such as *blue tank, map, cell, radar* and *obstacle* are represented in the working memory of the agent. The information about these objects in the environment is given to the working memory through the input-link of the agent. *Operator* and *direction* objects are produced in the working memory by the production rules loaded in the long term memory (*LTM*) of the agent. The *state* object is automatically created in the working memory of the agent. Two production rules, designed for the purpose, fire to initialize *RPD-Soar* agent and place the mission of the *advance-to-contact* operation as the desired state in the working memory of the agent.

The simulation is run through the next step and conditions based on the cues of experience for *advance* (Figure 6.7) as the suitable course of action is selected. There is no red tank in sight, the obstacle is five kilometres away, and there is no incoming missile. The presence of red tank and incoming missile are straight forward cues but in order to observe the cue of relative distance of tank to the obstacle some elaborations is required which is Level 2 *RPD* and is done with the help of production rules designed for the purpose. The *advance* course of action is an abstract operator. Therefore an operator no-change impasse occurs and a new *sub-state* is created to implement it.

In this context with *advance* as its major task the agent has four actions to choose from: move in the direction that the tank is facing and turn in the other three directions. And as all four are applicable in the situation then an operator tie impasse is generated. This is the case in *RPD* model where the decision maker can not select a course of action from a pool of courses of action that he knows can apply. Now the decision maker develops a mental model of the environment and mentally simulates the courses of actions serially to select the one which seems satisficing.

Now among the candidate operators in experiments discussed later, the agent has the experience to prefer one operator over the other for evaluation and the experience to judge when an operator is satisficing but this agent evaluates each and every candidate serially and randomly selects one to evaluate first. Therefore, one operator is selected for evaluation at random.

This operator named *evaluate-operator* is also abstract and therefore another space is created to implement evaluation and this is the mental model for simulating a course of action as of *RPD* model. In this space, all the objects in the environment are modelled again and the operator representing the course of action to be evaluated is selected to be applied.

The operator application is not on the real world rather on the model world created in the agent's head. In this case the course of action is being evaluated for advance which means a better action is the one that can take the agent close to the destination given in the original mission. In order to evaluate the candidate actions, the Manhattan distance is calculated after applying each action and the numeric value is recorded as evaluation factor. The Manhattan distance between two points (x1, y1) and (x2, y2) is defined in terms of X and Y as X = x2 - x1, and Y = y2 - y1. And then the action with the least numeric value is selected. This is achieved through the use of *selection space* implementation provided by *Soar* group (Laird, 2006a) and the production rules written for copying the objects and the application of operators in the mental model for this implementation. The majority of the production rules provided as *selection space* productions are being used as such in this implementation for mental simulation while some of them are modified to suite the requirements of this model.

After evaluating each action the *sub-states* of the mental model and thus all the *WME*s related to them are removed from the working memory of the agent and only the evaluated value is kept in the higher state evaluating these actions.

After evaluating all the candidate actions the *move north* operator is selected because it is taking the agent close to the destination and is applied to the real world. It is done through the output-link and with the help of the model for acquisition of commands from the agent explained earlier in the same chapter. The new location of the *Blue* agent in the environment after moving north is shown in Figure 6.10.

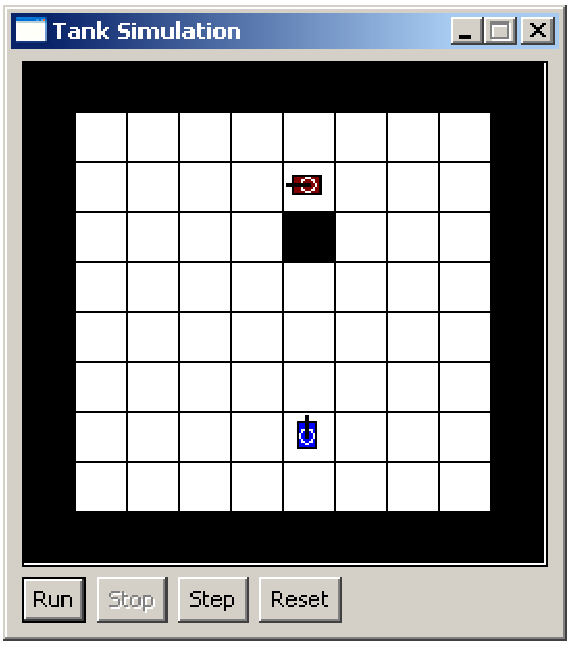


Figure 6.10 Situation after moving north

Now the distance to the high ground is equal to four kilometres and one of the expectations of the advance experience is not met, therefore the situation is re-evaluated and this time the experience manoeuvre is recognised as its conditions are met. The course of action for the experience manoeuvre is represented graphically in Figure 6.11. In this case the blue agent sees high ground on its approach to its destination and expects an enemy tank behind it. A similar approach has been adopted by Tambe and Rosenbloom (1995) where the pilot agent observes the actions of the enemy aircrafts and by observing the observable actions infers their unobserved actions, plans, goals, and behaviours.

The course of action manoeuvre is also at higher level of abstraction and creates an operator no-change impasse.

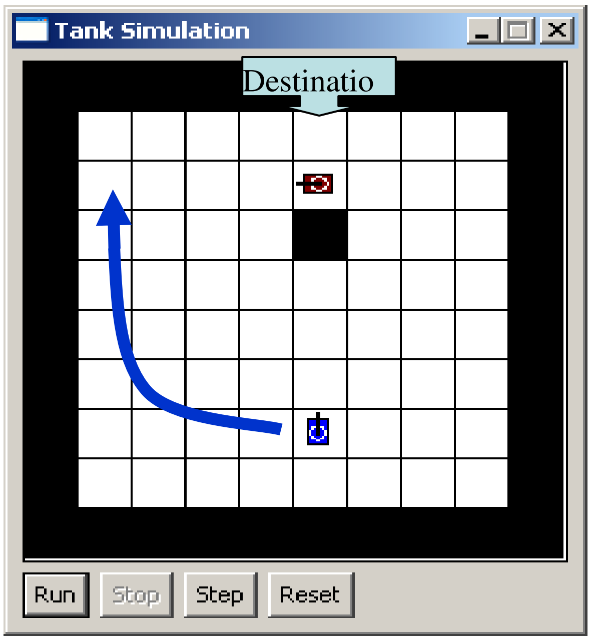


Figure 6.11 Experience – manoeuvre

Just like advance, this course of action for experience manoeuvre is implemented through atomic actions of move and turn but now the destination is the location pointed by the head of the arrow representing the planned path for movement of blue tank.

This location as the destination for completing the manoeuvre action is kept so that the Blue tank stops at a distance of four kilometres from the Red tank and therefore is out of the firing range of the enemy while the Red tank is within the firing range of Blue tank. The Blue tank commander is exploiting the weakness of the enemy to achieve his own aim of destroying the enemy forces as secondary mission while reaching the destination which is the main mission. In this situation it would have not been possible for the Blue tank to reach its destination without destroying the Red tank or making it retreat from its present location as the area would have been unsafe to advance.

The selection of the atomic actions in experience manoeuvre is through mental simulation as is the case of experience *advance*. It is not necessary for all the experiences to have all the components of situations as represented in the *RPD* model. It is understandable that the recognition of a situation requires more processing of information for comparatively high level contexts; therefore, it is expensive in time and resources to repeat the process with every single change in the world. It is also true that not all changes in the world are likely to change the higher context. It is also observed that the behaviours at a higher level persist for a comparatively longer time and consist of a combination of low level behaviours. There may not be a requirement to associate expectations with the courses of action in the experiences at atomic level behaviours where an action is taken that changes the world and then the situation is re-evaluated to select the next action. This is because the selected course of action does not persist long enough to require watching expectations while the action is under progress. The same is true for the goal at atomic level. The goal is the result of the action itself. Therefore, in this implementation of the *RPD* model, the goals and expectations are part of the experiences representing behaviour at a higher level of abstraction. At atomic level the experiences consist of only cues and the action. The success value or preference of one action over the other accompanies the experiences at even atomic level in most cases. This success value is used in two ways: the first, is the selection of a course of action straight away without mentally simulating it if one candidate is distinctly better than the others; and the second, is the selection of a course of action as the first one to consider for mental simulation when the chances of success of candidate courses of action are similar.

In *Soar*, it is effortless to model the phenomenon of watching the expectations while carrying out a course of action. In *Soar*, all the states are active at all times. Any change in a state at a higher level removes all the *sub-states* which are responsible for the creation of these *sub-states*. In the vignette under discussion (see Figure 6.6), the advance behaviour is selected and the course of action is under progress when the blue tank moves north and the distance between the blue tank and the obstacle reduces to four kilometres (Figure 6.10). The agent is expecting no obstacle this close while advancing thus an expectation is violated and the situation needs to be re-evaluated. In *Soar*, the re-evaluation of a situation given the violation of expectations is almost automatic if the conditions for selection of the concerned operators are set correctly. The abstract advance operator that creates the *sub-state* where this course of action is being implemented is removed due to one of its conditions for selection being violated and thus the *sub-states* implementing it are also removed. The situation therefore is re-evaluated to recognize new situations in order to find courses of action from other experiences to proceed with the task.

During the manoeuvre context the blue tank keeps moving by selecting actions that reduce its distance from the destination recognized as a goal with the present situation until it reaches the destination. To accomplish its goal completely the blue tank also turns east as shown in Figure 6.12. Now the blue agent finds the red tank on its radar sensor. The only cue in the *attack experience* is red tank (Figure 6.9) and for its selection the condition to be satisfied is red tank's presence. As the condition is met therefore the proposal to select attack as a context is fired by a production rule and as attack is the only operator proposed therefore it is selected. Attack is an action at a higher level of abstraction therefore a new *sub-state* is created through an operator no-change impasse to implement this abstract action. In this context a fire action is proposed, selected and applied and the red tank is destroyed.

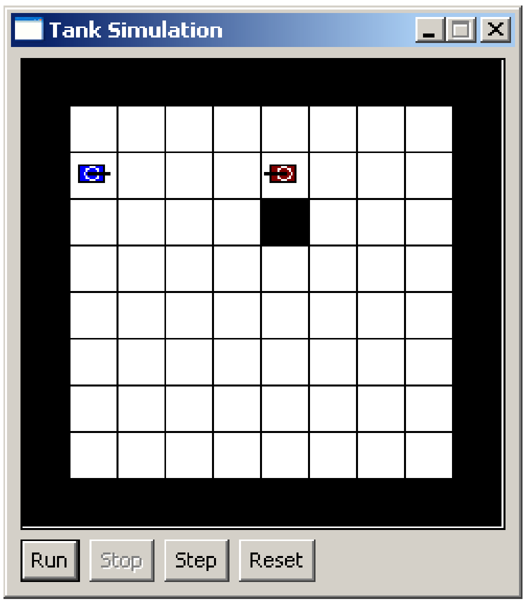


Figure 6.12 Situation after completing manoeuvre

The *attack experience* expects to see the red tank all the time but as the simulation removes the destroyed tank it is not visible on the radar sensor. The expectation of the situation is violated in the *RPD* model and situation is required to be re-evaluated and in *Soar* it is implemented by putting it as a condition in the production that proposes attack operator. As the conditions for the proposal of the *attack* operator are not satisfied therefore attack operator is removed and so is the *sub-state* created because of it.

The situation is re-evaluated and advance is selected which as discussed earlier is an abstract operator and creates an operator no-change impasse to create a *sub-state* to implement it.

The agent repeats *move* and *turn* actions after selecting them by evaluating through mental simulation and reaches its destination shown in Figure 6.13.

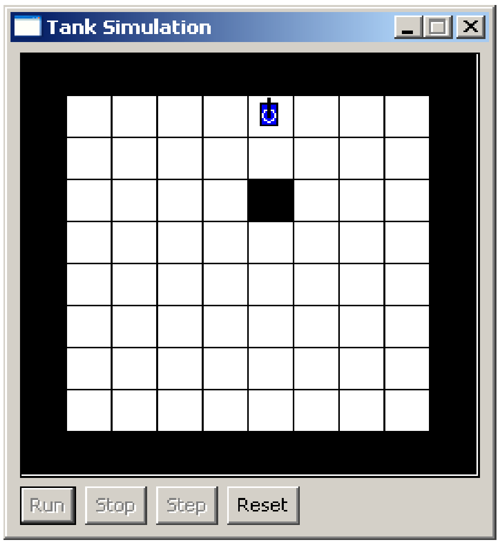


Figure 6.13 Blue tank reaches its destination

On completing the mission as in military operations and reaching the goal state as in *Soar*, the agent needs to halt and the simulation stops either for final termination or reset for another run. If the simulation needs to be terminated, the agent is stopped with the help of *Soar* production rules using *halt* command inherent in *Soar*. But if the simulation needs to be stopped and reset for another run then it needs to be done at the level of *environment* by stopping the agent and changing all the variables of the *environment* and the perception of the agent including the location of entities to the initial settings. The halt command irreversibly terminates the execution of the *Soar* program and should not be used when the agent needs to be restarted. This method has not been used in this implementation because the *Soar* program is run within a simulation which needs to be restarted for the next run until the number of required simulations is reached.

6.8 Integrating artificial neural network in the architecture

In rule based systems the antecedents of a production rule have to match exactly for the production to fire. If the current situation deviates from the conditions in the rule then the appropriate rule does not fire. Due to rule matching through an efficient algorithm like *RETE* and also advances in computer technology it is possible in *Soar* to add a large number of production rules to handle generalization. The *RETE* algorithm efficiently solves the many-to-many matching problem encountered when rules are matched to facts (Forgy, 1982). Writing large number of rules is possible but is not an efficient method of solving this problem. Alternate approaches like similarity-based generalization, fuzzy logic and artificial neural network may solve this problem in a more efficient way. In this implementation, an artificial neural network is used for situation recognition. There are two reasons for using an artificial neural network in this implementation: first, it has already been used for a similar task with promising results (Liang et al., 2001); second, it has the ability to automatically prioritize the situations according to their level of similarity.

A simplified diagram of the integration of the artificial neural network is shown in

Figure 6.14. The situations are fed to the trained artificial neural network which matches the new situation to one of the known situations and gives the agent a recognized situation. The recognized situation has the complete set or a subset of its four constituents that are goals, courses of action, cues, and expectations. The agent selects the course of action for the situation and implements it with the help of lower level actions selected through mental simulation if required. It is worth mentioning here that mental simulation is not being used at a high level and is being used at a low level that is to select atomic actions in pursuit of the goal set at a higher level. The pool of actions or a single action is proposed depending upon the experience of the agent based on the recognized situation.

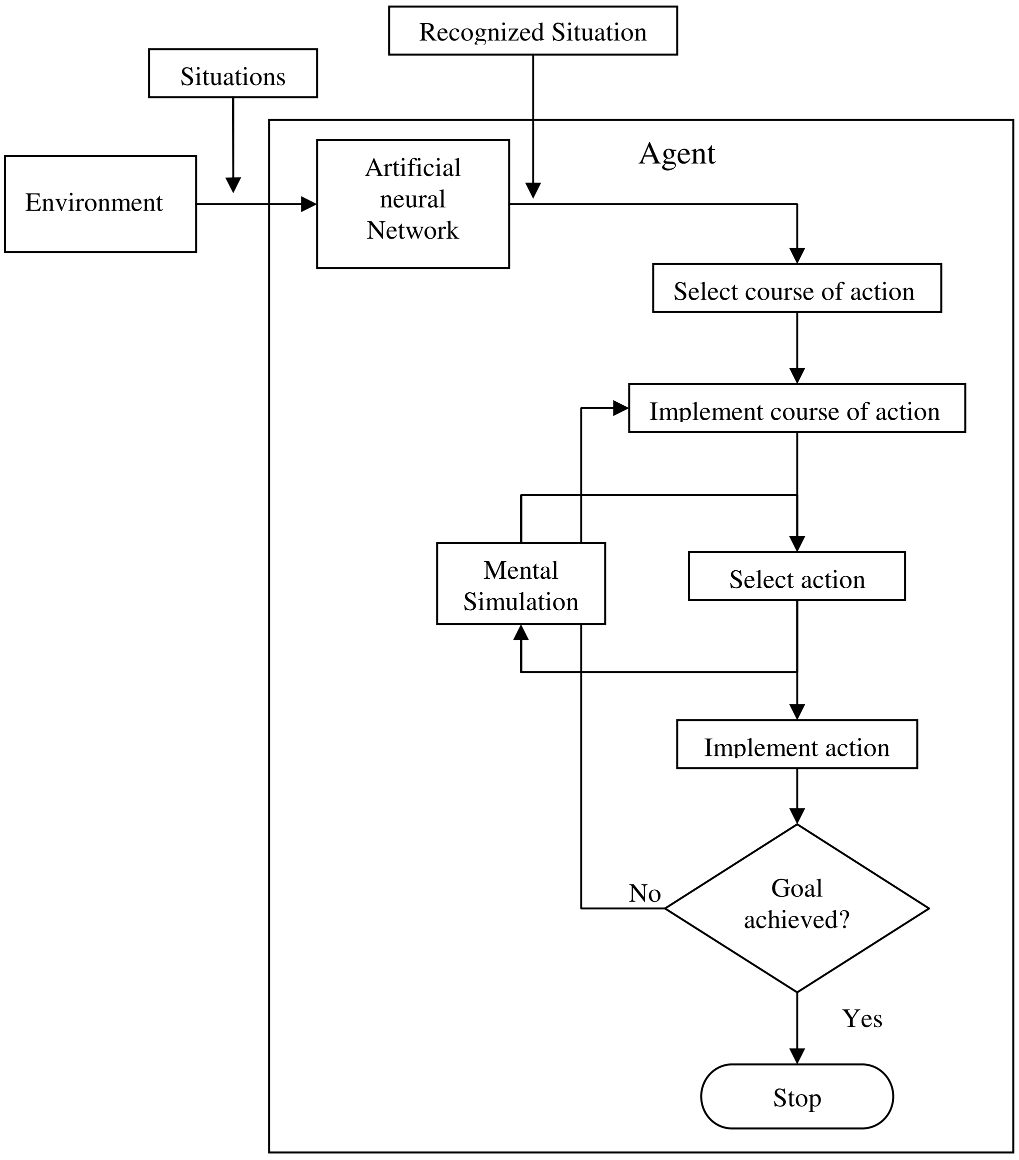


Figure 6.14 Integration of neural network in the architecture

The neural network is trained for each agent based on the range of situations it is likely to face. Motivated from the work of Liang et al. (2001), the neural net is a multi-layered normal feed forward network. It consists of an input layer of four nodes, three hidden layers of twelve nodes each, and the number of nodes in the output layer depends upon the number of known situations. The number of nodes in the output layer varies from situation to situation. For the experiments conducted in this research the configuration of the input layer and the hidden layers is not changed but these layers may also be reconfigured if required.

The standard back-propagation algorithm is used for learning. A dot product is used for the input to a node and a sigmoid function for the transfer function on all layers except for the output layer where a pure-linear function is used. The initial values of the weights are 1 and the network is trained for 1000 iterations with a constant learning rate of 0.01.

Matlab is used to train the network and then the simulator is implemented in Java so as to integrate the learnt net with the agent. The neural network is implemented as a Java class. Each output node represents a known situation, when a situation is given to the neural net then the output node with the highest value is selected and the corresponding situation is the recognized situation. The basic difference of this work and work of Liang et al. (2001) is that the latter uses the neural net for pattern recognition and plan generation at the same time and in this implementation the neural net is used for pattern recognition only. Liang et al. (2001) realize that his technique can be used only to reduce the search as all generated plans are not good solutions to the problem. In this implementation the *RPD-Soar* agent which has tremendous potential for reasoning with and implementing the plan is enhanced with pattern recognition capability of artificial neural network. The details of the neural net and its working in the model are further explained in the next chapter with an example experiment.

6.9 Summary

Soar provides a convenient framework to model most of the aspects of the *RPD* model. The elaboration phase in *Soar* decision cycle is used for situation awareness and the problem space based architecture, automatic *sub-goaling* and creation of *sub-states* due to impasse is used for mental simulation. The environment is developed in the Java object oriented programming language, the *RPD* model is implemented in the *Soar* cognitive architecture and the agent and the environment are interfaced with *Soar* mark-up language (*SML*). A trained artificial neural network is also integrated with the agent architecture to enhance the ability of the agent in handling new situations. The experiences of the command agent are stored in the *LTM* in the form of production rules. The success values for the courses of action for specific situations are represented numerically. All atomic actions, such as move, turn, fire, etcetera, expected to be performed by the agent in a simulation are coded by the modeller. The selection of an action for a specific situation in pursuit of single or multiple goals based on corresponding success values is the task of the *RPD-Soar* agent which forms the behaviour of the agent. This behaviour emerges at the simulation run time.

Part of implementation especially production rules specific to the agent in an experiment is explained within the experiments in the next chapter.

7 EXPERIMENTS, RESULTS, AND DISCUSSION

In this chapter, a number of experiments are discussed to demonstrate all the decision making strategies and processes adopted by *RPD-Soar* agents with varying degrees of expertise in different situations. Situational analysis is common to all types of agents used in various experiments because some form of information processing is always required based on the situational variables presented to the agent. Situational variables are the elements of the environment that form a situation in these experiments, e.g., location of an obstacle, the destination, and the agent's own tank etc. In these experiments the agents are using three types of decision making processes. The first type of decision making process is a case of definite recognition of a situation with only one possible course of action and the agent implements it without mentally simulating it. The second type relates to recognition of a situation with more than one course of action and then serially evaluating all of them one after the other through mental simulation to select the best suitable course of action for the present situation. Then there are situations where the agent sufficiently recognizes the situation to know which course of action is plausibly the best for the present situation but is not sure and therefore it evaluates the course of action through mental simulation and implements it only if it satisfices, otherwise the agent throws it away and mentally simulates the other applicable courses of action.

The preliminary experiment aimed at verifying *Soar*'s ability to store situations consisting of cues, goals, expectations, and courses of action in its *LTM* and bringing them up at logically correct time in its *WM* to produce the desired behaviour is already discussed in Chapter 6. In this chapter, the experiments discussed are aimed at demonstrating the flexibility in decision making and evaluating the performance and behaviour of various types of *RPD-Soar* agents. This will also demonstrate behaviour variability across agents, test the ability of the agent to recognize a situation in a changing context, test the mental simulation capability of the agent for dynamic situations, and demonstrate within agent behaviour variability, and adaptability of the agent. The last experiment is related to the integration of a trained neural network in the architecture to enhance the situation recognition ability of the agent.

This chapter contains some information related to implementation. It would be preferable to keep the scope of this chapter restricted to experiments and results only but some part of implementation is better understood in its context in this thesis due to the particular nature of the research.

7.1 Experiment 1 - Varying performance due to experience

In this experiment, the flexibility in decision making of the *RPD-Soar* agent is demonstrated. As the agent changes the decision making strategy according to its experience the change in performance is measured. This experiment demonstrates the possibility of generating agents with varying degrees of experience that exhibit the same behaviour but the time taken in decision making, represented by the number of *Soar* decision cycles consumed in making the decision, varies according to the experience. This experiment demonstrates the ability of the agent to change decision making strategies according to the availability of knowledge which is what humans do. This can be used to produce across-entity variability for command agents in military simulations based on the agent's expertise in the task assigned. By giving the agents a choice to select from all acceptable actions within-entity variability is also produced. In this experiment initially three types of agents are compared and then the performance of two *RPD* agents with different levels of experience is evaluated.

7.1.1 Vignette A - Static obstacles

In a 10 x 10 grid based environment, the tank has to start from the south and advance towards north to reach the destination as shown in the Figure 7.1. The agent has four actions to choose from: *move* in the direction that the tank is facing; and turn in any three directions other than the one that the tank is already facing. The tank has only the visual sensor that sees one cell around itself. The agent has been given the location of the destination cell and has been tasked to advance to that location. Although, there is only one obstacle in this environment, this obstacle always comes in the path of the agent unless the agent is moving randomly.

7.1.2 Random-walk agent

This agent has no experience. It only knows a set of actions that may be taken in a situation. For example, in the current state that is the starting position in Figure 7.1, the agent is facing north with an empty cell in its north, four actions: move north; turn east; turn west; and turn south are proposed. The agent has enough intelligence to avoid obstacles in the field and on the boundaries. It is avoiding obstacles by considering the actions that collides the agent with the obstacle as non-applicable actions. There are two ways of doing it. One way is to propose these operators and then give them low preferences. The other way, which is implemented in this experiment, is not to propose them at all. The agent has the ability to remember a pool of applicable actions in each situation and avoid collision with the obstacles but it does not have the capability to evaluate or mentally simulate actions and then select either the best or a better one out of them which can take the agent close to its goal state. Therefore, the activity may be called a random walk or searching the target location with brute force. The agent might well visit the same location many times. When the agent reaches its destination it recognizes its goal state and stops.

7.1.3 Less experienced *RPD-Soar* agent

This agent has the capability of a third-level *RPD* agent to mentally simulate the actions to find out how the world will change if current action is taken. The agent knows its destination, although, it can not see the destination unless it is in the adjacent cell to it. The agent knows the distance to and direction of its destination. Like the random-walk agent this agent also proposes all applicable actions and avoids collision with the obstacles. This may also be termed as experience of an agent as the agent knows there is no advantage to colliding with an obstacle. In *RPD* terms the agent already knows the low or zero ‘success value’ of this course of action in this situation.

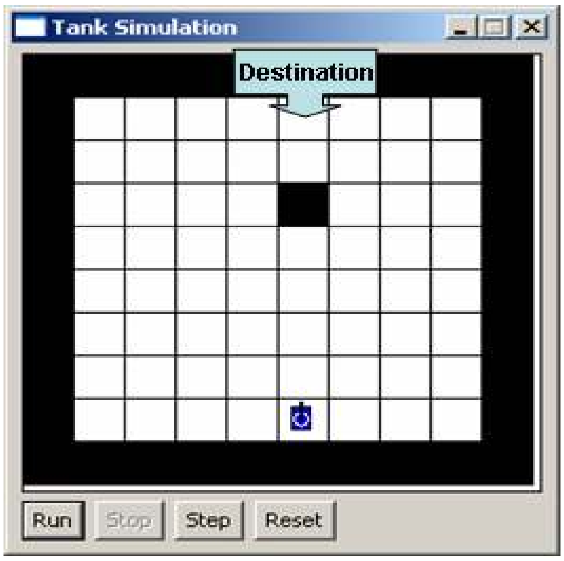


Figure 7.1 Simulation environment. The Blue tank is located at cell in the middle of bottom row, and is heading north. There is only one static obstacle located at the cell in the middle of fifth row in the north of the tank. The destination is marked in the middle of top row.

Out of the remaining actions the agent does not select one action straight away because it has not recognized the situation completely. This means the recognition is not specific for an action for the agent to behave like Level 1 *RPD* agent rather it gives a pool of actions for the present situation. The agent is a Level 3 *RPD* agent but it does not have sufficient experience to select one action as the first one to consider for mental simulation. Rather this agent due to its lack of experience, indifferently selects each action turn by turn and mentally simulates it to find if implemented will the action under consideration take the agent close to its present goal or otherwise. After the evaluation of each one of them this agent selects the most promising action.

7.1.4 Experienced *RPD-Soar* agent

Like the *Random-walk* and *Less-experienced* agents, this agent also proposes all applicable actions and avoids collision with the obstacles. And like the previous agent this agent is also not recognizing the situations straight away so it can not behave like

Level 1 *RPD* agent. That means all applicable actions for the present situation are proposed and the agent does not have enough experience in the form of success values to prefer one operator over the other. This agent is also a Level 3 *RPD* agent and has the capability to mentally simulate the proposed courses of action. Now at the stage of mental simulation the agent knows that if there is a move action among the actions that require to be evaluated then it has more chances of making the agent progress to its goal. Therefore, unlike the previous agent it has enough experience to recognize the situation and an associated course of action as the first one to consider for mental simulation. It creates a mental model and simulates the prioritized course of action through a single step and if the action seems promising applies it to the external environment and does not evaluate other applicable actions. As the agent is not testing other actions so as to know what they have to offer, it may be said that the agent is not optimizing rather it is satisficing. Satisficing is a decision making strategy which does not attempt to find an optimal solution rather it tries to meet criteria for a set threshold in a solution of the problem. But if mental simulation results in a negative evaluation value for the selected course of action which means if implemented in the real world this course of action will take the agent away from the goal then the complete mental model is removed and the course of action is rejected. And a new mental model is developed to evaluate the next course of action.

For this agent, apart from the first course of action all courses of action are equally preferable for selection for mental simulation. In this case, the courses of action are not prioritized for mental simulation for every situation rather a general preference is given to the move action over turn action. This is due to the nature of the problem, because the agent does not move any closer to its goal when the agent turns at its present location. Moving to a new location is what will take the agent closer to its goal but at the same time it can take the agent away from the goal, therefore, move is not selected straight away but is considered first for mental simulation. In other problems the agent may need to have a preference for a particular action in one situation and a preference for a different action in another situation of the same problem for mental simulation. Therefore, it will not be possible to generalize the preference for one action for all the situations in a problem. The *Soar* production rule that prefers the move action over turn shown in Figure 7.2 is put in this problem as a default rule as part of productions for selection space.

```
sp {selection*prioritise*evaluate-operator
   :default
   (state <s> ^name selection
            ^operator <o1> +
             ^operator <o2> +)
   (<o1> ^name evaluate-operator
              ^superoperator.name move)
   (<o2> ^name evaluate-operator
              ^superoperator.name turn)
   -->
   (<s> ^operator <o1> > <o2>)
}
```

Figure 7.2 Production: selection*prioritise*evaluate-operator

7.1.4.1 Evaluation criteria

The courses of action are evaluated for their suitability in achieving the goal. In this problem the goal is to reach a location in the environment marked as the destination. For simplicity, the courses of action are evaluated for reducing the distance between the cell marked as destination and the cell in which the agent is located and also for avoiding obstacles. In order to do this, the present Manhattan distance of the agent from its current location to the destination is recorded and then the action being evaluated is applied in the mental model. Recall that the Manhattan distance between two points is defined as the respective differences of abscissas and ordinates of the two points. After the action is taken then again the Manhattan distance is calculated and the difference is one of the evaluating factors. The production rules calculating these two Manhattan distances are shown in Figure 7.3. The *WME state ^tried-tied-operator* is created when the operator required to be evaluated is also selected in the decision phase for application. And as the production *RPD*elaborate*state*manhattan-distance* tests the ^tried-tied-operator *WME* therefore, the Manhattan distance is calculated just before the application of the selected operator for correct comparison.

```
sp {rpd*elaborate*state*manhattan-distance
  (state <s> ^name rpdsoar-ms1    ^desired <d>
               ^bluetank <bt>     ^tried-tied-operator)
  (<d> ^bluetank <dbt>)
  (<bt> ^x <bx> ^y <by>)
  (<dbt> ^x <dbx> ^y <dby>)
-->
  (<s> ^mhdistance (+ ( abs ( - <dbx> <bx>)) ( abs ( - <dby> <by>))))
}

sp {rpd*elaborate*state*present-manhattan-distance
  (state <s> ^name rpdsoar-ms1    ^desired <d>
           ^bluetank <bt>      -^io    ^operator <o>)
  (<d> ^bluetank <dbt>)
  (<bt> ^x <bx> ^y <by>)
  (<dbt> ^x <dbx> ^y <dby>)
-->
  (<s> ^present-mhdistance (+ ( abs ( - <dbx> <bx>))
       ( abs ( - <dby> <by>))))
}
```

Figure 7.3 Productions: to calculate Manhattan distances for evaluation

```
sp {rpd*prefer*operator*turn*west-and-east
  (state <s> ^name rpdsoar-ms1
           -^io
            ^bluetank <tank>
            ^map.cell <c>
            ^operator.actions.turn.direction << east west >>
            ^desired.bluetank.y < <y>)
  (<tank> ^x <x> ^y <y>)
  (<c> ^x <x> ^y <y>)
  (<c> ^north.content obstacle)
-->
  (<s> ^obstacle-factor 1)
}
sp {rpd*prefer*operator*turn*north
  (state <s> ^name rpdsoar-ms1
           -^io
            ^bluetank <tank>
            ^map.cell <c>
            ^operator.actions.turn.direction north
            ^desired.bluetank.y < <y>)
  (<tank> ^x <x> ^y <y>)
  (<c> ^x <x> ^y <y>)
  (<c> ^north.<< east west >>.content obstacle)
-->
  (<s> ^obstacle-factor 2)
}
sp {rpd*prefer*operator*move*west-and-east
  (state <s> ^name rpdsoar-ms1
           -^io
            ^bluetank <tank>
            ^map.cell <c>
            ^operator.actions.move.direction << east west >>
            ^desired.bluetank.y < <y>)
  (<tank> ^x <x> ^y <y>)
  (<c> ^x <x> ^y <y>)
  (<c> ^north.content obstacle)
-->
  (<s> ^obstacle-factor 3)
}
```

Figure 7.4 Productions: to evaluate actions for manoeuvring obstacles

The other evaluation factor is measuring the ability of the course of action in making the agent quickly manoeuvre obstacles. The production rules evaluating the actions for manoeuvring obstacles are shown in Figure 7.4.

7.1.5 Results

Thirty simulations are run for each agent and the result of comparing all three types of agents is shown in Figure 7.5. Thirty samples are taken so that the *Central Limit Theorem* will mean that a Normal approximation to the distribution of results will enable certain statistical tests to be applied. The y-axis represents *Soar* decision cycles that each agent is using to get to the same destination in the same environment from the same starting position. If the Random-walk agent is compared with even the *Less-Experienced RPD-Soar* agent the difference is notable (Table 7.1). It is worth mentioning here that the number of moves made in the external world by both *RPD-Soar* agents is far less than the number of *Soar* decision cycles as these agents do mental contemplation using *Soar* decision cycles while the *Random-walk* agent physically moves with every *Soar* decision. Most of the time during the simulation the *Random-walk* agent displays behaviour which does not look intelligent to the observer while the other two agents with the ability to mentally simulate their actions before implementing them in the real world display a plausible intelligent behaviour to the observer.

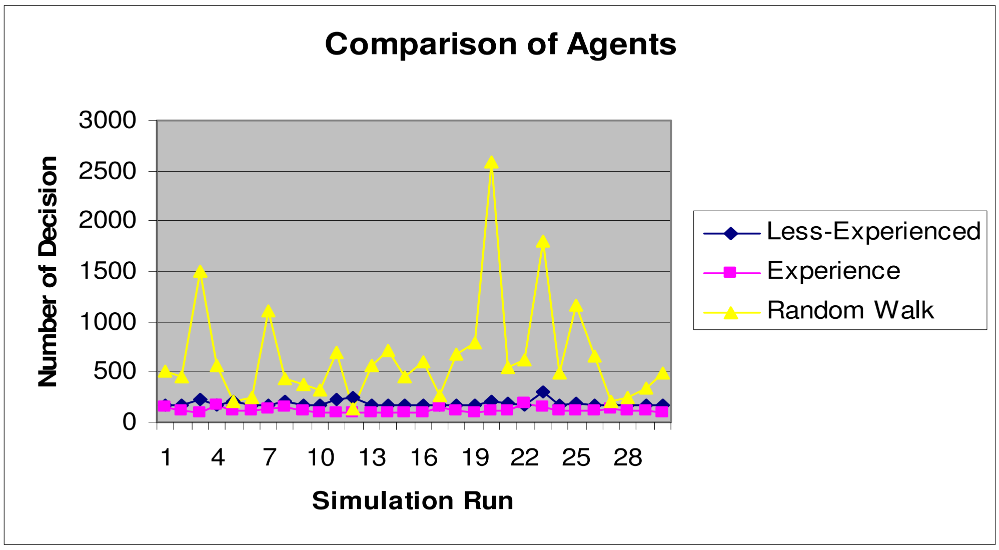


Figure 7.5 *RPD-Soar* agents vs. Random-walk agent

The intelligence in the behaviour of the *RPD-Soar* agents can be further improved with more knowledge and experience. The agents with more intelligent behaviour are used in experiments discussed later in this chapter. The across-entity variability in behaviour of the agents produced due to varying experience is clearly visible in Figure 7.5.

Table 7.1 Performance of *Random-walk* and *RPD-Soar* agents

Agent	Mean	Variance
Random-walk	656.37	514.93
Less-experienced *RPD-Soar*	184.50	29.90
Experienced *RPD-Soar*	113.97	25.18
Difference between Mean(less-experienced) and Mean(experienced)	≈ 70	

A two tailed t-test is performed on the simulation data of less-experienced and experienced *RPD-Soar* agents which confirms that the two means are different at 95% confidence level with $p = 1.19 \times 10^{-13}$. In two tailed t-test there are two hypotheses, H_o: $\mu_1 = \mu_2$ versus H_a: $\mu_1 \neq \mu_2$, where μ_1 and μ_2 are the two means being compared. In this case, μ_1 and μ_2 are the means of less experienced and experienced *RPD-Soar* agents respectively approximated by the means of sample data. At 95% confidence level if p is smaller than the significance level of 0.05 then H_o is rejected. In this case p is much smaller than the significance level of 0.05 thus H_o is rejected and the means of less-experienced and experienced *RPD-Soar* agents are significantly different at 95% confidence level.

To highlight the differences between both types of *RPD-Soar* agents the two *RPD-Soar* agents have been compared in Figure 7.6.

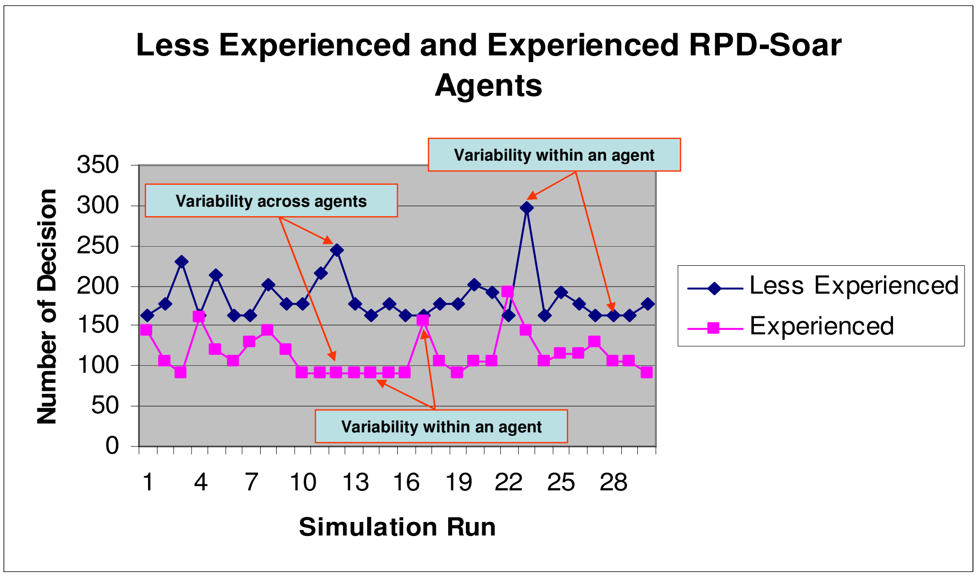


Figure 7.6 Less-Experienced vs. Experienced *RPD-Soar* agents

It is evident that comparatively more experience reduces the time and effort required for mental simulation by selecting one course of action to consider first and not evaluating other options if not required. The Less-experienced *RPD-Soar* agent generally consumes more number of *Soar* decision cycles to reach the destination. Behaviour variability within an entity is a desirable characteristic in a command agent and is more difficult to produce as compared to across-entity behaviour variability. It is defined as the variability in behaviour of the same agent in performing the same task over many episodes. Both *RPD-Soar* agents are displaying this variability. The within-entity variability in behaviour in this experiment is produced by giving choices to the agents so long as these choices do not take them away from the goal. Therefore, at no time during the simulation does the agent seem to be going away from the goal but some times the agent turns at one place and these turning actions can be wasteful because *Soar* decision cycles are consumed without the agent moving towards its goal. For example, if the agent is facing west whereas the destination is in the north, and we know that the agent can move only in the direction that it is facing but it changes its direction to south instead of north. This behaviour in the agents is improved by giving preference to turn actions towards north if the north cell is not blocked by an obstacle.

7.2 Experiment 2 - Changing Context

The basic concept in *RPD* is to recognize a situation in a changing context; therefore, in this experiment we have given the same situations as in the previous experiment to the same agents in changed contexts. The experiences should be sufficiently general to be applicable in a changing context. But if the experiences are over generalized then they will apply at places where they are not required and produce incorrect behaviour. In this experiment the agents have been tested to recognize situations in changing contexts in two different environments; the first environment (Figure 7.7) is an extended version of the environment of Experiment 1 (Figure 7.1), while in the second environment the number of obstacles is increased moreover, the size of the obstacle itself is doubled Figure 7.9.

7.2.1 Effect of enlarged environment on agents

The scenario is kept the same and only the environment is changed from a 10 x 10 to a 100 x 100 grid. The start point, the obstacle on the way and the target location are all kept at relatively the same distances by stretching out proportionally (Figure 7.7).

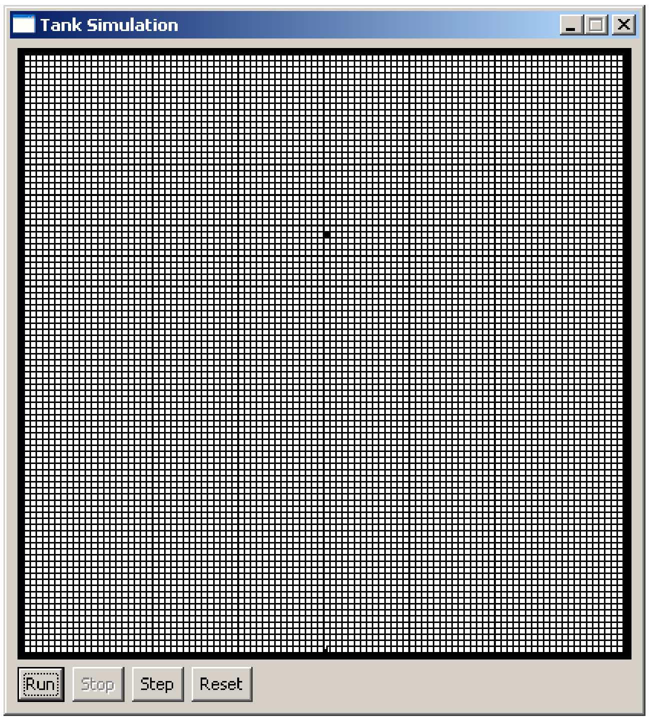


Figure 7.7 Enlarged environment

The number of decisions made by the Random-walk agent in reaching the target location increases exponentially, sometimes taking hours to reach the goal state, as is expected for such a large scale environment. The Random-walk agent is not discussed any further in the results. However, both of the *RPD* agents recognize the situations in the changed context and their behaviour remains that of expert agents as required of *RPD* model as per their levels of expertise as shown in Figure 7.8. The only difference in the results from the previous experience is that the agents have consumed more *Soar* decision cycles as is expected because of the requirement of more decision making by the agents and thus a higher number of evaluations.

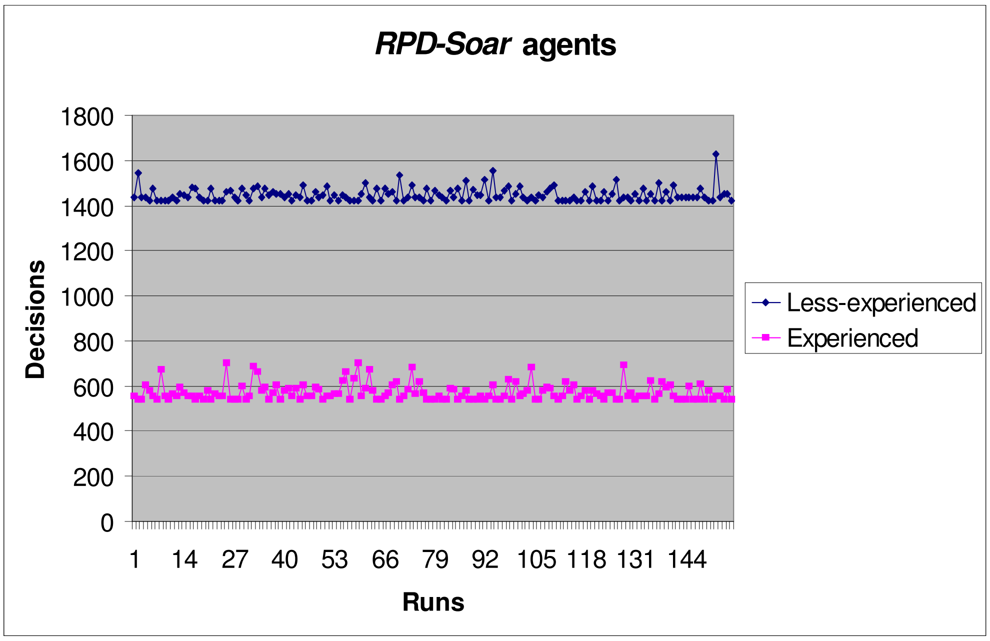


Figure 7.8 Less-Experienced vs. Experienced *RPD-Soar* agents in an enlarged environment

One important point to consider is the advantage of experience is prominent in this enlarged environment as compared to the previous environment. The difference in the means of the *Soar* decision cycles consumed by the agents in the previous environment is approximately 70 (Table 7.1) while in enlarged environment it is approximately 877, (~ 60% reduction in decision cycles) (Table 7.2).

Table 7.2 Performance based on Soar decision cycles of *RPD-Soar* agents in an enlarged environment

Agent	Mean	Variance
Less-experienced *RPD-Soar*	1448.33	30.84
Experienced *RPD-Soar*	571.76	37.63
Mean(less-experienced)-Mean(experienced)	≈ 877	

7.2.2 Changed obstacle pattern

In this experiment, the scenario is kept the same and the environment is changed to give the same agent an entirely changed context by placing a complex pattern of obstacles in the field to manoeuvre to reach its destination (Figure 7.9). The agent is designed to manoeuvre only a single-cell obstacle, but in this environment the same agent is exposed to an obstacle occupying two adjacent cells.

The aim of the experiment is to observe whether the agent still recognizes the situation and the associated course of action when it looks at a two-cell obstacle instead of one-cell obstacle. The agent successfully manoeuvres the obstacle in this environment and takes the same action of moving to east or west after recognizing the situation from the previous environment of an obstacle in its north. If it decides to move east, the number of *Soar* decision cycles would not change but if it decides to move west, the number of decisions increases in the range of 20 – 30. But in both cases, the agent manoeuvres the obstacles and finds its way to the destination.

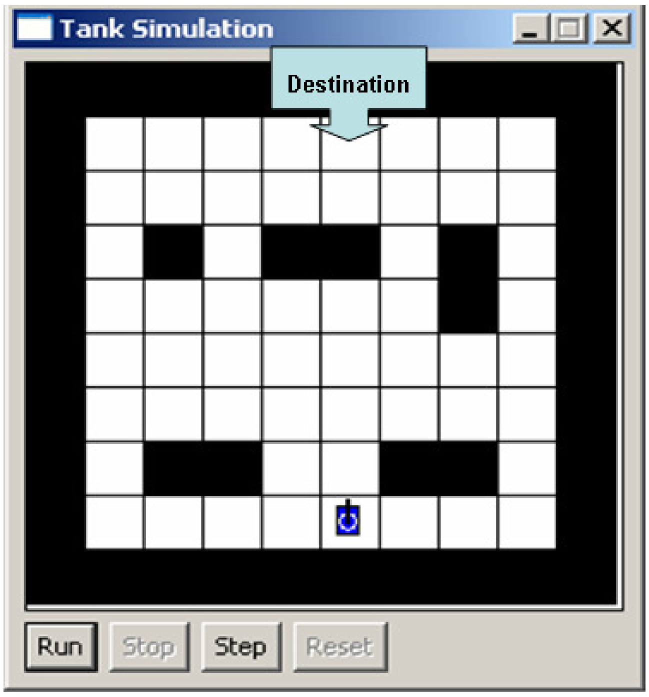


Figure 7.9 Changed Obstacle pattern

7.3 Experiment 3 - Variability within an entity

In this experiment, a more mobile Blue agent with the ability to move to all eight neighbouring cells reaches its destination by manoeuvring around static obstacles and avoiding collision with an equally mobile Red agent. Previous experiments are carried out to verify the ability of *RPD-Soar* agent to recognize a situation based on cues and sometimes only on a single cue with overwhelming significance. And also the ability of the agent to recognize the associated goals, expectations, further cues to look for in this situation, and an action with the highest success value. The abilities of *RPD-Soar* agent to take advantage of the opportunity arising from a situation by recognizing plausible goals and to change its decision making strategy with experience is also demonstrated. The aim of this experiment is to demonstrate, analyze and discuss within-entity variability in behaviour of *RPD-Soar* agents. Variability in behaviour is discussed very briefly in Experiment 1 and Experiment 2, but in this experiment variability in the behaviour of an *RPD-Soar* agent is discussed in detail.

7.3.1 Explanation of the experiment - Moving threat

In this experiment (Figure 7.10), the Red agent is moving diagonally, starting from the bottom left destined to top right. The Red agent is not intelligent and is following a prescribed route. The Blue agent has the same capabilities and the same goal as that of the *RPD-Soar* agent described in Experiment 1 or Experiment 2, but in this experiment it only has the visual sensor and does not have the radar sensor, and there are two improvements; one is the ability to avoid collision has been added and the second change is that it has been made more mobile and flexible. Now, it does not have to change directions before moving to any cell and it can also move in four more directions of north-east, north-west, south-east and south-west. Therefore, at each step, the Blue agent has at most eight actions to choose from.

7.3.2 Mental simulation to avoid Collision

If the Blue agent keeps pursuing its initial goal without changing its goals according to the situation then the Red agent and the Blue agent are due to collide in the next cell as indicated in Figure 7.10. As the Blue agent can see only one cell around itself, therefore, the Blue agent detects Red agent on its west when the Blue and Red agents reach the locations as shown in Figure 7.10. The Blue agent takes its turn to act first and then the Red agent takes its turn. As both agents move one cell at a time, therefore, the Blue agent knows that Red agent can go to the location where the Red agent is now, or any one of the other locations marked with red circles (light grey) as shown in Figure 7.11. Therefore, there are three safe locations for Blue agent to avoid collision, one where it is located now and the other two are marked with blue circles (dark grey) in Figure 7.11. Thus, Blue agent moves east to avoid collision which in this case happens to be exactly the opposite direction from the present location of Red agent.

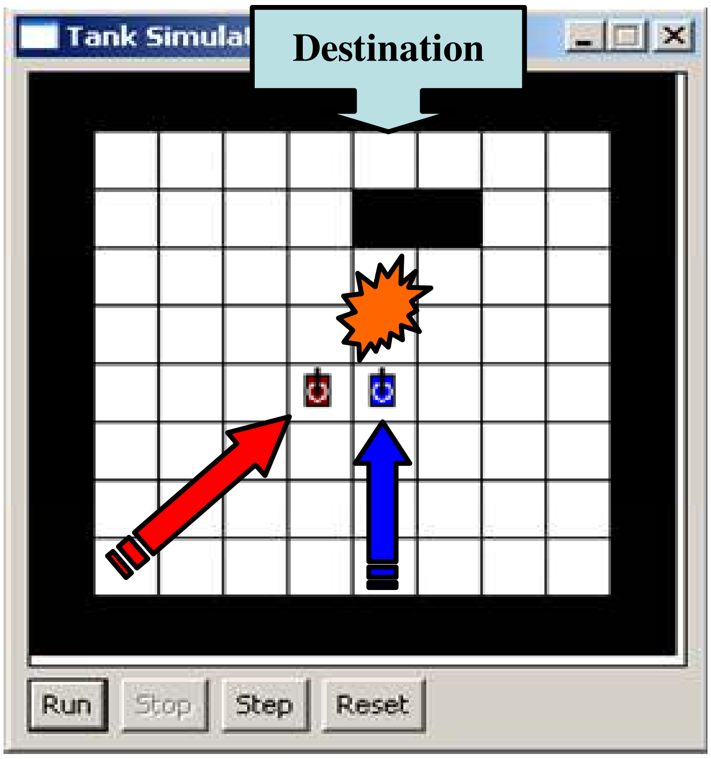


Figure 7.10 Collision course

The situation and the corresponding action discussed above is only one example from the set of situations presented to and actions taken by the Blue agent. In order to check the ability to recognize the situation in different contexts, the Red agent starts from four different locations on the west and four different locations on the east of the Blue agent. And moving diagonally it tries to collide with the Blue agent at different locations. The Blue agent successfully recognizes the situations in all cases and avoids collision with the Red agent. One hundred simulations are run for each case to observe the variability of the agent's behaviour for the same situation. The appropriateness of choosing to carry out one hundred replications will be tested later.

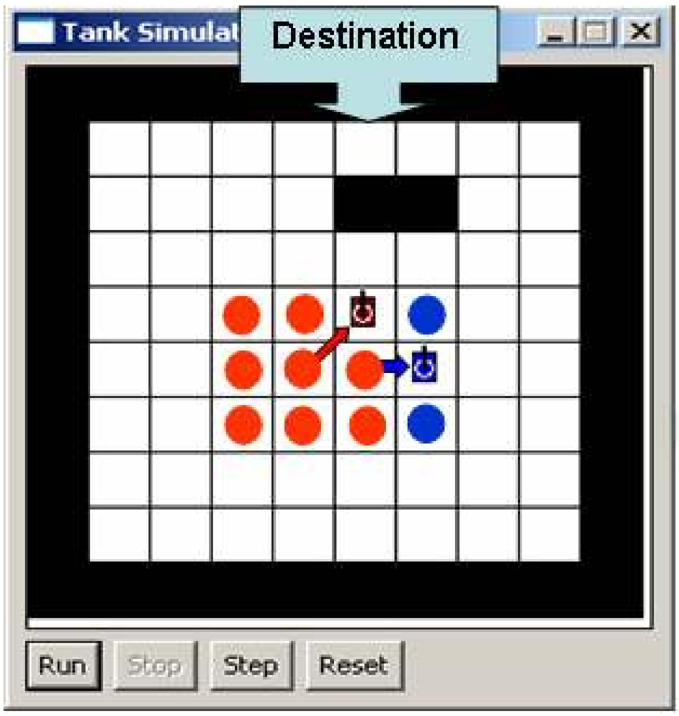


Figure 7.11 Mental simulation to avoid collision

7.3.3 Factors affecting the decision of the Blue agent

The evaluation of an action during mental simulation is based on the difference in Manhattan distance, relative position of static obstacles and moving Red tank with respect to the Blue tank, and also the last action taken by the Blue agent.

In the mental model of the Blue agent the world is modelled as the Blue agent sees it. The Manhattan distance from the Blue agent to the destination is measured before starting the mental simulation and then the move action in the direction that is required to be mentally simulated is applied in the mental world. And then the Manhattan distance is measured again. If the agent is moving towards the destination then the difference in Manhattan distance is positive and if the agent is moving away then the difference is negative. Likewise, success values for other applicable factors are given to the action being evaluated. The criteria for assigning these success values to an action are given in the succeeding paragraphs. The success values given due to different factors to the action being evaluated are summed up and the action collecting the highest success value is selected.

7.3.3.1 Manhattan distance

The factor that always affects the decision of the Blue agent is the Manhattan distance which is the signed difference of Manhattan distance before and after the application of operator. The numeric value ranges from -2 to +2.

7.3.3.2 Static obstacle

In order to manoeuvre around static obstacles while the Blue agent is moving towards its destination in the north, the move to east or west is given a numeric value of +2 when there is an obstacle in the north of the Blue agent.

7.3.3.3 Red tank

The numeric values to evaluate an action of the Blue agent when it encounters a moving Red tank are as follows:

- +2 for moving to a cell exactly opposite to cell containing the red tank.
- -2 for moving to north or east cells if the red tank is in the cell north-east to the blue tank.
- -2 for moving to south or east cells if the red tank is in the cell south-east to the blue tank.
- -2 for moving to north or west cells if the red tank is in the cell north-west to the blue tank.
- -2 for moving to south or west cells if the red tank is in the cell south-west to the blue tank.
- -2 for moving to south, north, north-east, or south-east cells if the red tank is in the cell east to the blue tank.
- -2 for moving to north, south, north-west, or south-west cells if the red tank is in the cell west to the blue tank.
- -2 for moving to east, west, south-east, or south-west cells if the red tank is in the cell south to the blue tank.
- -2 for moving to east, west, north-east, or north-west cells if the red tank is in the cell north to the blue tank.

7.3.3.4 Undoing last action

A numeric value of -1 is given to the action if the action makes the Blue agent undo what it has done in the last turn.

7.3.4 Results

One hundred simulations are run for each starting location of Red agent. Different starting locations of Red agent correspond to different situations as for each starting location Blue agent sees the Red agent at either a different location or in a different direction. And in some cases both the location and the direction is different. The total number of behaviours for each case ranges from 19 to 24. The number of behaviours displayed by Blue agent in all situations faced by it is summarised in Table 7.3.

As in this experiment every path traversed by the Blue agent is a different behaviour, therefore, we have used the term path and behaviour interchangeably. The case when the Red agent starts from two squares east of the Blue agent is discussed in detail as this provides more space for the Blue agent to manoeuvre and therefore produces the maximum number of distinct behaviours.

Table 7.3 Number of distinct paths traversed by Blue agent in eight different situations

Start location Red agent	Visible to Blue agent	No. of Paths
4 squares west – 1 north (1, 7)	Step 3, north-west	20
3 squares west – 1 north (2, 7)	Step 2, north-west	22
4 squares west (1, 8)	Step 3, west	21
3 squares west (2, 8)	Step 2, west	19
2 squares east - 1 north (7, 7)	Step 1, north-east	22
2 squares east (7, 8)	Step 1, east	24
3 squares east – 1 north (8, 7)	Step 2, north-east	22
3 squares east (8, 8)	Step 2, east	21

In one of particular run, with the Blue agent starting at position (5, 8), the Blue agent moves first and reaches location 1 in Figure 7.12, the Red agent also moves to its location 1, and then the Blue agent detects Red agent and moves to its location 2 to avoid collision. The Blue agent keeps moving to locations 3 and then 4 to avoid collision with the Red agent until both reach their locations number 5. Here the Blue agent takes a risky decision and moves to location 6. In this point in time the Red agent on its turn can move down and collide with Blue agent, but it is moving on a prescribed route and therefore the Blue agent survives. In one hundred runs the Blue agent has chosen to adopt this route, requiring nine moves to reach the destination, only once. The Blue agent adopts the path, displayed in Figure 7.13, most frequently and selects it 16 times in one hundred trials. This risk aversive behaviour and also a non-optimal move to location 9 increased the length of the route to thirteen steps.

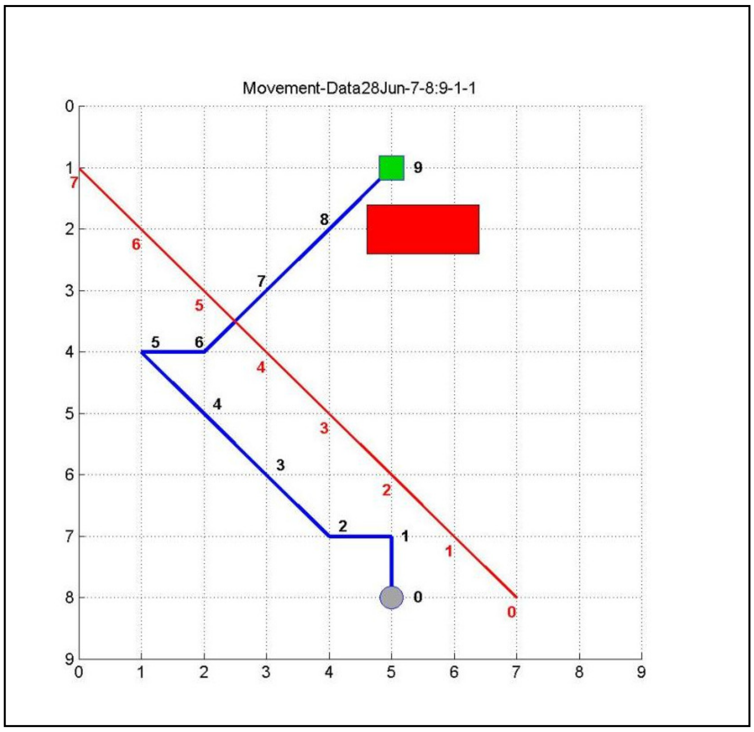


Figure 7.12 The least frequently used path. The labels with each path are depicting the step number of the corresponding agent.

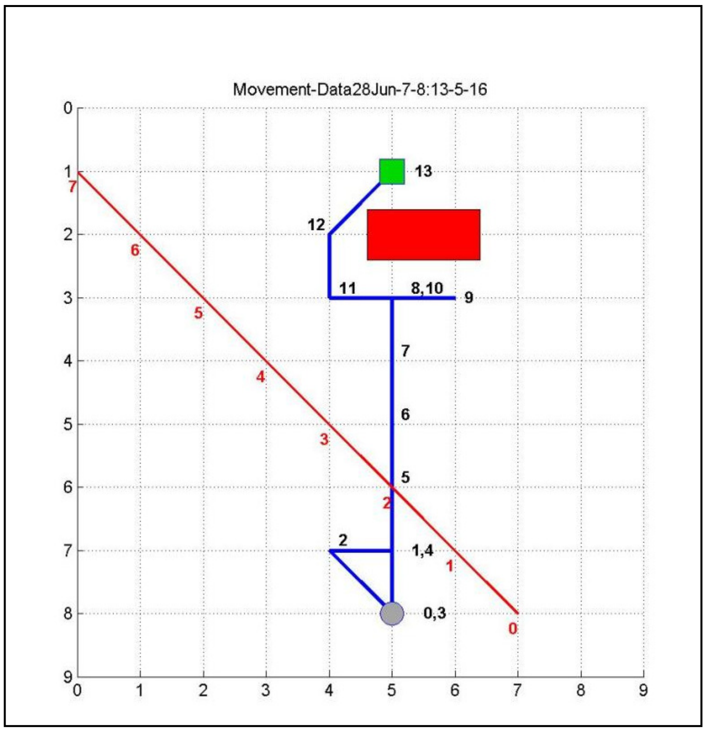


Figure 7.13 The most frequently used path. Some times the Blue agent traverses the same location number of times. The label on the left shows the step number earlier in time than the number on the right. 0, 3 means agent visits this location at step 0 and then comes to the same location in step 3.

In one hundred simulation runs for this case in which the Red agent starts from the same location and moves on the same prescribed route, the Blue agent finds twenty four distinct routes to the destination. These routes are shown in Figure 7.14 through Figure 7.17 for quick comparison.

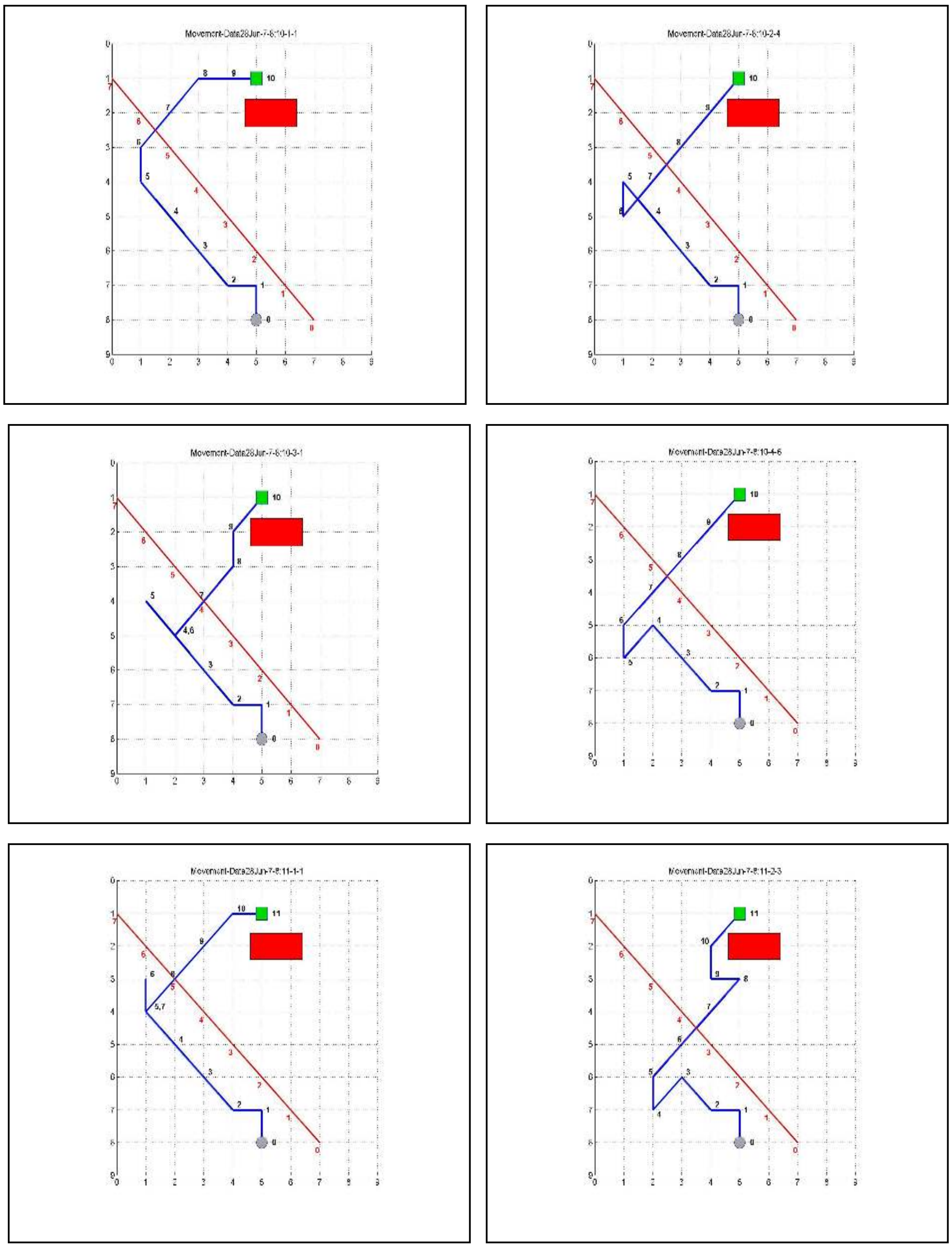


Figure 7.14 Behaviours of Blue agent for starting position of Red - (7, 8)

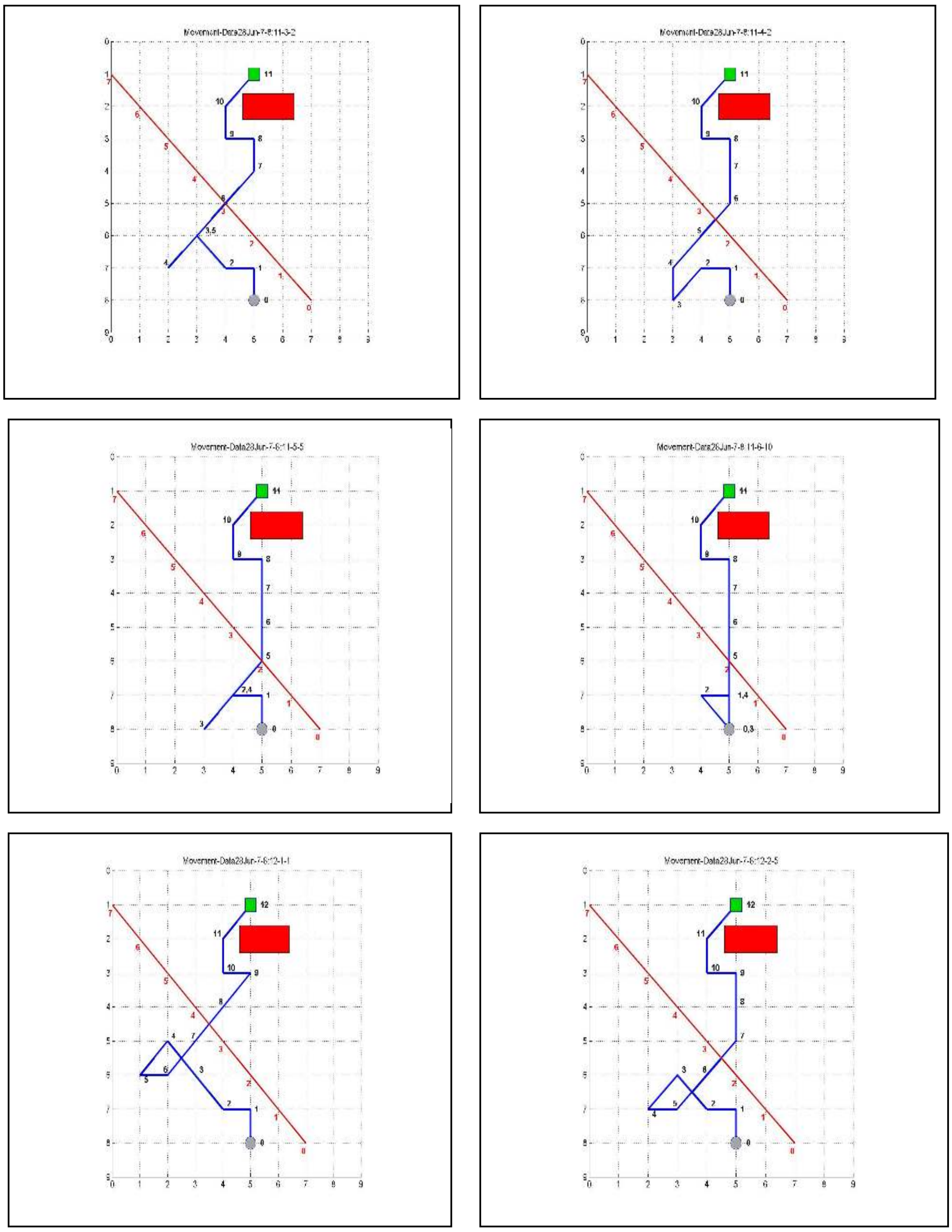


Figure 7.15 Behaviours of Blue agent for starting position of Red - (7, 8)

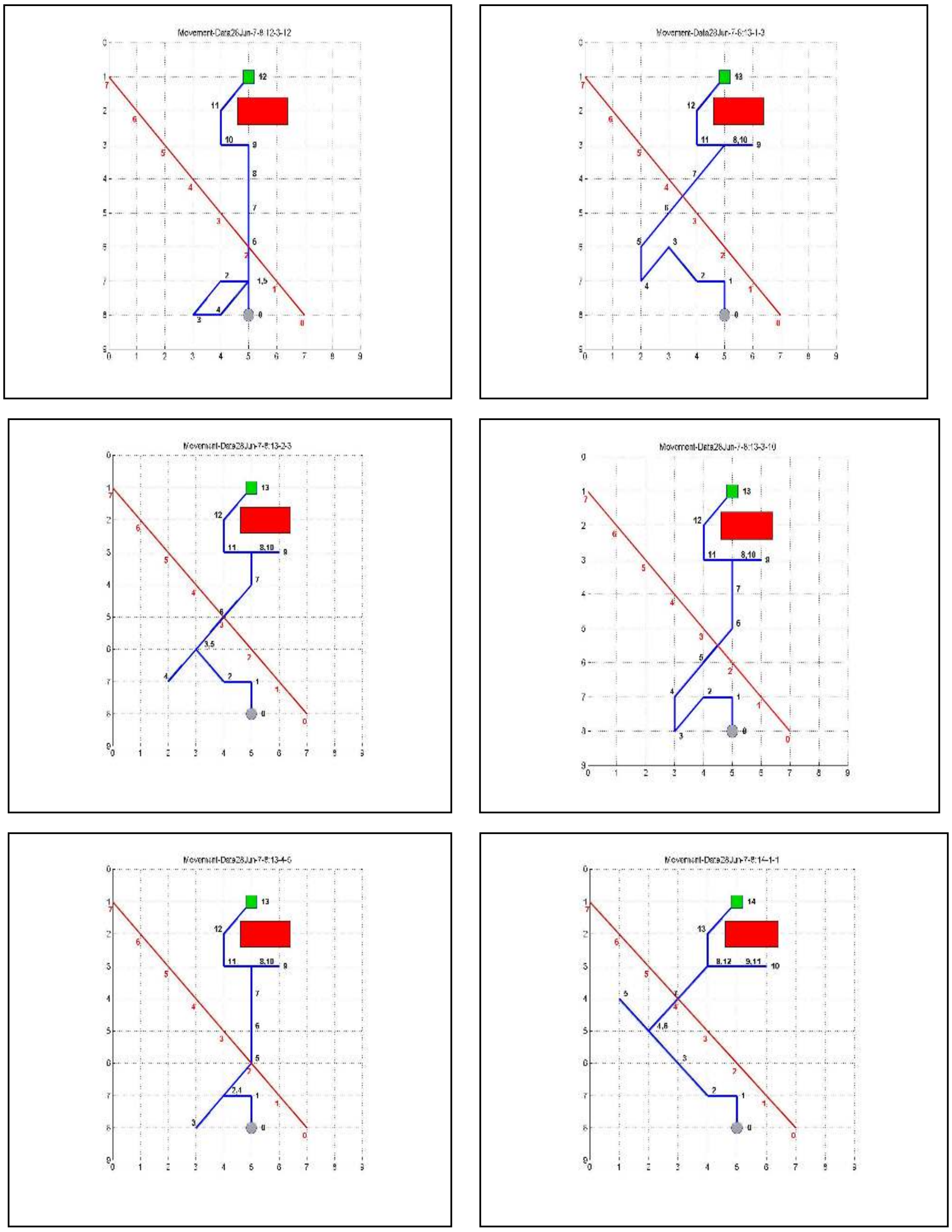


Figure 7.16 Behaviours of Blue agent for starting position of Red - (7, 8)

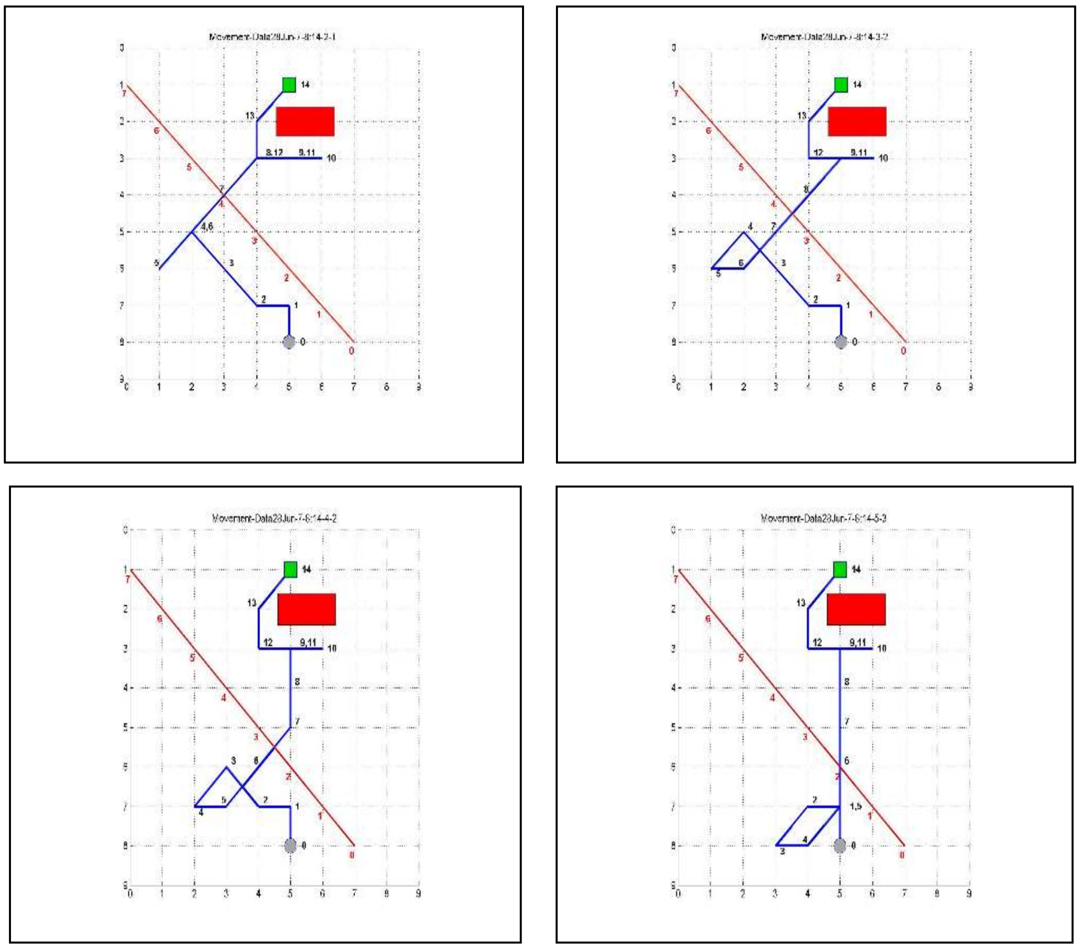


Figure 7.17 Behaviours of Blue agent for starting position of Red - (7, 8)

The probabilities of occurrence of these twenty four behaviours are shown in Figure 7.18 (bottom graph). Behaviours have been grouped according to their path lengths, and the probability distribution of these groups is displayed in the upper part of the same figure. For maximum variability in behaviour the paths should be equally distributed. The probability of a single behaviour or a group of behaviours in case of equal distribution is shown as red horizontal line for easy reference. While the proposed implementation achieved a reasonable spread of behaviours, there still remains some bias towards Group 5 (this relates to behaviours of path length 13).

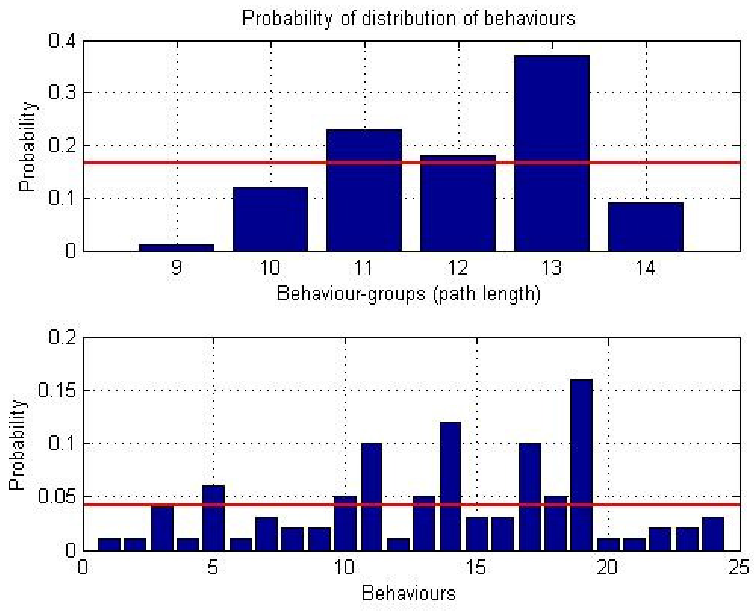


Figure 7.18 Probabilities of Blue agent's behaviour

Although, for maximum variability in behaviour, the paths should be equally distributed but one of the requirements of human-like behaviour variability is the presence of a hidden pattern in the behaviour. Correct individual behaviour and producing intended population-level distribution is a requirement on *HBR*. During training simulations the trainees should be able to identify patterns and take advantage of it (Wray and Laird, 2003). This agent is displaying a distribution with some behaviour more likely than others in a population. This is produced by fine tuning the set of numeric values given to different actions during mental simulation and a different behaviour may emerge as most favoured with a different set of values.

Behaviour patterns of the same agent for the same situation i.e., same starting location of Red agent (Red starting two squares east of Blue), for 100, 200 and 1000 simulation runs with path lengths are shown in Figure 7.19, Figure 7.20, and Figure 7.21.

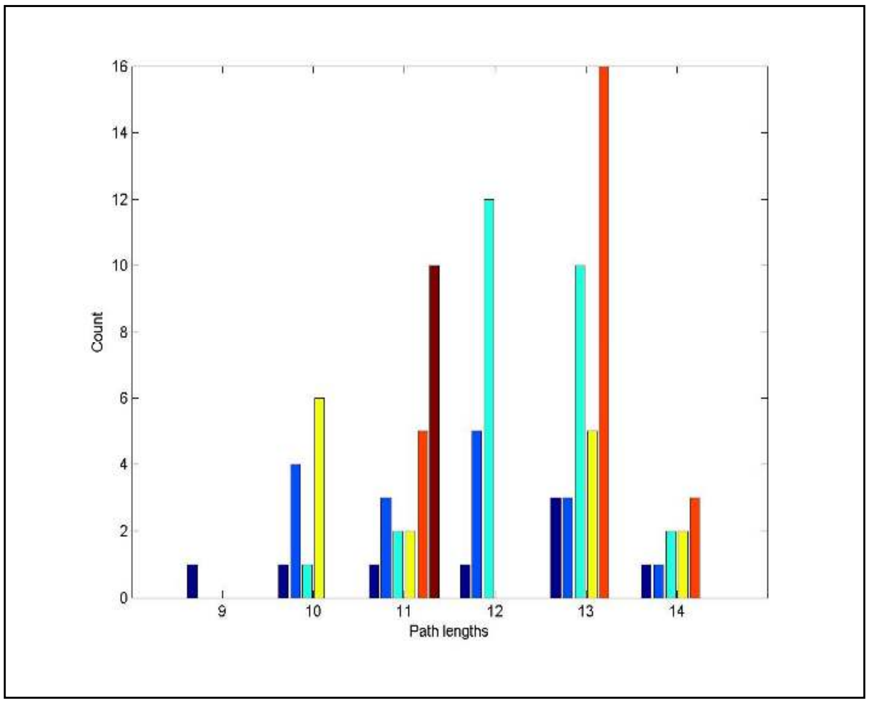


Figure 7.19 Behaviours of Blue agent over 100 simulation runs – Red starts: (7,8)

The same behaviour emerges as the most frequently occurring behaviour in 100, 200, and 1000 simulation runs. The most frequent behaviour for 100 simulation runs is shown in Figure 7.13, and for 200 and 1000 simulation runs are shown in Figure 7.22. The second most frequent behaviour in 1000 simulation runs is shown in Figure 7.23. This is exactly the same path with length 11 as that of the most frequently used path with length 13 (see Figure 7.13) except for one move that the agent makes just below the obstacle. The agent on encountering the obstacle has two options either to move east or move west. In the case of behaviour with path length 13 the agent moves east to avoid the obstacle but finds out that it will have to move further east and more away from the objective to manoeuvre the obstacle. When the agent evaluates this move in the direction of east, the Manhattan distance to the objective increases, which gives a negative success value to the move, therefore, it moves back towards west and then moves west again to clear the obstacle and move towards its objective in the north. In the case of behaviour with path length 11 the agent decides to move west instead of east as it encounters the obstacle and avoids the two moves in the direction of west therefore completes the goal in comparatively lesser number of steps. Except for this difference both behaviours are identical.

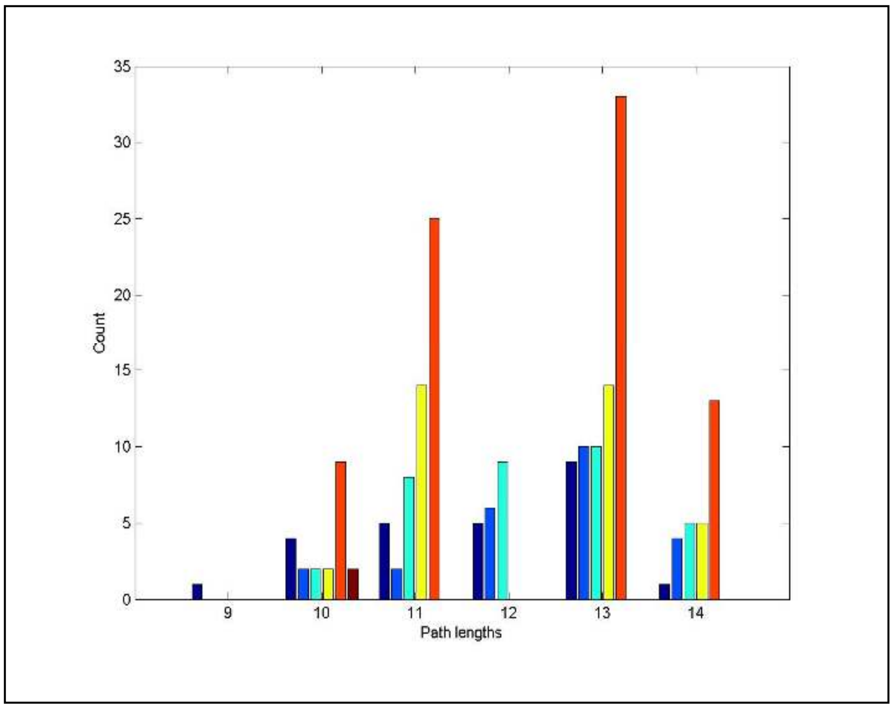


Figure 7.20 Behaviours of Blue agent over 200 simulation runs – Red starts: (7,8)

There is one more point to note in the 200 and 1000 simulation runs and that is the change in the total number of behaviours. In 200 simulation runs, two new behaviours are produced with path length 10 and one behaviour of path length 11 is not produced. Therefore, the total number of behaviours is increased by one making it 25. In 1000 simulation runs, all behaviours are included and the total number of behaviours is 26. There is a possibility that other behaviours exist that have not so far been generated.

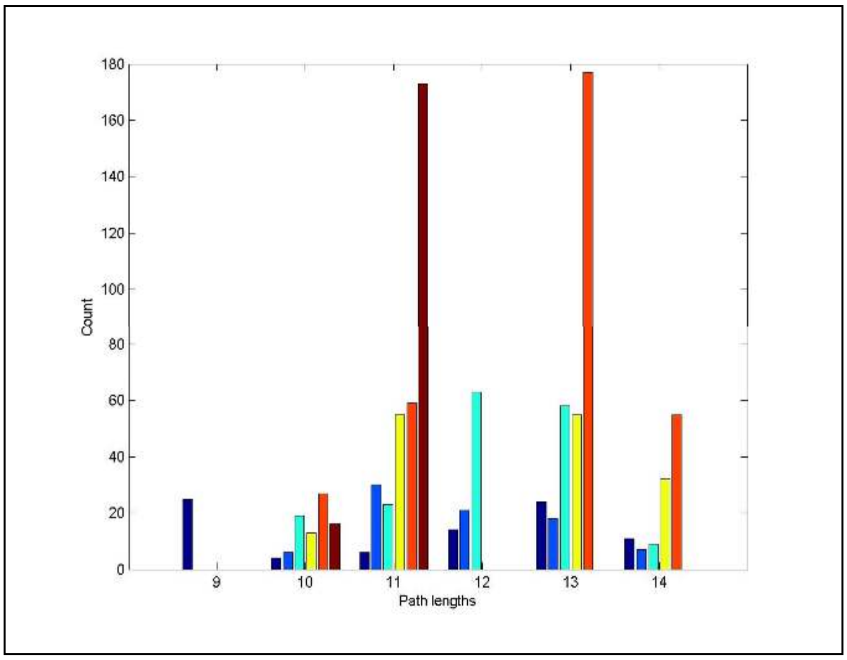


Figure 7.21 Behaviours of Blue agent over 1000 simulation runs – Red starts: (7,8)

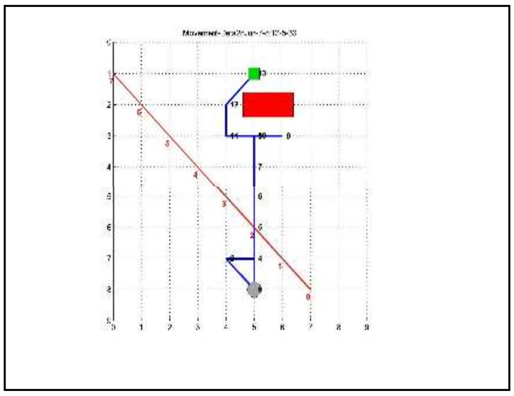
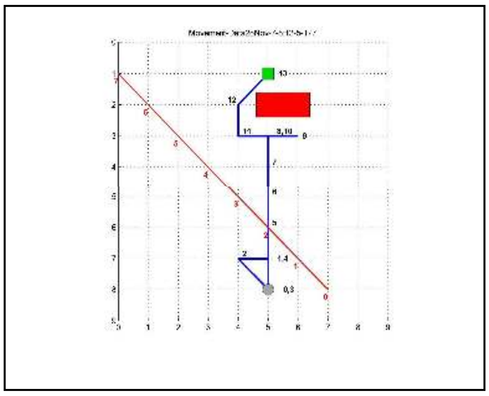
 

Figure 7.22 Most frequent behaviour for 200 (left) and 1000 (right) simulation runs

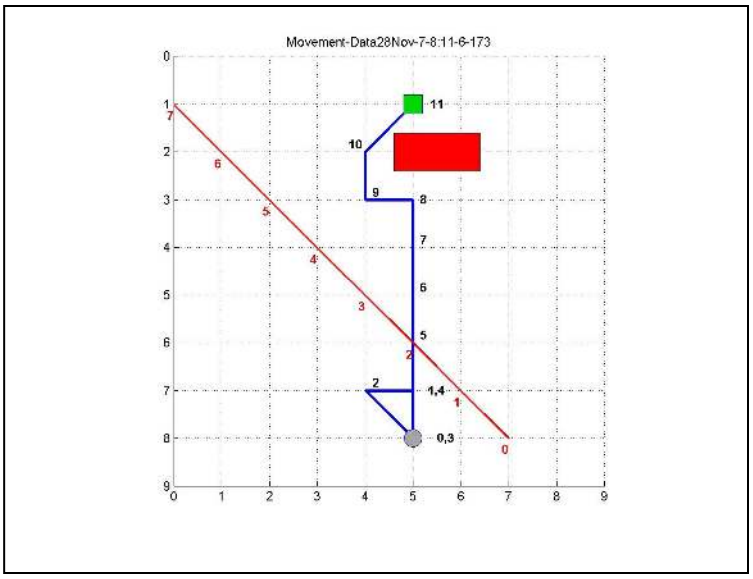


Figure 7.23 Second most frequent behaviour in 1000 simulation runs

The variability produced in this *RPD-Soar* agent is not due to randomness that produces undesirable behaviour rather it has been produced because of the reasonable but some times sub-optimal choices given to the agents.

7.4 Behaviours resulting from strategies formulated by humans

Three subject matter experts and two non experts were asked to give their recommended strategy to avoid the collision with the Red tank, if they have the same task with the same sensors as that of the agent in this experiment. Two strategies to avoid the moving agent were recommended by them. One strategy emphasizes safety, in which the Blue agent mentally simulates all possible future moves of the Red agent in the next step, and then the Blue agent selects its own move; which is one from the pool of all of the possible moves that can take it to a cell which can not be occupied by the Red agent in next step. The second strategy involves a calculated risk, in which, the Blue agent, where possible predicts Red agent's one move from all possible future moves of Red agent by observing Red's two previous moves. And then Blue takes a risk only if this risk takes the Blue agent closer to the goal by selecting to move to a cell which may possibly be occupied by the Red agent in its next move but is not the predicted one. The proposed *RPD-Soar* agent model demonstrated both of the behaviours that result from the use of both of these strategies formulated by humans.

Although, the ability to predict the most probable move from the history of Red agent's moves is not implemented, but this behaviour is also generated due to the success values of actions incorporated in the experiences. On the corners where the choices of moves of Blue agent is restricted and where the advantage of taking the risk is more than the negative success value of a risky action the Blue agent takes the risk.

As it is explained earlier in this chapter that the case where the red agent starts from location (7, 8) is discussed in detail because in this situation the red and blue agents start interacting with each other from the very first step in the simulation. The distributions of behaviours for rest of the seven cases for 100 simulation runs are shown in Figure 7.24 through Figure 7.30.

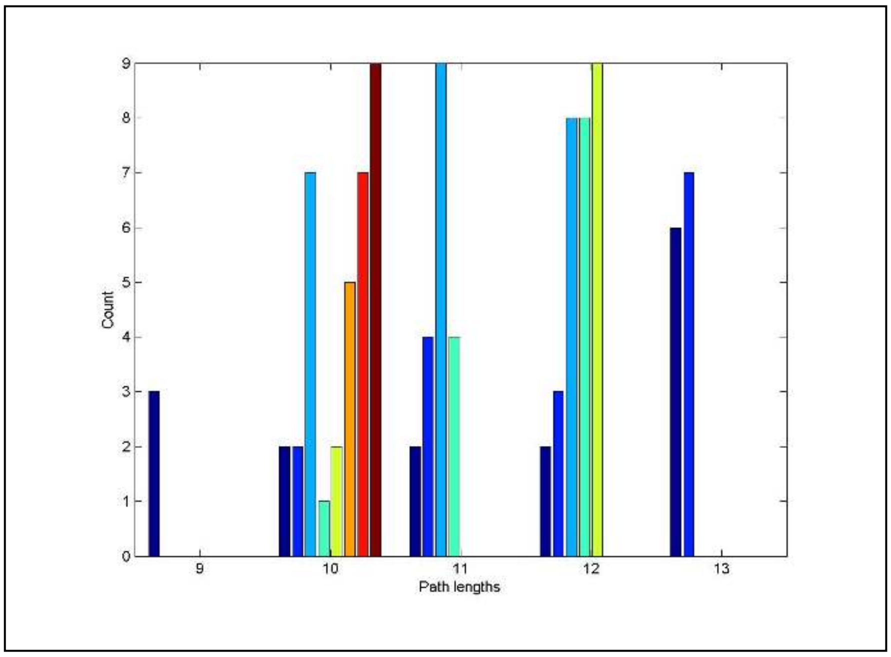


Figure 7.24 Behaviours of Blue agent over 100 simulation runs – Red starts: (1, 7)

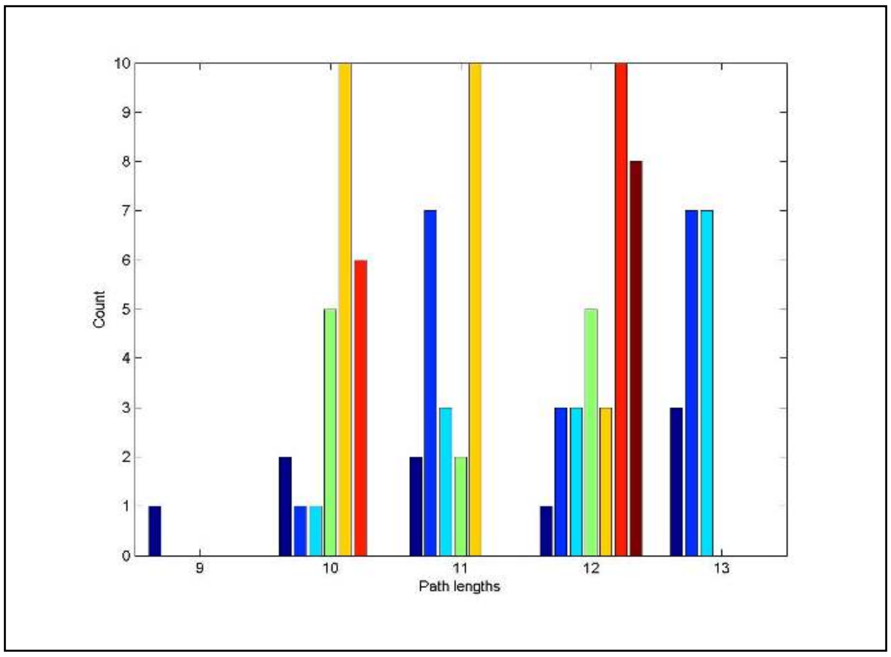


Figure 7.25 Behaviours of Blue agent over 100 simulation runs – Red starts: (2, 7)

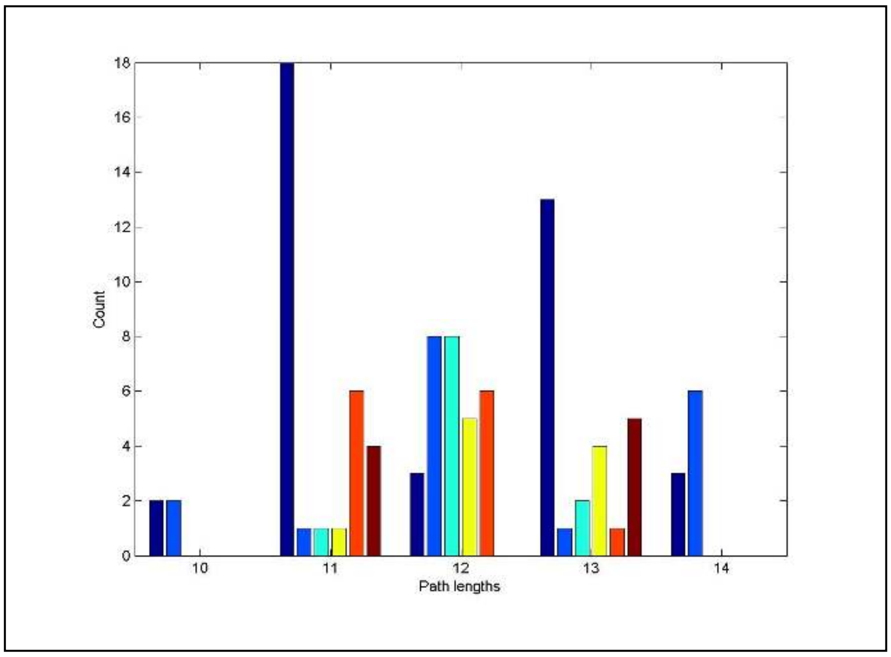


Figure 7.26 Behaviours of Blue agent over 100 simulation runs – Red starts: (1, 8)

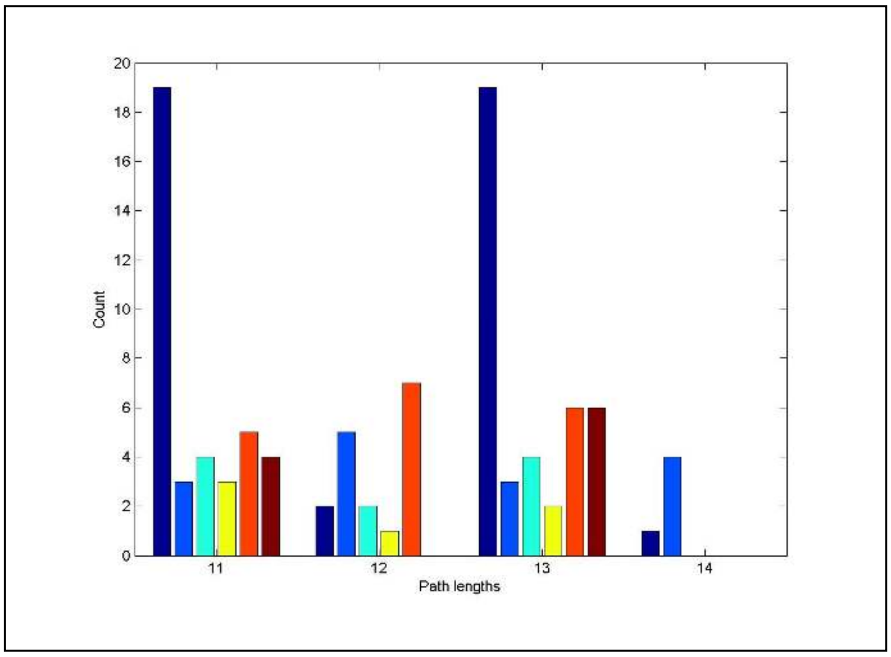


Figure 7.27 Behaviours of Blue agent over 100 simulation runs – Red starts: (2, 8)

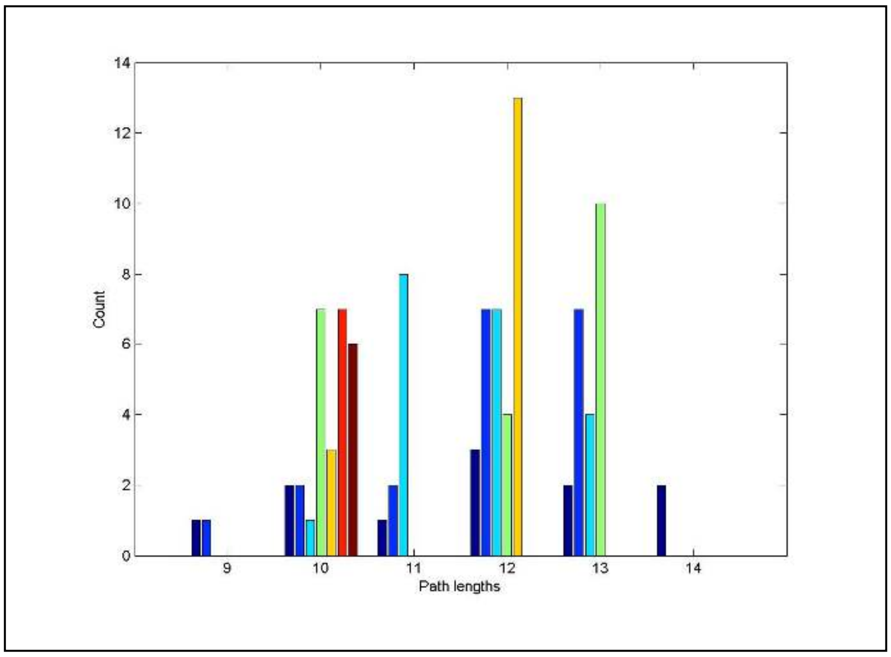


Figure 7.28 Behaviours of Blue agent over 100 simulation runs – Red starts: (7, 7)

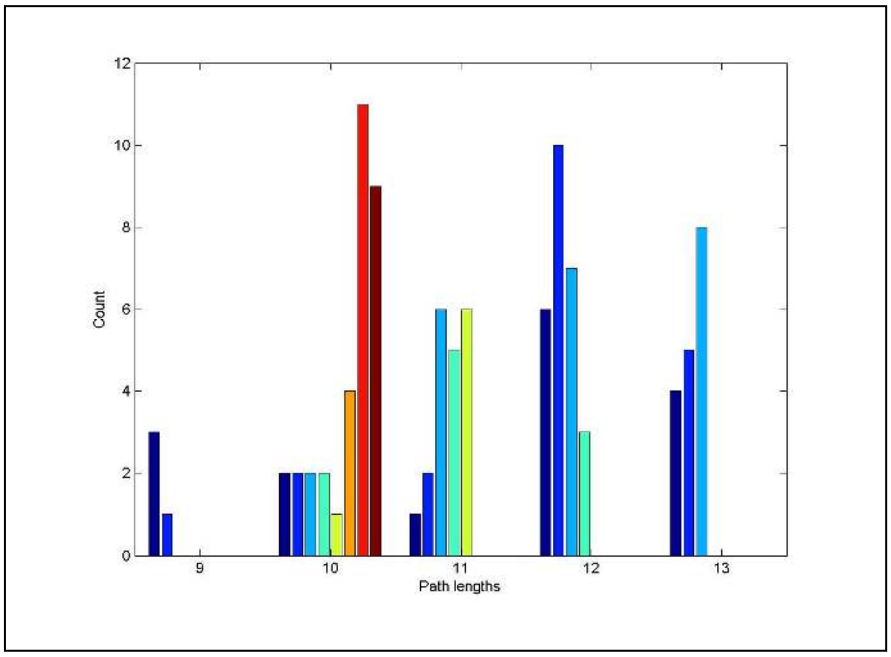


Figure 7.29 Behaviours of Blue agent over 100 simulation runs – Red starts: (8, 7)

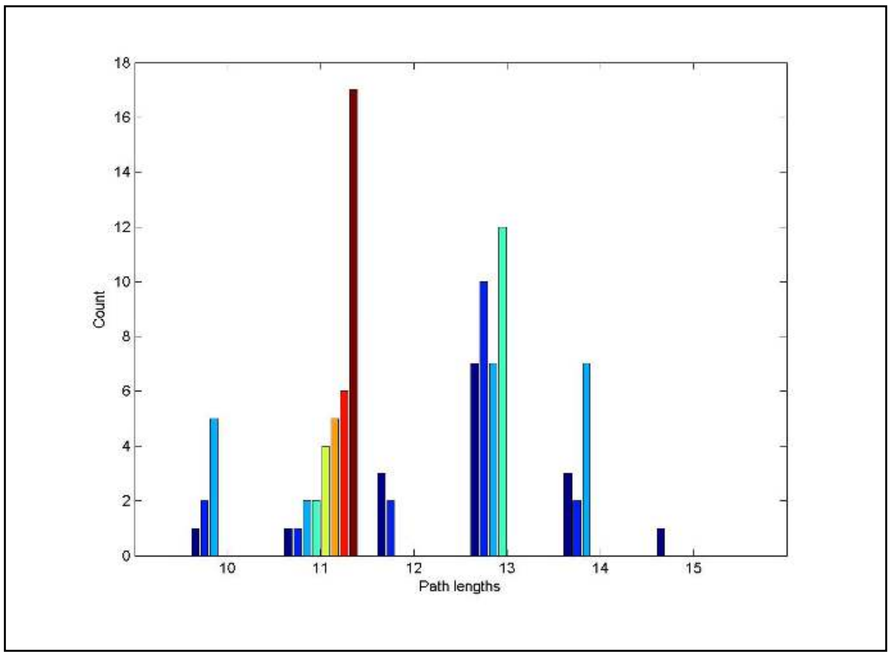


Figure 7.30 Behaviours of Blue agent over 100 simulation runs – Red starts: (8, 8)

The number of behaviours in one hundred simulation runs for these seven cases range from nineteen to twenty two depending upon the number of steps taken from the start of the simulation to the point where the agents start to interact. Results of all these experiments show a similar trend to that of the case that is discussed in detail in which the red agent starts from location (7, 8).

7.5 Experiment 4 - Learning

This experiment is about the adaptability of an *RPD-Soar* agent during the simulation that adds dynamism to the simulation environment. The experiments are aimed at observing the learning process in four different situations and then the transfer of learning from one situation to the other.

7.5.1 The change in the agent

The environment, the task, and the agents are same as that of Experiment 3, the only difference in the Blue agent is that more specific evaluation of behaviour of the agent in manoeuvring around an obstacle located north of the agent is added. In the previous experiment the agent is trained to handle single-cell obstacles and it does defeat a two-cell obstacle but half of the times the manoeuvre is not very efficient. When the agent moves west which is a random choice between the two choices of east and west, then the move seems intelligent but when it moves east then it has to come back to its original cell which is wasteful. In this experiment the agent is designed to handle two-cell obstacles in an efficient way. The agent while moving to its destination in the north finds the two-cell obstacle south of destination (Figure 7.31). The agent at this point not only recognizes an obstacle to its north but also identifies the obstacle in its north-east and knows that if it moves east still it will be blocked by an obstacle in the north; therefore, it gives a success value of 2 during mental simulation of move west action. The production rule in Figure 7.32 checks for the conditions in this situation during mental simulation and gives this success value as *obstacle-factor*. This is the only situation where a numeric value is given to an action in relation to the obstacle because if the agent is in the south of the east part of the two-cell obstacle then the agent moves west anyway because of the attraction of the agent towards the destination. And if there is a single-cell obstacle blocking the agent in any location exactly in the south of the destination, the agent may randomly chose one from the two choices of move east or west. To check the efficacy of the production it is tested in a difficult situation where both the obstacle and the agent are moved to extreme west from their present location displayed in Figure 7.31. Now the production in

Figure 7.32 seems to be pushing the agent in the wall by giving a success value of 2 to the action move west, but in this situation there is no proposed action as move west because the location to move to is an obstacle.

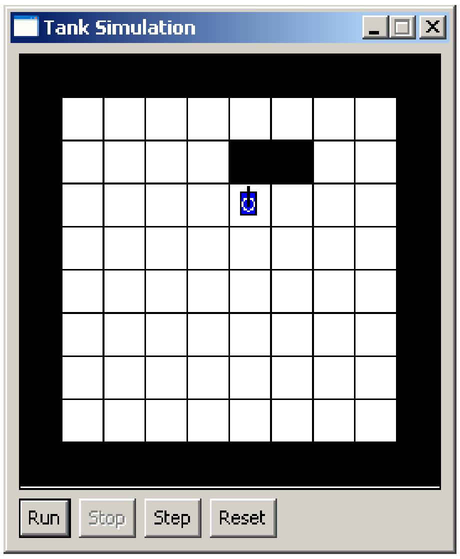


Figure 7.31 Agent south of two-cell obstacle

7.5.2 Learning method

The learning mechanism inherent in *Soar* is called chunking and is a form of explanation based generalization. Explanation based learning is a type of inductive learning. Inductive learning requires a certain number of training examples to achieve a given level of generalization accuracy. Artificial neural network and decision tree learning are examples of inductive learning. Analytical learning augments the information provided by the historical examples using domain knowledge and deductive reasoning. Deductive reasoning is that type of logical reasoning in which conclusions must follow from their premises (Giarratano and Riley, 1998). The use of domain knowledge and deductions aids the learning process and substantially reduces the number of training examples required for adequate learning. Explanation based learning belongs to this sub category of inductive learning.

```
sp {rpd*evluate*numeric-value*obstacle-factor*north
  (state <s> ^name rpdsoar-ms1
          -^io
           ^bluetank <tank>
           ^map.cell <c>
           ^operator.actions.move.direction west
           ^desired.bluetank.y < <y>)
  (<tank> ^x <x> ^y <y>)
  (<c> ^x <x> ^y <y>)
  (<c> ^north.content obstacle)
  (<c> ^north-east.content obstacle)
-->
  (<s> ^obstacle-factor 2)
}
```

Figure 7.32 Production: evaluate move west action - two-cell obstacle in the north

Due to the ability of the *RPD-Soar* agent to mentally simulate applicable courses of action it is possible to use the agent without rigorous training as it can handle new situations effectively. Given a high level task the agent, through mental simulation, finds out the sequence of implementation of low level tasks itself to achieve the aim of the high level task. The mental simulation takes time because during contemplation of the course of action a large number of *Soar* decision cycles are consumed. Therefore an untrained agent tends to be slow in deciding and taking an action compared to an experienced agent. It is worth mentioning here that both of the agents respond within real time. Through this learning technique it is expected that the number of *Soar* decision cycles which represents the time taken to make a decision reduces. In terms of *RPD* model it is a process of increasing expertise through experience and the same situation which is first handled through Level 3 *RPD* after training is handled through Level 1 *RPD*. For a given task the agent starts to learn and makes decision straight away in situations for which it has already learnt. The time required to complete a task reduces as the agent repeats the task again and again and the number of new situations reduce that the agent may encounter in this task. Due to this learning mechanism, the time taken by an agent to complete the same task becomes proportional to the experience of the agent. In a simulation, if there is a requirement of agents with varying level of experience to perform a task then it may be met by producing agents using this learning mechanism.

In *Soar*, the learning can be turned on and off. The *Soar* command that turns learning on is "learn --on". When learning is on then *chunks* are produced when an impasse is resolved. These *chunks* are straight away loaded in the *LTM* of the agent as soon as they are produced and are ready to fire like any other production rule present in the *LTM* of the agent. In this experiment and also the previous experiments when the simulation is reset then the agent is initialized which means the agent keeps its *LTM* as such and initializes only the working memory. As the chunks are loaded in the *LTM* therefore chunks remain stored in the *LTM* when the simulation is reset. When the simulation is terminated then the agent is killed and then creating the agent again requires all production rules to be reloaded in the *LTM* and chunks if not stored elsewhere are lost. The procedure for storing chunks and other data is shown in Figure 7.33. The chunks are stored after every simulation run and the data storage holds the chunks for all stages of learning. For example if a set of 50 simulations are run in one go, then the experience of this agent can be scaled from 0 to 50. All the learnt chunks can also be stored together after completing all the simulation runs with a little variation in the code.

7.5.3 The problem of over generalization in *chunking*

During experimentation on learning it is observed that sometimes the learned production rules (*chunks*) are over generalized and apply in situations where they are not required and thus produce undesirable behaviour. The point to note is that the problem representation where learning is involved has two aspects: the first aspect is the correct representation for problem solving; and the other is correct representation of the problem for learning (Ritter, 2007). When the problem is being solved correctly then the representation is correct for problem solving but it may or may not be correct for learning.

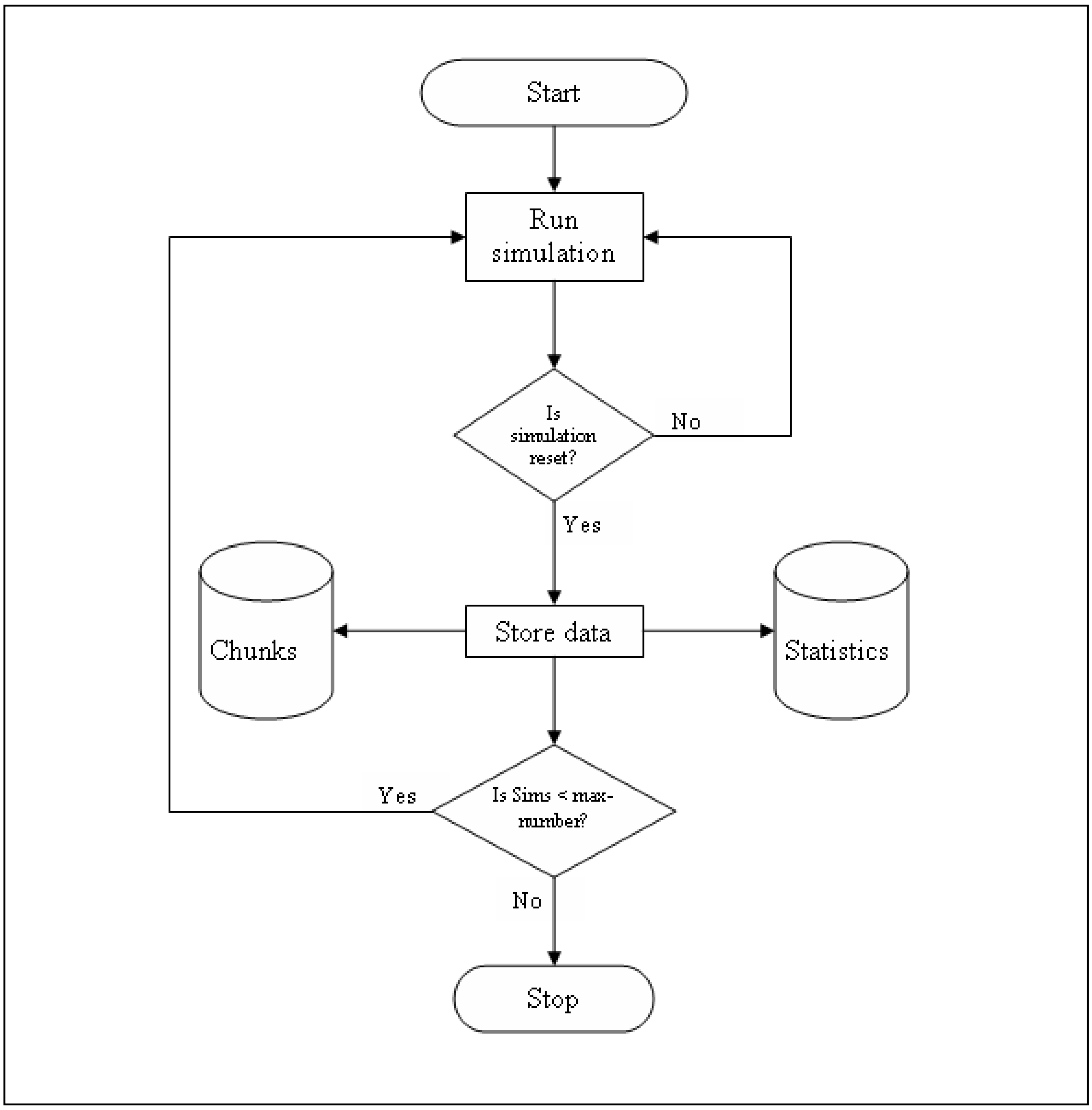


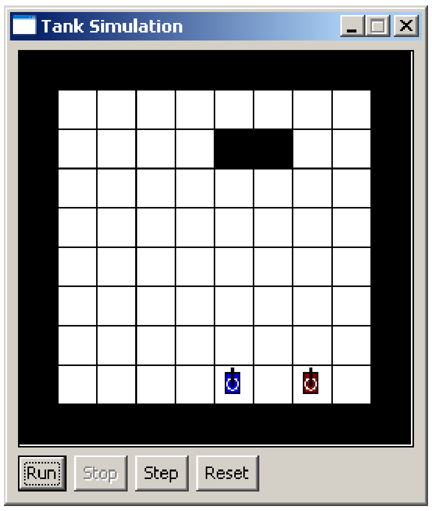
Figure 7.33 Storing process of chunks and statistics

For example, in our experiments the problem is being solved correctly and the Blue agent is avoiding collision with static and moving objects and reaching its destination every time but when learning is set to on then there is unresolved conflict of two operators. The problem is that in one simulation run when the Blue agent is at a certain location it finds Red agent in its neighbouring cell. The Blue agent mentally simulates the actions and finds one to avoid the collision and through chunking marks this action as the best suitable for the Blue agent when at this location. It is worth mentioning here that one of the antecedents to this learnt chunk tests for the presence of Red agent. In another simulation run, the Blue agent at the same location does not find the moving Red agent in its neighbouring cells, and mentally simulates actions and marks one action best suited for this situation which may be different from the chunk produced in the situation discussed above where the chunk is produced for selecting a course of action aimed at avoiding Red agent. Now in another simulation run, the Blue agent at the same location finds a conflict between these two actions when Red agent is present in the neighbouring cell. The reason for this conflict is the chunk that is learnt when there is no Red agent because it is more general and fires even when the Red agent is present. This is a case of over generalization. It is solved by dividing the applicable operator into two different operators, one operator is action-tank-present and the other is action-tank-not-present.

Initially when a chunk is created it contains actual identifiers of objects but this makes the chunk very specific and the chunk only fires when the actual objects are matched. To improve generality the identifiers of actual objects are replaced by variables. The constants in the conditions are not changed. One modification in the representation of the problem that can improve generality in the learnt chunks is to reduce the use of constants to a minimum. Reduction of constants improves the generality and also increases the quantity of transfer of learnt knowledge to other tasks. In our opinion, in this implementation if the locations of the agents, the destination, and the obstacles is represented in relative distance and directions from the point of view of Blue agent using spatial reasoning then the generality can be improved further.

7.5.4 First task – Red agent starting position- (7, 8)

The Blue agent starts from location (5, 8) and Red agent starts from location (7, 8) (Figure 7.34). Blue agent moves north towards its destination and Red agent moves north-west and then Blue agent sees the Red agent as it can only see one cell around itself, just like the visual sensor of Blue agent in the previous experiment. The starting locations and the situation after first moves of both agents are shown in Figure 7.34.



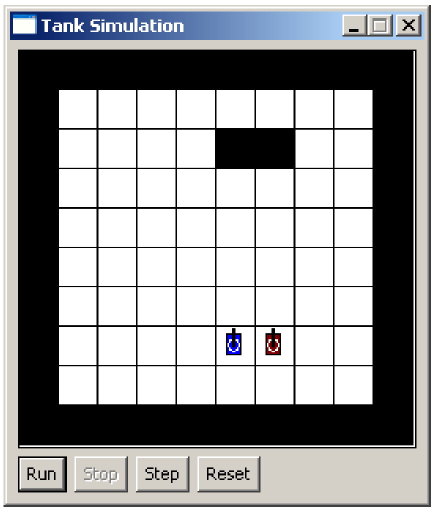


Figure 7.34 Situations before and after first moves of both Red and Blue agents

From the situation displayed in the right part of Figure 7.34 the Blue agent moves to its destination in the north and avoids collision on the way. On every step it mentally simulates its own next move while keeping in view the possible moves of the Red agent if the Red agent is visible and the main task of reaching the destination in the north across the two-cell obstacle. After every mental simulation it stores the chunk so that next time when it faces the same situation the agent does not have to mentally simulate candidate courses of action but behave as Level 1 *RPD* agent and decide straight away which action is most suitable in this situation. For example, the chunk shown in Figure 7.35 is for a situation displayed in Figure 7.36. The destination of the Blue agent is marked with a green square just above the two-cell obstacle which is given as the desired state for the task and can be seen as one of the conditions of the learnt chunk. The Blue agent is at location (3, 6) and sees Red agent in its neighbouring cell towards north-east. Blue agent is to choose one from two actions: first action moves the agent to its west cell; and the second action moves the agent to the north-west cell. The Blue agent in this situation has learnt to prefer to move north-west instead of west without any mental simulation due to its experience which it acquired in its earlier simulation runs. Note also that this chunk is specific to this configuration of agent location and is not transferable to similar configuration elsewhere.

```
sp {chunk-362*d20*tie*6
   :chunk
   (state <s1> ^name rpdsoar-ms1 ^desired <d1> ^bluetank <b1>
        ^operator <o1> + ^operator <o2> +
          ^problem-space <p1> ^map <m1>)
   (<d1> ^better higher ^bluetank <b2>)
   (<b1> ^y 6 ^x 3)
   (<b2> ^y 1 ^x 5)
   (<o1> ^name move-redtank-present ^actions <a2>)
   (<o2> ^name move-redtank-present ^actions <a1>)
   (<p1> ^name rpdsoar-ms1)
   (<m1> ^cell <c1>)
   (<c1> ^y 6 ^x 3 ^north-east <n1>)
   (<n1> ^content redtank)
   (<a1> ^move <m2>)
   (<m2> ^direction west)
   (<a2> ^move <m3>)
   (<m3> ^direction north-west)
   -->
   (<s1> ^operator <o2> < <o1>)
}
```

Figure 7.35 Chunk learnt to avoid collision with Red agent

These agents learn another type of chunk which records the success value of an action that is evaluated in mental simulation such that when next time this situation arises and an action is required to be evaluated then the success value is given straight away without further mental simulation. An example of this type of chunk is shown in Figure 7.37.

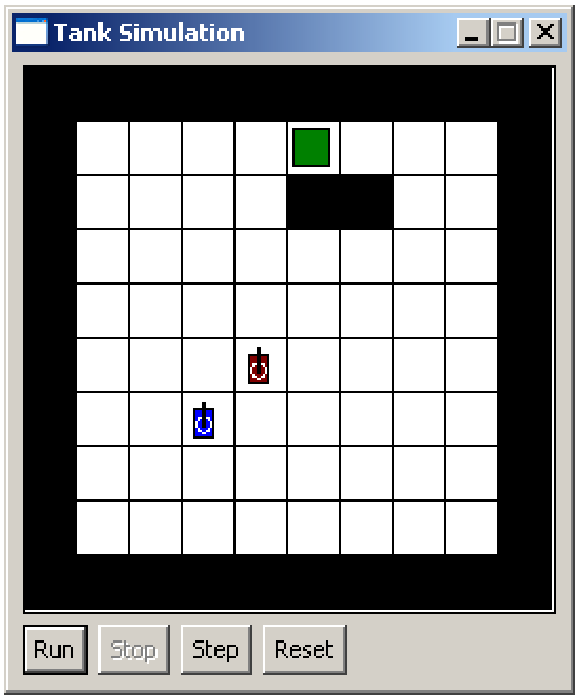


Figure 7.36 Situation for the chunk learnt to avoid collision with Red agent

```
sp {chunk-313*d227*opnochange*1
   :chunk
   (state <s1> ^operator <o1> ^evaluation <e1>)
   (<o1> -^default-desired-copy yes ^name evaluate-operator
^superproblem-space <s2> ^superoperator <s3> ^evaluation <e1> ^super-state <s4>)
   (<s2> ^name rpdsoar-ms1)
   (<s3> ^name move-redtank-not-present ^actions <a1>)
   (<s4> ^name rpdsoar-ms1 ^bluetank <b1>)
   (<b1> ^y 2 ^x 4)
   (<e1> ^desired <d1>)
   (<d1> ^bluetank <b2>)
   (<b2> ^y 1 ^x 5)
   (<a1> ^move <m1>)
   (<m1> ^direction north-east)
   -->
   (<e1> ^numeric-value 2 +)
}
```

Figure 7.37 Chunk learnt to remember success value of an action for a situation

For the first task Blue agent learns for 50 simulation runs. The performance of Blue agent over these fifty simulation runs is shown in Figure 7.38. The Blue agent consumes 253 *Soar* decision cycles in the first simulation run and learns 344 chunks.
It keeps learning chunks for first five simulation runs and then it does not encounter any new situation for up to seventh simulation run, during this time it uses its learnt knowledge and behaves like an experienced Level 1 *RPD* agent for these situations. Then again it finds new situations and has to mentally simulate the applicable actions to evaluate them and then select one which takes more *Soar* decision cycles. Up to thirty simulation runs the agent faces situations within a run that are new and keeps learning and for the rest of the simulations after the thirtieth it uses its learnt knowledge. By the time it reaches fiftieth simulation run it performs the same task in

14 *Soar* decision cycles, and behaves like a Level 1 *RPD* agent for this task. And by this time it is not learning any more chunks.

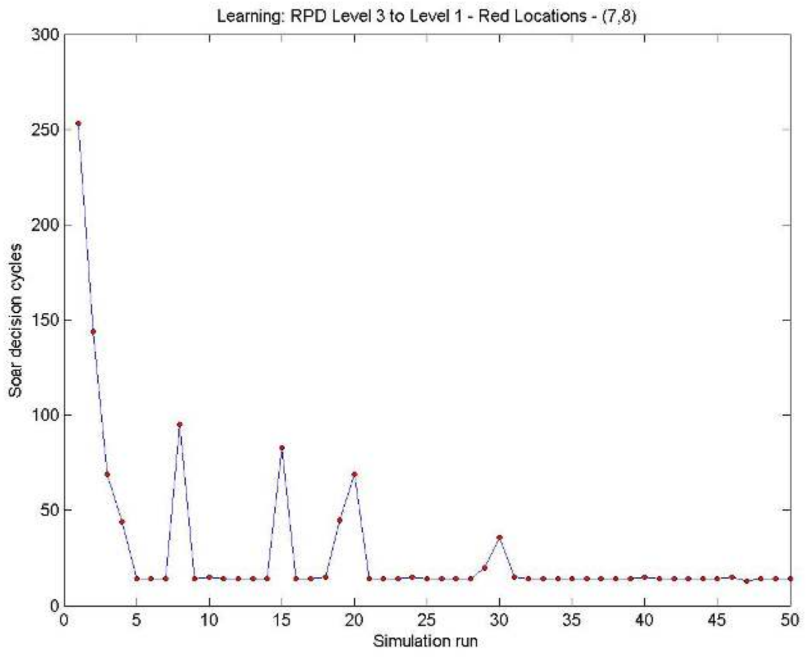


Figure 7.38 Learning curve of Blue agent – Red agent starts from location (7, 8)

7.5.5 Second task – Red agent starting position - (8, 8)

The Blue agent starts from location (5, 8) and Red agent starts from location (8, 8) (Figure 7.39). Blue agent moves north twice towards its destination and Red agent moves north-west twice and then Blue agent sees the Red agent. The starting locations and the situation after first two moves of both agents are shown in Figure 7.39.

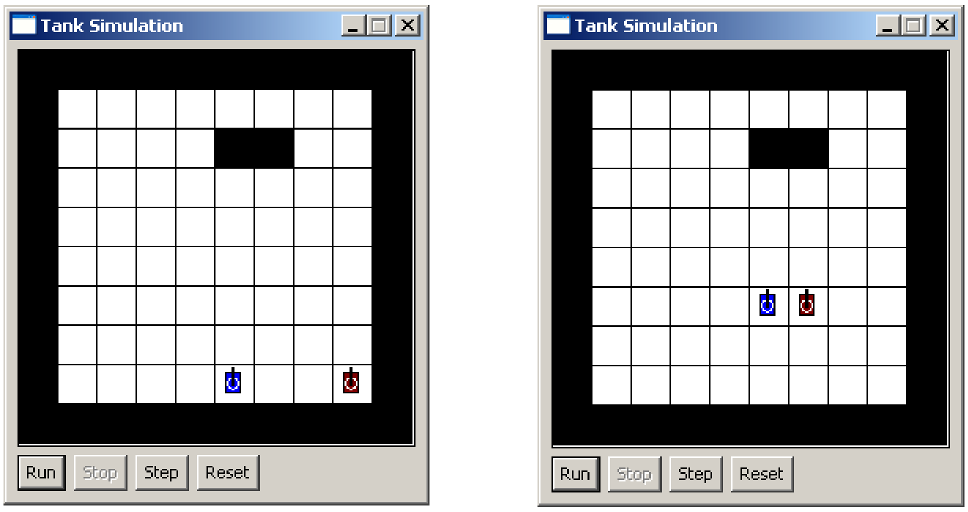


Figure 7.39 Situations before and after two moves of both Red and Blue agents

From the situation displayed in the right part of Figure 7.39, the Blue agent moves to its destination in the north and avoids collision on the way. On every step it mentally simulates its own next move while keeping in view the possible moves of Red agent if the Red agent is visible and the main task of reaching the destination in the north across the two-cell obstacle. After every mental simulation it stores the chunks that it learns.

Blue agent learns for 50 simulation runs for the second task also. The performance of Blue agent over these fifty simulation runs is shown in Figure 7.40. The Blue agent consumes 262 *Soar* decision cycles in the first simulation run and learns 369 chunks. It keeps learning chunks for first seven simulation runs and then it does not encounter much of new situations for up to thirteenth simulation run, during this time it uses its learnt knowledge and behaves mostly like an experienced Level 1 *RPD* agent for the situations faced. It keeps learning uptil 45th simulation run most of which is done up to the 28th simulation run. By the time it reaches fiftieth simulation run it performs the same task in 14 *Soar* decision cycles, and behaves like a Level 1 *RPD* agent for this task. And for the last few simulation runs it does not learn new chunks. There is one spike in between 25th and 30th simulation runs, it is because the agent selects a new path to the destination and gets into more number of new situations. Because of the inherent variability in the behaviour of the agent it may happen during any simulation run but it is observed that this agent for this task learns most of the chunks within fifty simulation runs.

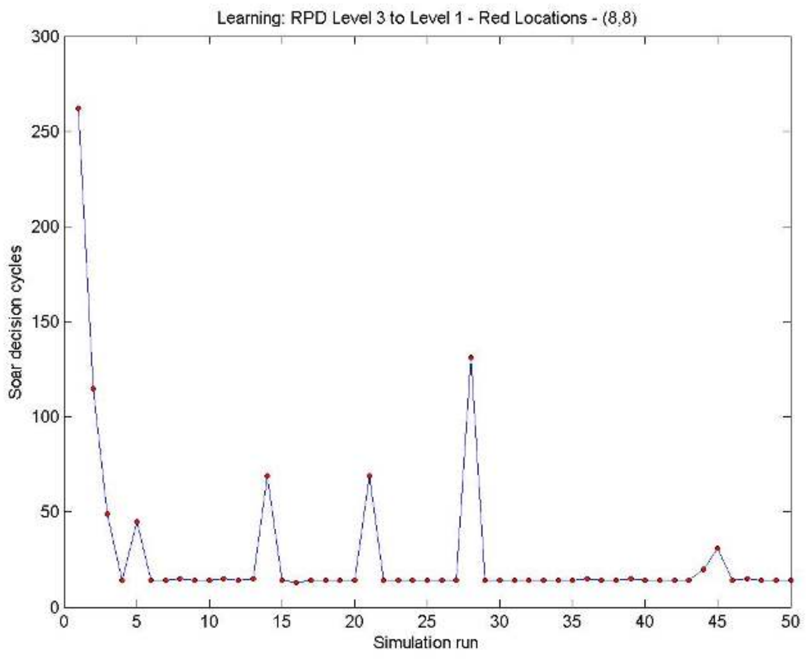


Figure 7.40 Learning curve of Blue agent – Red agent starts from location (8, 8)

7.5.6 Third task – Red agent starting position - (9, 7)

The Blue agent starts from location (5, 8) and Red agent starts from location (9, 7). Blue agent moves north towards its destination and Red agent moves north-west and then after three moves from both agents, Blue agent sees the Red agent in the cell to its north-east (Figure 7.41).

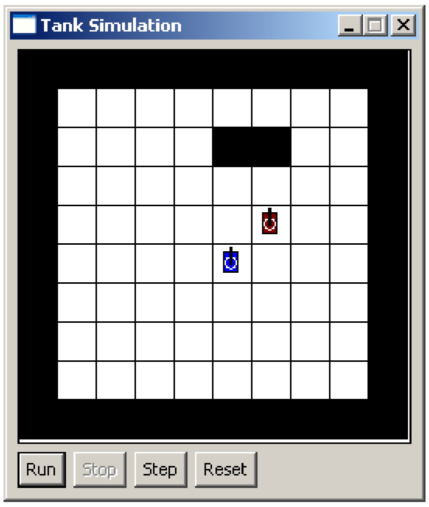


Figure 7.41 Situations after three moves of both Red and Blue agents

From the situation displayed in Figure 7.41, the Blue agent moves to its destination in the north and avoids collision on the way. On every step it mentally simulates its own next move while keeping in view the possible moves of Red agent if the Red agent is visible and the main task of reaching the destination in the north across the two-cell obstacle. After every mental simulation it stores the chunks that it learns.

Similar to the first two tasks, Blue agent learns for 50 simulation runs for the third task also. The performance of Blue agent over these fifty simulation runs is shown in Figure 7.42. The Blue agent consumes 238 *Soar* decision cycles in the first simulation run and learns 331 chunks.

It keeps learning chunks for first forty simulation runs and then it does not encounter new situations for up to the end that is the fiftieth simulation run except the 45th simulation run where the agent faces new situations. By the time it reaches fiftieth simulation run it performs the same task in 13 *Soar* decision cycles, has learnt 923 chunks and behaves like a Level 1 *RPD* agent for this task.

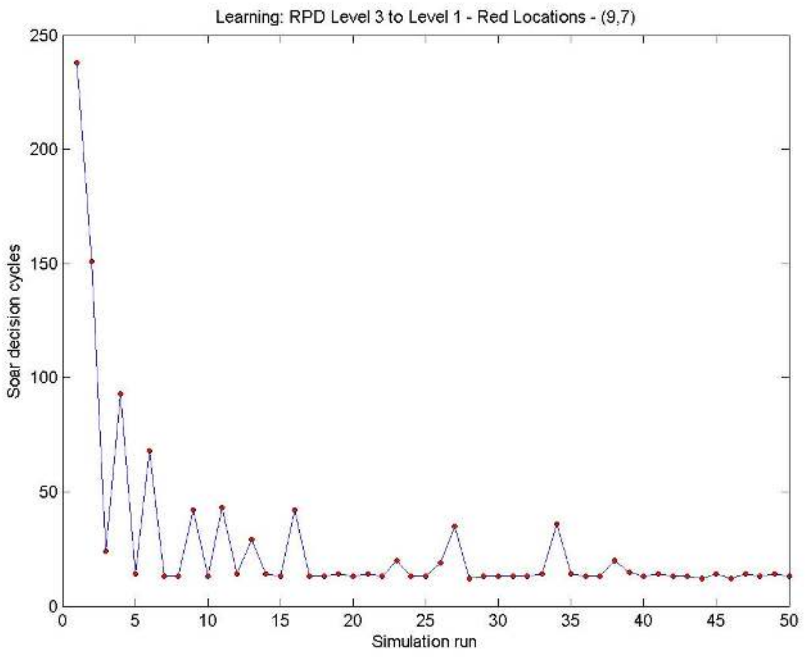


Figure 7.42 Learning curve of Blue agent – Red agent starts from location (9, 7)

7.5.7 Fourth task – Red agent starting position - (9, 8)

The Blue agent starts from location (5, 8) and Red agent starts from location (9, 8). The Blue agent moves north towards its destination and Red agent moves north-west and then after three moves by both agents, Blue agent sees the Red agent in its east (Figure 7.43).

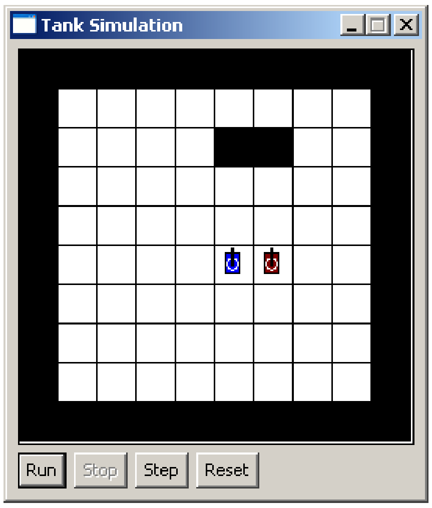


Figure 7.43 Situations after three moves of both Red and Blue agents

From the situation displayed in Figure 7.43, the Blue agent moves to its destination in the north and avoids collision on the way. On every step it mentally simulates its own next move while keeping in view the possible moves of Red agent if the Red agent is visible and the main task of reaching the destination in the north across the two-cell obstacle. After every mental simulation it stores the chunks that it learns.

Similar to the last three tasks, Blue agent learns for 50 simulation runs for the fourth task also. The performance of Blue agent over these fifty simulation runs is shown in Figure 7.44. The Blue agent consumes 285 *Soar* decision cycles in the first simulation run and learns 405 chunks.

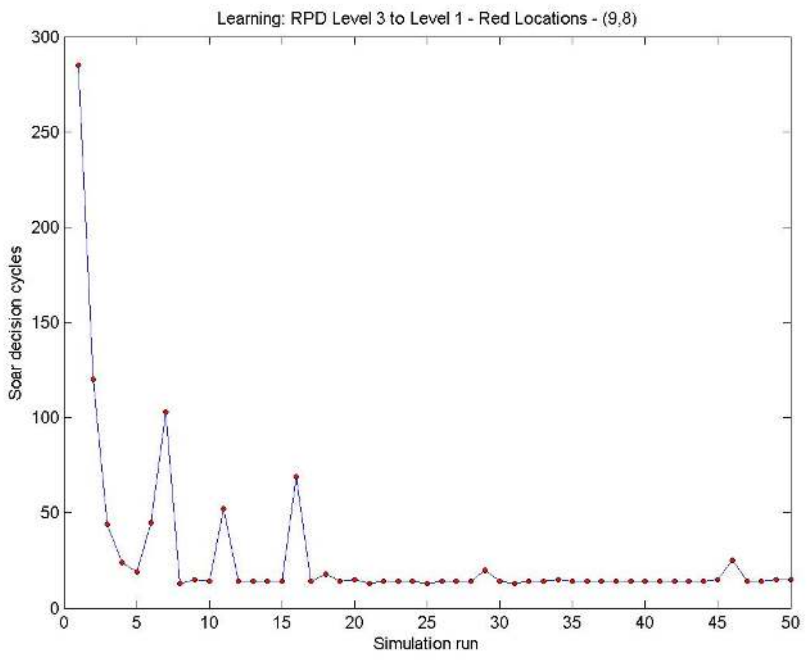


Figure 7.44 Learning curve of Blue agent – Red agent starts from location (9, 8)

It learns most of the chunks within first twenty simulation runs and then it does not encounter much of new situations for up to the end that is the fiftieth simulation run, except for few new situations just before thirtieth and after forty-fifth simulation run. By the time it reaches fiftieth simulation run it performs the same task in 15 *Soar* decision cycles, has learnt 905 chunks and behaves like a Level 1 *RPD* agent for this task.

The results of experiments on learning discussed in the Sections 7.5.4 – 7.5.7 demonstrate the ability of the agent to learn from its experience. Maximum learning occurs in the initial runs of the simulation for each of the four tasks given to the agent. This learnt knowledge of the agent is used on exactly the same task. However, if there are certain situations that occur in other tasks then the learnt knowledge from one task may be utilized in other tasks also. The next experiment tests the ability of the agent to transfer learnt knowledge from one task to another with overlapping problem spaces.

7.5.8 Transfer of learning

This experiment is aimed at testing an *RPD-Soar* agent for learning that is transferrable from one task to the other during the life of an agent. It is observed in the previous tasks that the agent learns all the chunks for the task in approximately 50 simulation runs as almost all new situations that may arise are encountered by the agent within these exposures. Therefore, this experiment is based on 200 simulation runs, fifty for each of the four tasks discussed in the Sections 7.5.4 – 7.5.7. These tasks are appropriate to test for evidence of transfer of learnt knowledge because the problem spaces of the tasks overlap, and similar situations are likely to arise across tasks. However, the quantity of transferred knowledge varies from one task to the other due to inherent variability in behaviour of the agents. In this experiment, the agent learns from one task in fifty simulation runs, holds the learnt chunks and then the task is changed for the next fifty simulation runs, and so on until all four tasks are performed by the agent for fifty times each. It is assumed, based on the results of the last four experiments, that fifty simulations for one task enable the agent to learn most of the chunks that can be learnt for this type of tasks. The learning performance of the *RPD-Soar* agent is displayed in Figure 7.45.

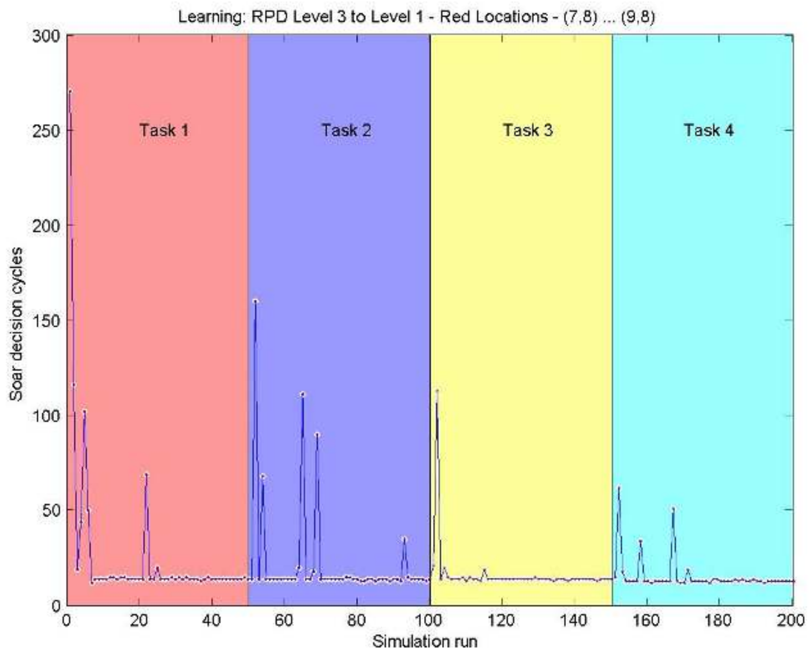


Figure 7.45 Learning curve of Blue agent over four tasks

The agent starts performing the first task exactly in the same way as it performed in the first experiment related to learning for the first task discussed above. It consumes 265 *Soar* decision cycles but by the time it completes thirty simulation runs it is consuming approximately 14 to 15 *Soar* decision cycles. After fifty simulation runs the task of the agent is changed to the second task. Now in performing this new task the agent consumes approximately 100 *Soar* decision cycles less than it consumed in the second experiment related to learning discussed in Section 7.5.5. After 100 simulation runs the task is changed again and the agent performs the third task in the first simulation run for the new task in only 112 decision cycles, which is 126 Soar decision cycles less than it consumed in third experiment related to learning discussed in Section 7.5.6. At 150th simulation run the task is changed again and this time in the first simulation run of the new task agent completes the task in only 60 *Soar* decision cycles. This performance graph displays clear evidence of transfer of knowledge from one task to the other. The total number of chunks learnt in performing each of the four tasks separately in the experiments discussed above is shown in Table 7.4. The total number of learnt chunks are reduced from 3807 to only 1703 required to perform the same four tasks by the same agent due to transfer of learning.

The transfer of learning is largely attributable to the chunks that can be used in the similar situations in other tasks. The agent learns from the results of mental simulation which is the knowledge to select the action in a situation which offers the best success value. During mental simulation all candidate actions are evaluated one after the other and the results are stored as chunks.

A file containing all the chunks learnt by the *RPD-Soar* agent in this experiment is available in the attached CD, see Appendix E.

Table 7.4 Comparison of chunks learnt

Tasks	Number of chunks learnt	
	Independent tasks	Tasks in-sequence
1 – Red (7, 8)	1021	-
2 – Red (8, 8)	958	-
3 – Red (9, 7)	923	-
4 – Red (9, 8)	905	-
Total	3807	1703

*This number is expected to remain approximately same for any ordering of these tasks.

7.6 Experiment 5 – Recognition of situation by neural network

The vignette for this experiment is motivated from the work of Liang et al. (2001). The domain is a military ground based operation. A military commander of a troop of tanks consisting of three tanks selects a strategy to attack the enemy tank in the north. The terrain is simple and it can have 0, 1, or 2 passable hills. The terrain with two hills is shown in Figure 7.46; the enemy is represented with a red square in the north at location (0, 1) and own position is the blue circle in the south at the origin. The locations are represented in Cartesian coordinates, the abscissa ranges [1, -1] and ordinate [0, 1]. The agent's own starting position and the enemy position remains the same through out the experiment. The enemy is static and fights from the same location until the battle is over. The commander selects a strategy based on the decisions that whether to divide the troop of tanks in an assault group (*AG*) and a fire support (*FS*) group or to use them as one group only. The commander also selects the intermediate and the final locations or location of these groups or group which also dictate the route to be adopted by the group(s).

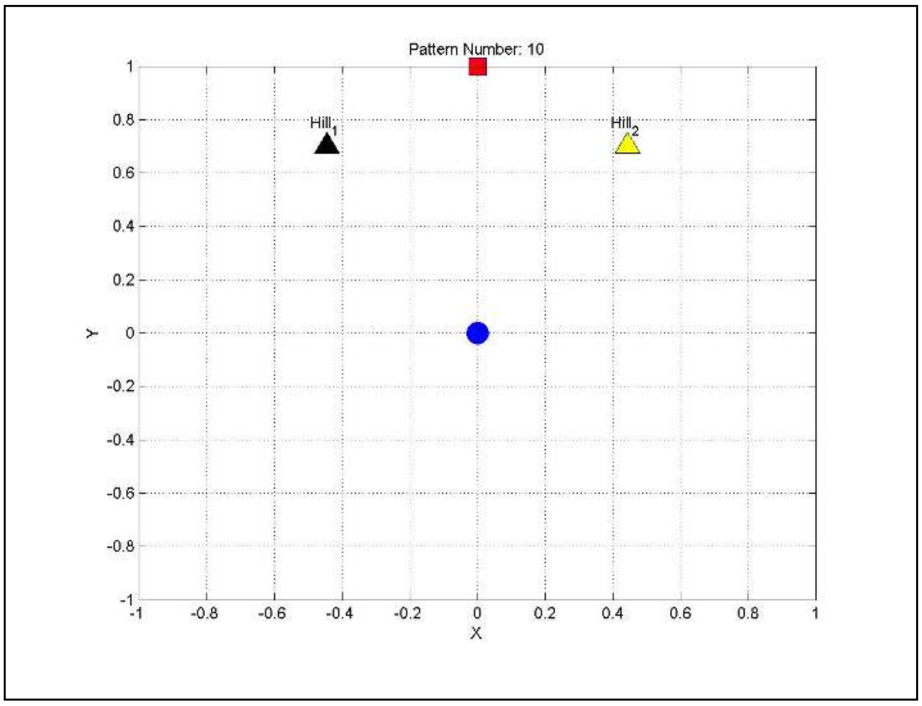


Figure 7.46 Example terrain with two hills

As discussed in Chapter 6, the neural net in this experiment is used for pattern recognition and not for plan generation because, as Liang et al. also realize, the option to generate plan directly from the trained neural network does not prove to be successful. In this experiment the target for training in each case is the numeric value of '1' for the output node corresponding to the recognized situation and '0' for the rest of the output nodes. It is assumed that such a clear difference between two target values will produce better results compared to the mixed target values for the output nodes corresponding to different strategies in the work of Liang et al. (2001). Moreover, there is potential advantage in this representation for an *RPD* model. The advantage in this design is that for a given situation the output node with the highest value is considered as the recognized situation and if the evaluation of the corresponding course of action through mental simulation is not promising then the output node with the second highest value may be considered. During training of the network, it is observed by the author and also mentioned by Liang et al. (2001) that training the same neural network for two different plans for the same situation or minor changes in the situation such as one for an aggressive and the other for a conventional commander reduces the learning performance in terms of increased residual error. Therefore, the training set is divided into two parts one for the aggressive commander and the other for the conventional commander. The basic situations and corresponding strategies in the work of Liang et al. (2001) is used but some strategies are modified and some more strategies are added in the training set for this experiment based on the knowledge of the author on the subject. The reason for the addition of examples in the training set is to improve the performance of the net in recognition of new situations which are related to number of training examples and also to sufficiently cover the problem space.

One training example that is modified is shown in the Figure 7.47. In this example the final location of *AG* is almost in the line of fire of the *FS* group, this strategy based on making maximum use of the cover from enemy observation and fire available to own tanks due to the hill may result in fratricide.

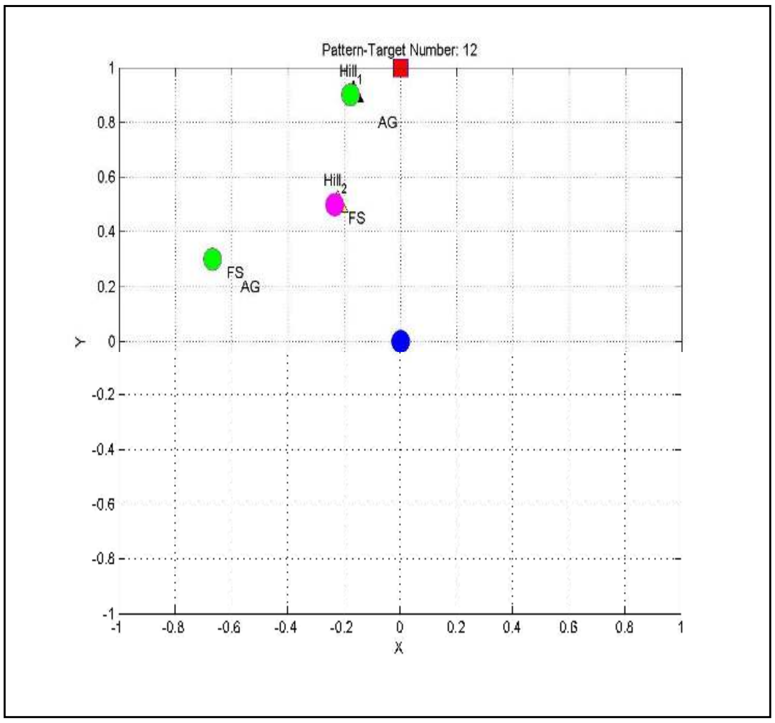


Figure 7.47 A typical training example. Note the placement of the assault group (*AG*) and the fire support group (*FS*), the *FS* which is supposed to support the *AG* with fire during the attack is behind the *AG* and almost in the same line. This strategy makes maximum use of cover from fire and observation available to own tanks from the enemy due to hills but this placement can result in fratricide and is unrealistic and needs to be modified.

Now consider the example in Figure 7.48. Again, in this example the cover from observation and fire available in the form of the hills is used but the distance of the fire support group from the enemy is relatively more compared to the other training examples. Although, the scale of the map and the firing range of the weapon systems have not been explicitly given by the author, the general idea about the reasonable distance of the fire support group from the target may be established keeping in view the rest of the plans in the training set. In this case it is relatively more close to the own position than the enemy and therefore this plan is modified.

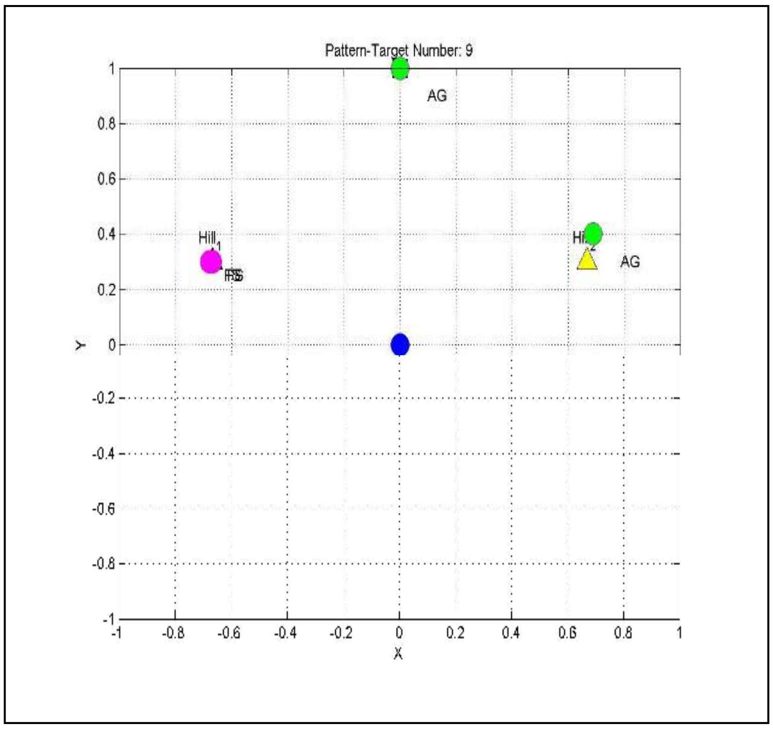


Figure 7.48 Another training example that needs modification. In this strategy also in order to make maximum use of hills to protect own tanks from enemy observation and fire, the *FS* is positioned relatively more close to the own initial position than the enemy positions being attacked by the *AG*. The *FS* should be positioned closer to the enemy to provide effective fire support.

7.6.1 Training examples

The locations of the hills 1 and 2 and the corresponding situation for training the neural net to represent the conventional commander is given in Table 7.5.

Table 7.5 Training set for conventional commander

Hill 1		Hill 2		Situation
Y	X	Y	X	
0.00	0.00	0.00	0.00	1
0.30	-0.67	0.30	0.67	1
0.60	0.00	0.00	0.00	2
1.00	-0.17	0.00	0.00	3
0.70	-0.11	0.00	0.00	4
1.00	-0.17	0.40	0.00	5
0.90	-0.44	0.70	0.00	6
0.50	0.00	0.00	0.00	7
0.90	-0.17	0.90	0.17	8
0.90	-0.17	0.50	-0.22	9
0.70	-0.44	0.70	0.44	10
0.60	0.00	0.70	0.17	11
0.60	-0.67	0.60	0.67	12

For every situation there is a corresponding strategy and these strategies are shown in Figure 7.49 through to Figure 7.60. It is worth mentioning here that the first two terrain patterns given in the first two rows of Table 7.5 relate to situation 1 and the corresponding plan is shown in Figure 7.49, and from then onwards each row representing a single terrain pattern has a distinct plan.

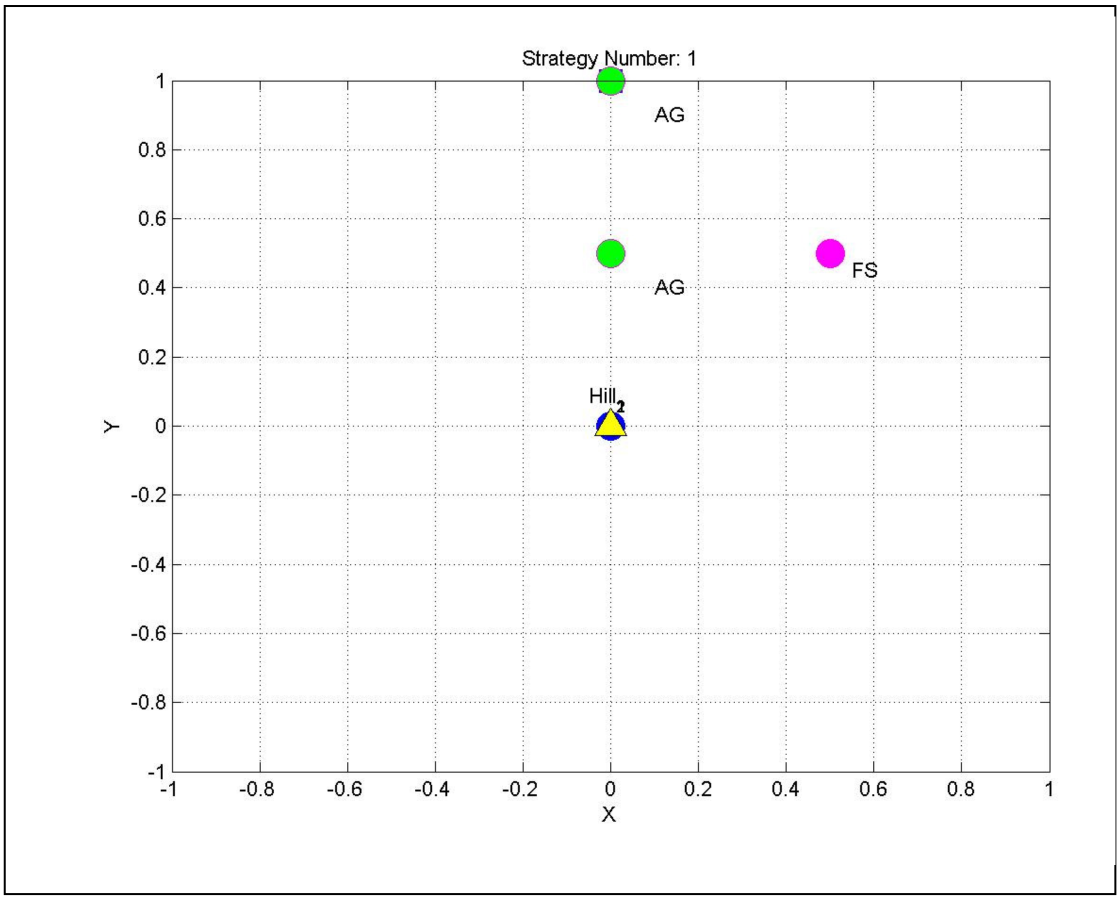


Figure 7.49 Strategy for Situation 1. For all those situations where either there are no hills present in the battlefield or the hills are located closer to the own position than the enemy (first two entries of Table 7.5 correspond to such situations) this strategy is used. This plan is conventional in which the own troop of tanks is divided into two groups the *FS* and the *AG*, the *FS* is positioned on the east to provide fire support while the *AG* attacks from the south.

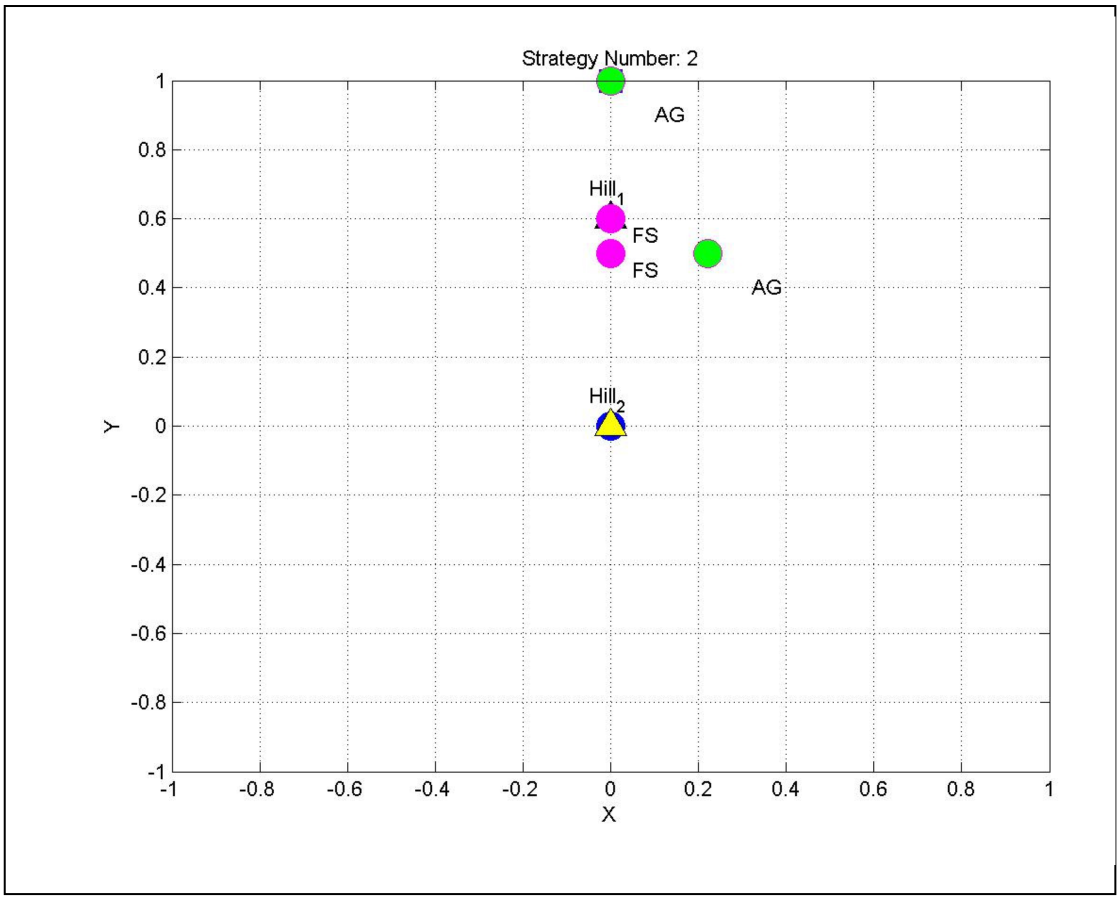


Figure 7.50 Strategy for Situation 2. In this battlefield there is only one hill in the middle ground that affects the selection of strategy. The *FS* is positioned behind the hill to protect it from enemy observation and fire. The *AG* manoeuvres from the east and attacks the enemy positions.

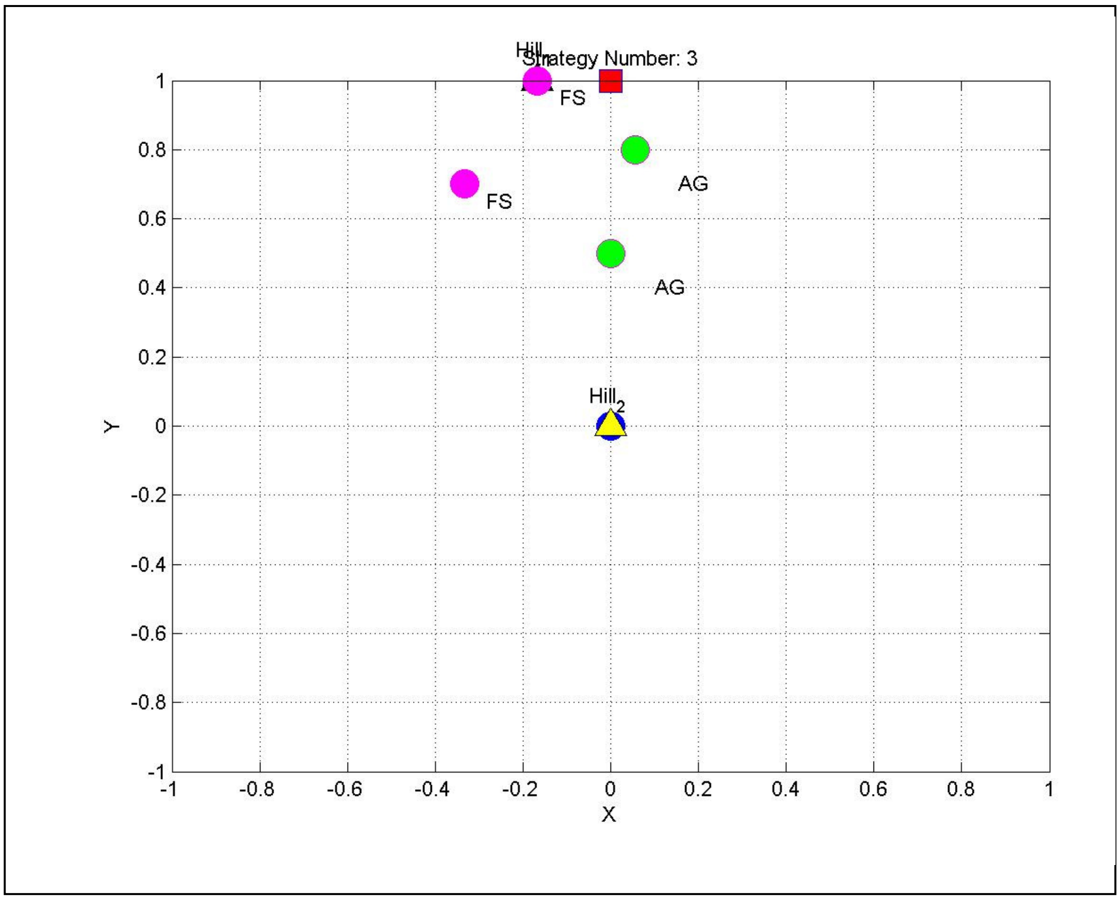


Figure 7.51 Strategy for Situation 3. In this battlefield there is only one hill located in the west and very close to the enemy position. The *FS* takes position behind this hill to provide fire support and the *AG* attacks the enemy from the south. The position of the fire support is very close to the enemy and can provide very effective fire support. Although it is protected behind the hill but due to proximity to the enemy *FS* group is threatened and this strategy is based on a calculated risk as regards *FS* group.

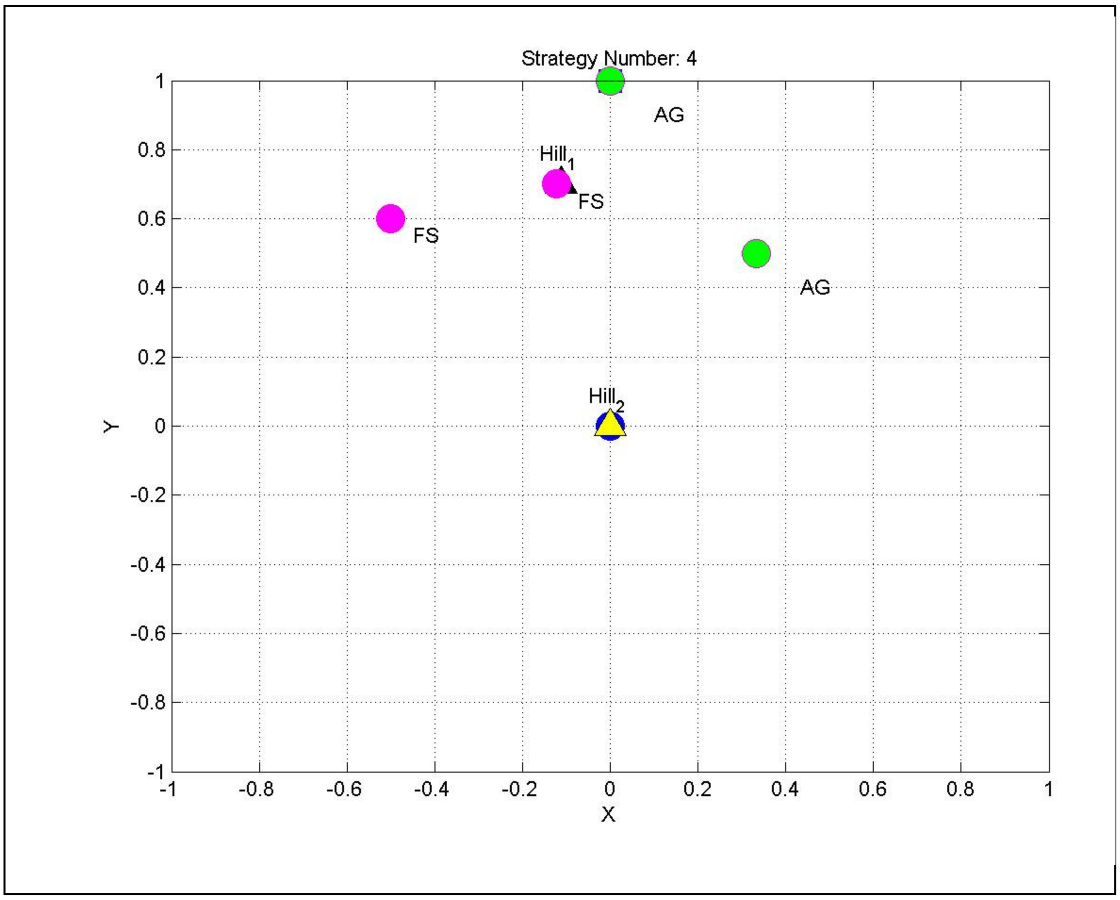


Figure 7.52 Strategy for Situation 4. In this battlefield there is only one hill located in the south west of the enemy position. The *FS* group occupies the position behind this hill moving to its position from the west. The *AG* manoeuvring from the east attacks the enemy position.

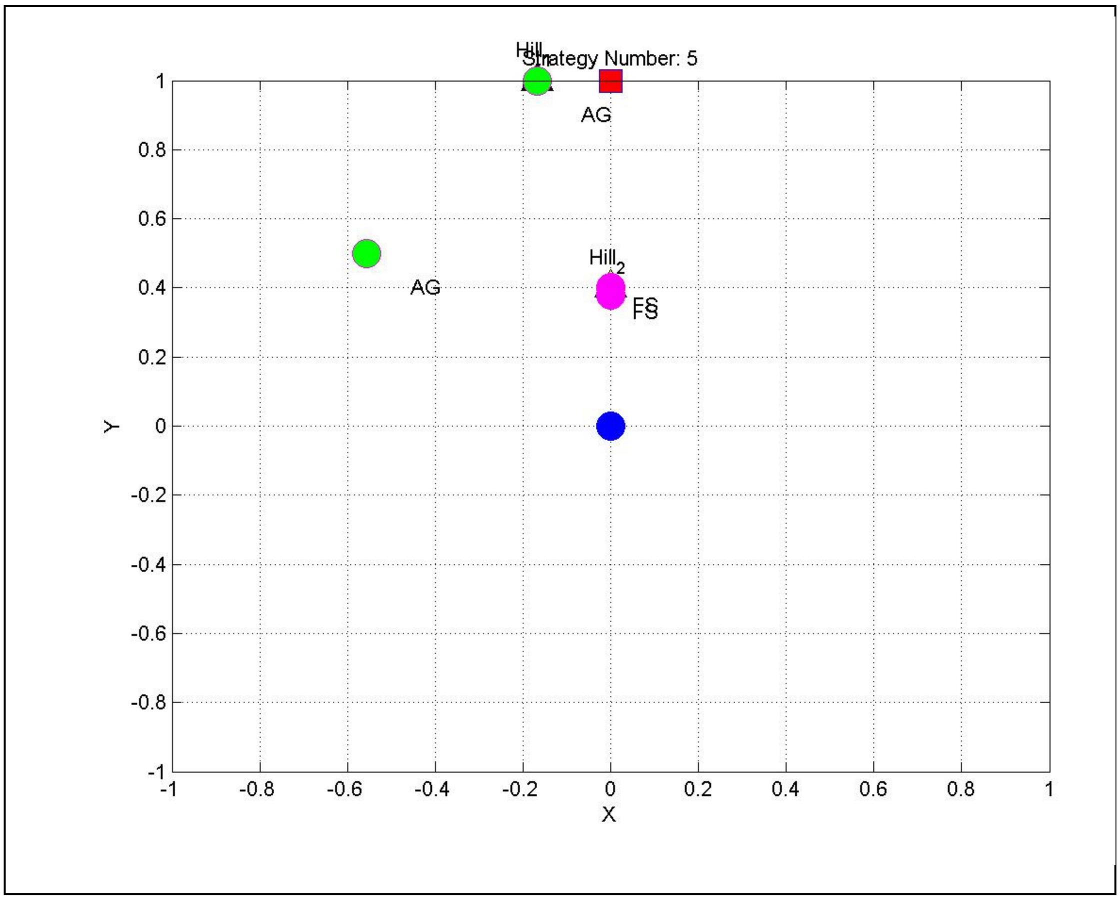


Figure 7.53 Strategy for Situation 5. In this battlefield there are two hills: one hill is located just short of the middle ground; and the other is located a little west of the enemy position. The *FS* group moves north to occupy its position south of the hill in the middle ground from where it supports the attack. The *AG* manoeuvres from the west and attacks the enemy position from behind the hill located in the west of the enemy.

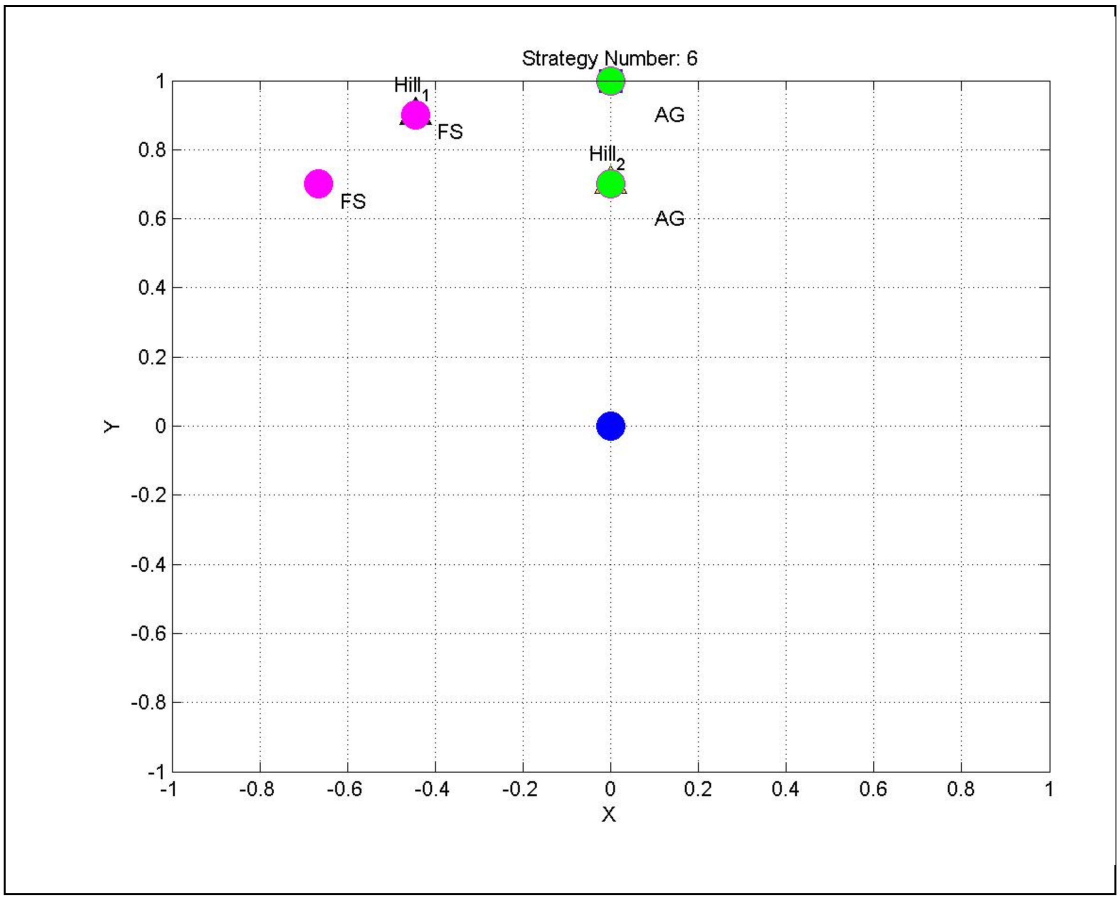


Figure 7.54 Strategy for Situation 6. There are two hills in this battlefield; one hill is located close to enemy position on its south and the other hill is located south-west-west of the enemy position. The *FS* group moves from the west and occupies its position behind the west hill to provide fire support for the *AG*. The *AG* moves north to the hill south of the enemy and attacks the enemy position from there.

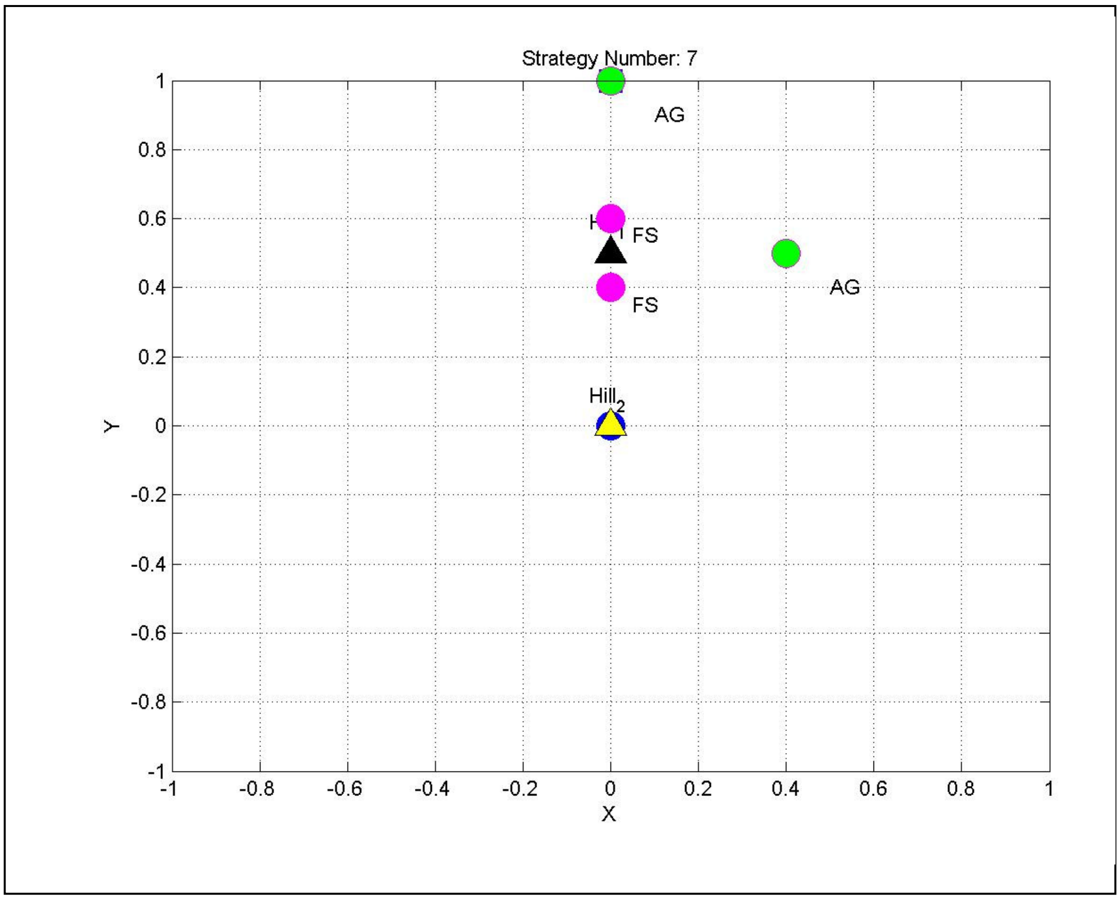


Figure 7.55 Strategy for Situation 7. There is only one hill in this battlefield located in the middle ground south of the enemy position. The *FS* group moves north of the hill to support the *AG*. The *AG* manoeuvres from the east to attack the enemy position. This situation is quite similar to the situation in Figure 7.50 with the only difference that the hill in this case is comparatively more towards south of the enemy position.

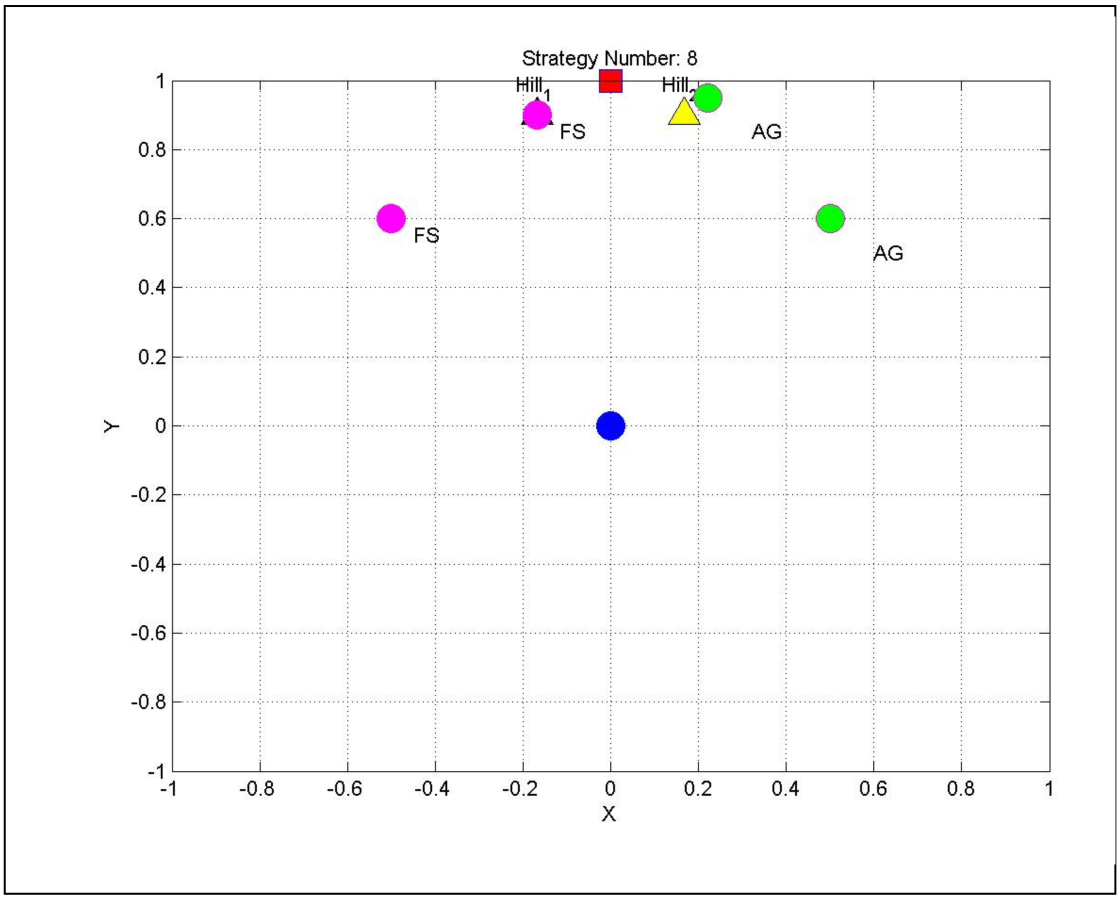


Figure 7.56 Strategy for Situation 8. There are two hills in this battlefield. Both of the hills are close to the enemy position; one on the south-east and the other on the south-west. The *FS* group moves from the west and occupies its position behind the south-west hill to provide fire support to the *AG*. The *AG* manoeuvres from the east to attack the enemy position from the south-east hill.

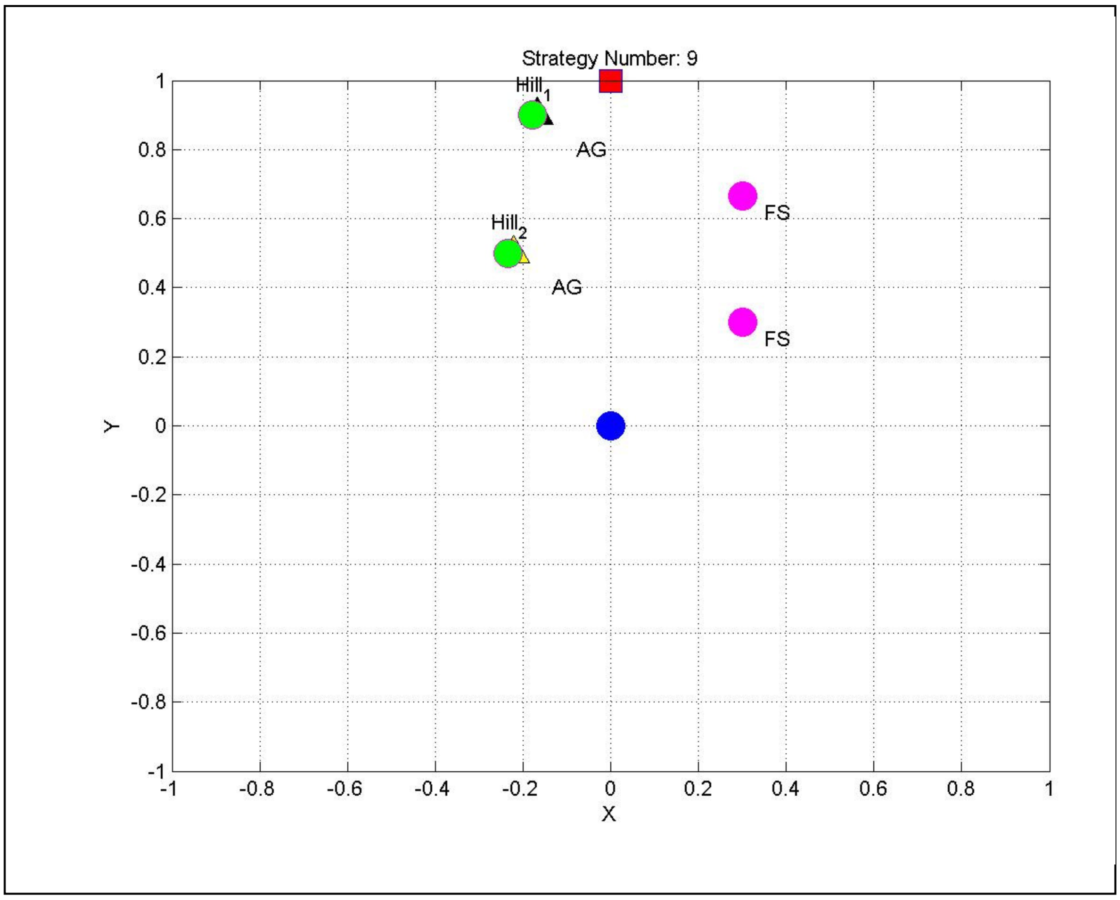


Figure 7.57 Strategy for Situation 9. There are two hills in this battlefield and both of them are south-west of the enemy position. The *AG* manoeuvres from the west taking cover of these hills and attacks the enemy position from behind the hill close to the enemy position. The *FS* group moves from the east and takes position in the open terrain in the south-east of the enemy position to provide fire support to the *AG*.

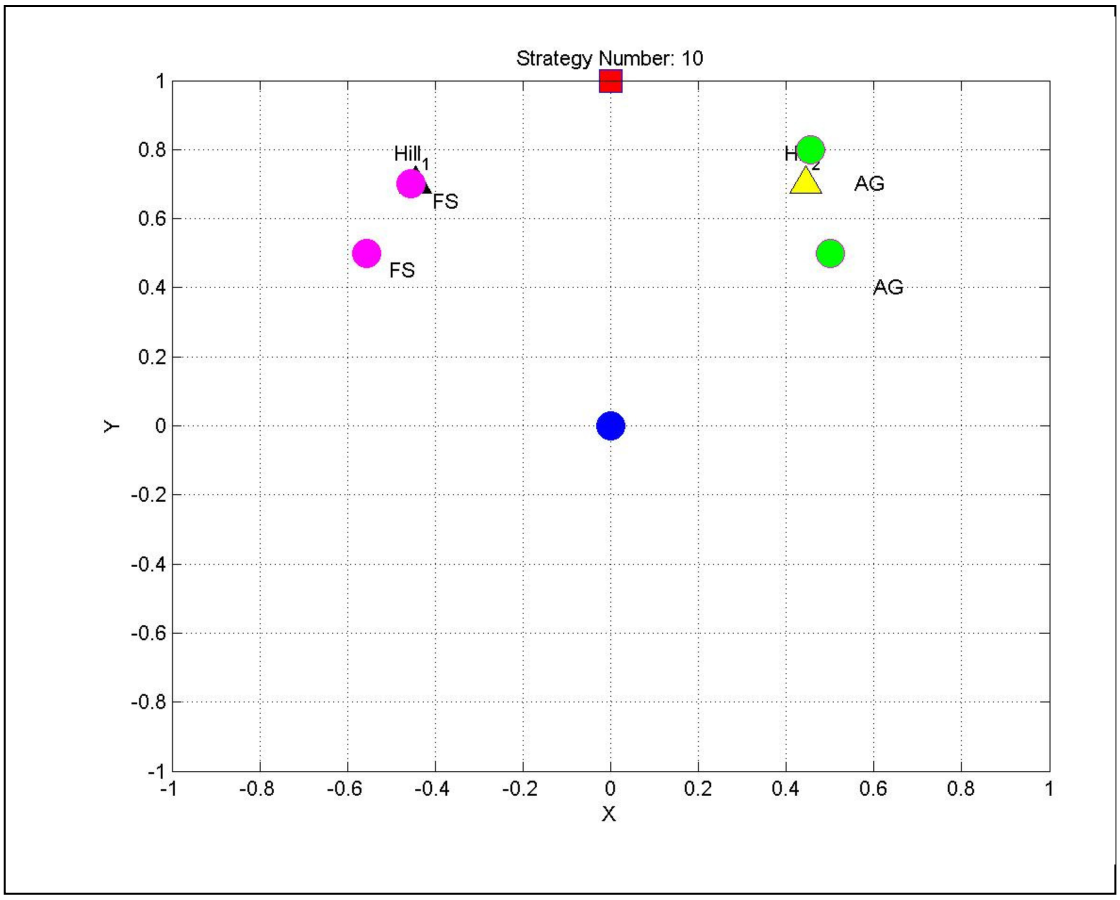


Figure 7.58 Strategy for Situation 10. It is a very idealistic battlefield for the attacker, due to two hills present at suitable locations to provide cover for both of it's groups that is the *AG* and the *FS.* The *FS* moves from the west and occupies position behind the south-west hill to support the *AG* with fire and the *AG* manoeuvres from the east to attack the enemy position from behind the south-east hill.

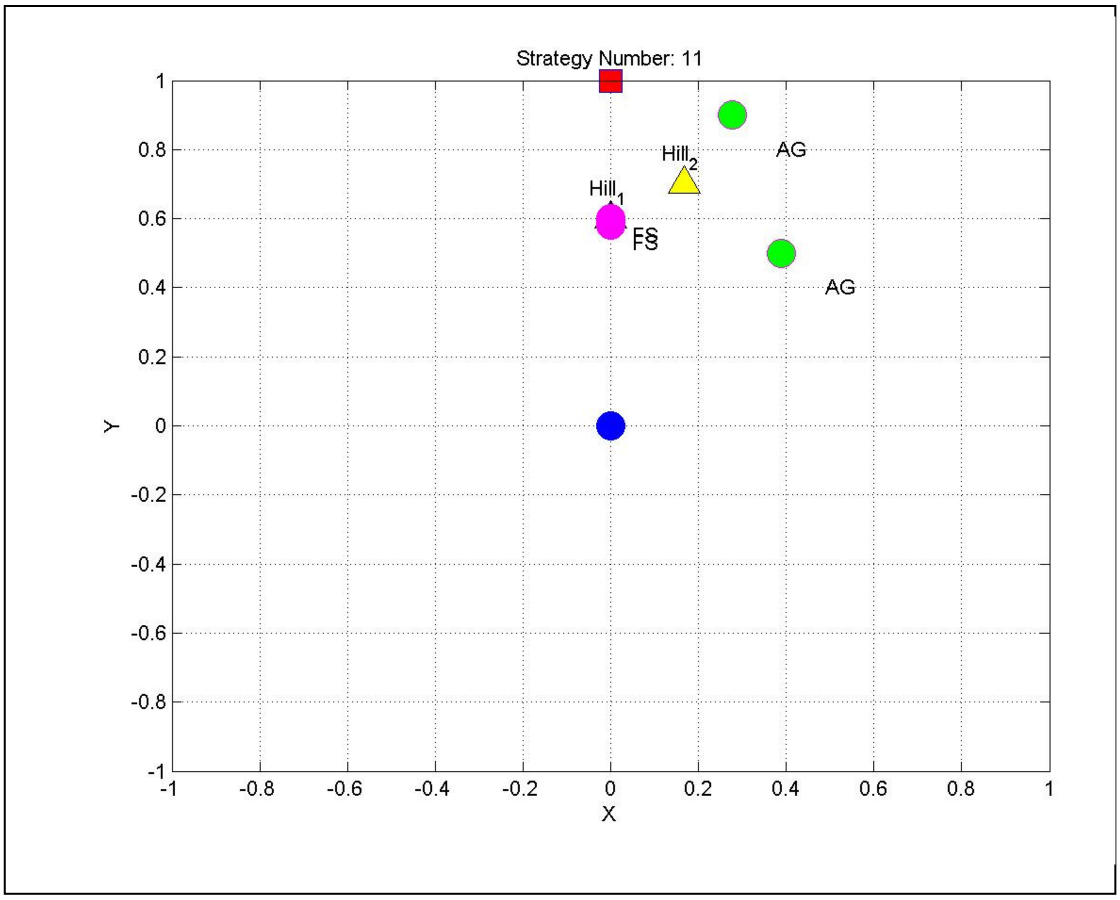


Figure 7.59 Strategy for Situation 11. There are two hills in this battlefield; one is in the south and the other is in the south-south-east of the enemy position. The *FS* group takes advantage of the hill in the south and moving north occupies the position behind the hill to support the attack of *AG* with fire. While the hill in the north-east of this south hill is relatively close and is not suitable for the *AG* to position behind it because this narrow angle from the view point of the enemy is suitable for effective engagement of both groups with fire. Therefore, the *AG* manoeuvres further east taking partial cover from the hill and attacks the enemy position from the east.

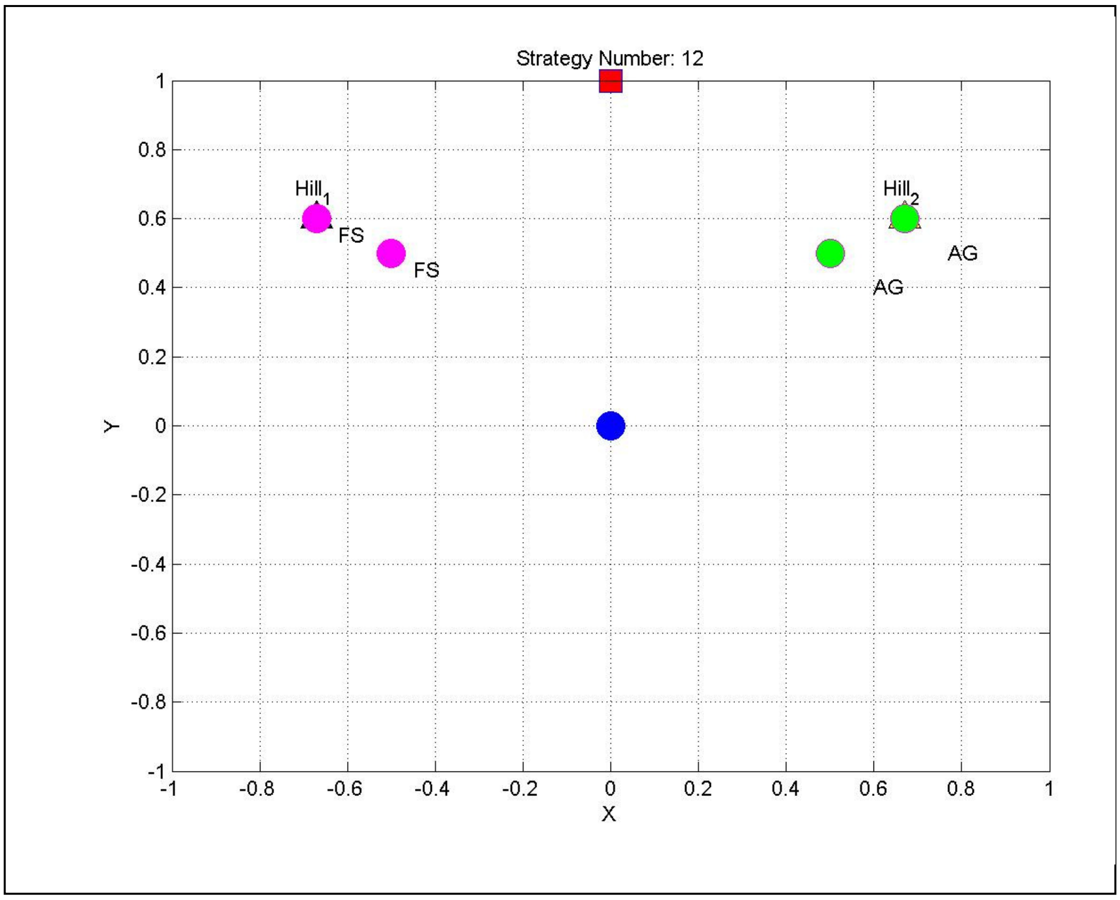


Figure 7.60 Strategy for Situation 12. There are two hills in this battle field; both in south and one each in either directions east and west. This situation resembles the situation presented in Figure 7.58 with the difference that the hills in this case are comparatively a little south and further away in easterly and westerly directions. Although the distance of the hills from the enemy position is a little more than what is ideal for positioning *FS* and *AG* for the attack but is sufficiently advantageous and therefore *FS* group positions behind the westerly hill and the *AG* attacks from behind the easterly hill in this strategy.

7.6.2 Results

In order to explore the problem space we fixed one hill and moved the other hill on the given terrain on an interval of 0.01 on both axes. The neural net part of the agent is required to recognize new situations produced in the environment. In the battlefield, the locations of enemy and own positions are fixed. The situational variables are the locations of hills in the terrain. As the locations of the hills are changed in the terrain, new situations are generated. The ability of the agent to recognize new situations need to be ascertained. The new situations are the ones that are not included in the training examples and for which the agent has not been trained. By fixing one hill at a suitable location and moving the other hill throughout the battlefield new situations are generated. The hill is fixed at such locations so that maximum problem space is explored and important new situations are produced. Plans produced for new situations by the neural net trained for the conventional commander are shown in Figure 7.61 through to Figure 7.64.

The result of experiment where one hill is fixed at (0, 0.17) is shown in Figure 7.61. The diagram shows the situation that is recognized when the second hill is in different positions. All the situations in which one of the hills is to the south of the enemy location at a middle distance are expected to be recognized. These are Situations 2, 5, 6, 7 and 11. Situation 5 is recognized when the second hill is in the area just west of the enemy location. Situation 5 should claim some of the area in its south, presently occupied by Situation 1 and the area around own position that is middle bottom should have been claimed by Situation 2. The reason for Situation 1 to be claiming these areas probably is that Situation 1 has two training examples and that might have increased its influence on recognition. Situation 12 is recognized in the area when the second hill is moved to the extreme west in the upper part of the battle field which is expected. Situation 9 is not recognized at all and it is not expected to be recognized because in Situation 9 both of the hills are in the west and that situation never occurs in this setting.

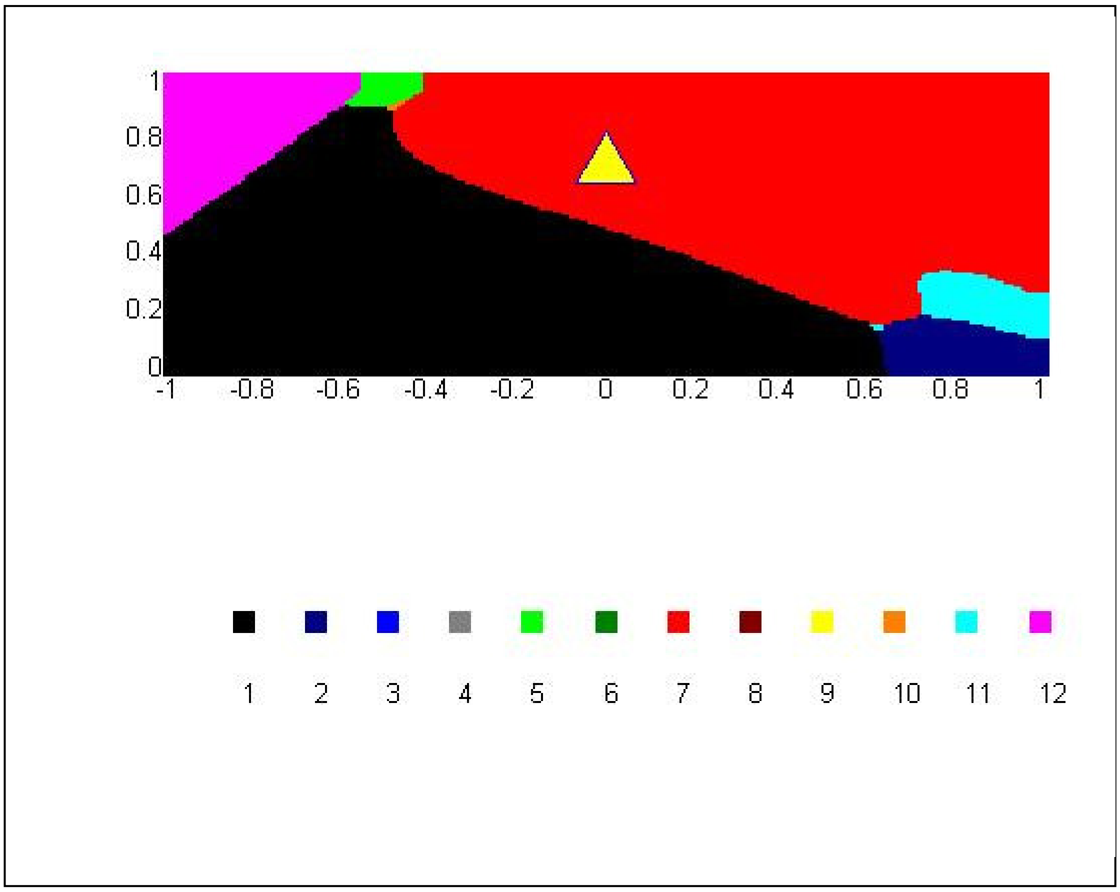


Figure 7.61 Situations with one hill fixed at (0, 0.7). In this case a total of six situations are recognized, but two situations recognized most of the time are 1 (Figure 7.49) and 7 (Figure 7.55). If the other hill is south-westerly then Situation 1 is recognized but if it is towards east then Situation 7 is recognized. The strategies applied to these two situations are similar but only the locations of *AG* and *FS* are interchanged. The other recognized situations are 2, 5, 11 and 12. One desirable feature common to all the strategies applied to these situations is the use of the hill in the middle ground as protection from observation and fire for either *AG* or *FS*.

The result of the experiment where the location of one hill is fixed at (0.4, 0.7) is shown in Figure 7.62. Because of fixing the location of one hill in the east Situations 8, 10, 11 and 12 are expected to be recognized.

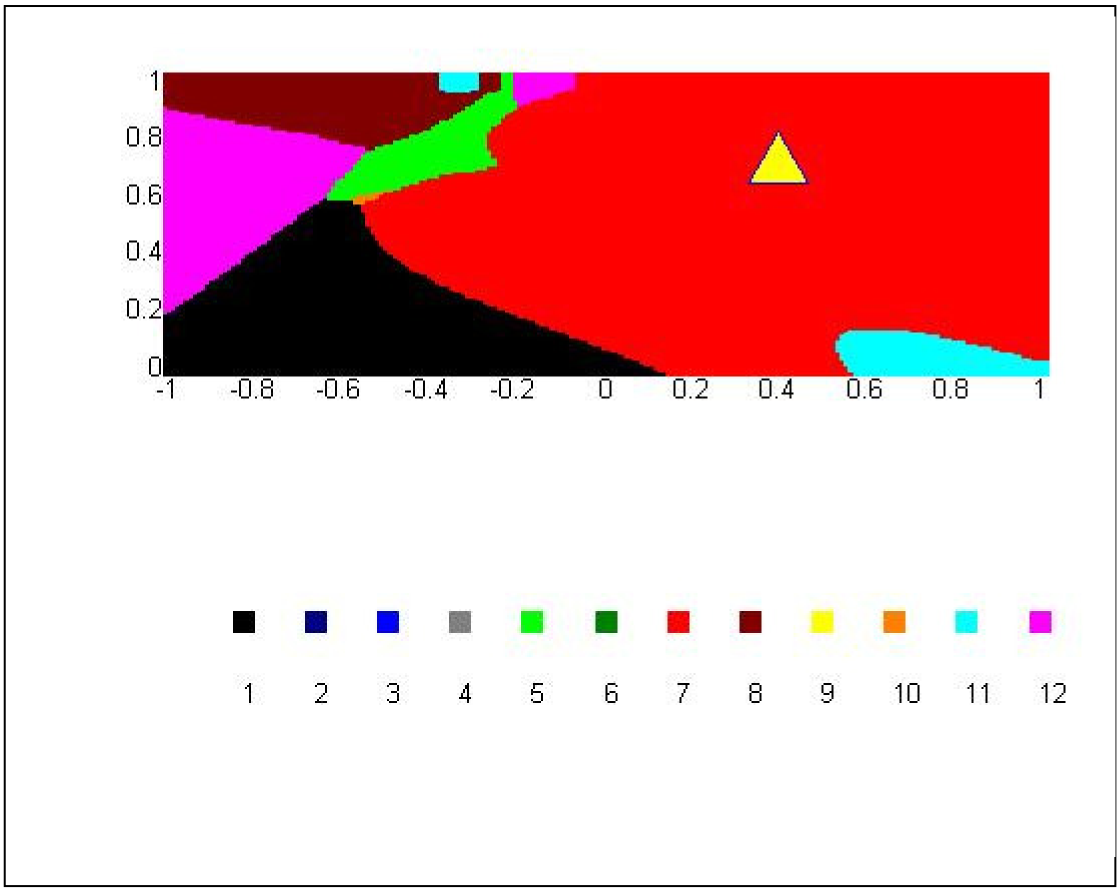


Figure 7.62 Situations for one hill fixed at (0.4, 0.7). In this case a total of seven situations are produced. The setting is quite similar to the setting in Figure 7.61 and therefore, again the two main situations recognized are 1 and 7. The other recognized situations are 5, 8, 10, 11 and 12, and strategies applied to all these situations also use the hill in the east for protection against observation and fire from the enemy for either the *AG* or *FS*, except for the strategy for Situation 5. In the strategy for Situation 5, the *AG* uses the hill in the west which gives more advantage to the attacker.

Situation 10 is expected to be recognized more than it is in this experiment and some part of the area occupied by Situation 5 should be claimed by Situation 10. The larger area occupied by Situation 12 in the west is expected but the small area in the north is somewhat unexpected. The area occupied by Situation 11 in the south east and a small square in the top are not expected. The neural net is not trained for the situations in which the hills are located on the boundaries that are away from the starting positions because there is no importance of these hills in selecting the strategy for attack. Therefore, some of the results for hill(s) on the boundaries are not explainable. Situation 9 is not recognized at all and it is not expected to be recognized because in Situation 9 both of the hills are in the west and that situation never occurs in this setting.

The result of the experiment where the location of one hill is fixed at (-0.17, 0.7) is shown in Figure 7.63.

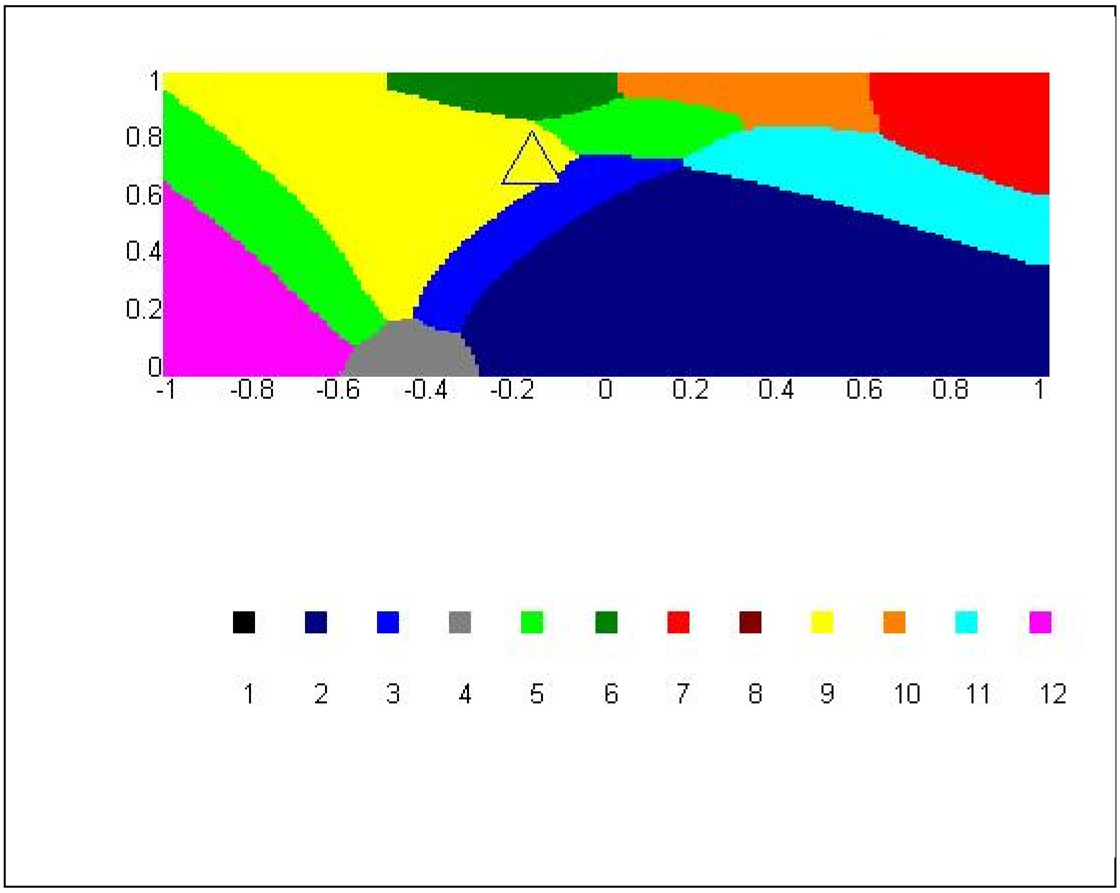


Figure 7.63 Situations for one hill fixed at (-0.17, 0.7). In this case a total of ten situations are produced. More number of situations are recognized in this case as compared to the previous experiment because most of the training examples are based on either both hills or at least one hill in the west therefore, the agent produces ten out of a total of twelve possible situations.

Situations 2, 4, 5, 6, 9, 10 and 11 are expected because of the location of the fixed hill in the west. The difference between Situations 10 and 11 is that one hill in Situation 10 is in the west and in Situation 11 it is in the middle of the battle field while the other is in the west in Situation 10 and in the middle in Situation 11. In this experiment one hill in the west is fixed in the middle of the position of the hill in Situations 10 and 11, and therefore, the recognition of either situation is decided only due to the location of the hill in the east. The area occupied by Situation 7 in the north east, Situation 12 in the south west and part of the area occupied by Situation 5 above the area occupied by Situation 12 is not expected and is probably present due to the reason that neural net is not trained on the boundaries away from enemy and own positions.

The result of the experiment where the location of one hill is fixed at (0, 0) is shown in Figure 7.64. Due to the location of one hill fixed exactly at (0, 0), Situations 1, 2, 3, 4 and 7 are expected. Situation 5, 9 and 12 are recognized due to the location of the moving hill in the regions where the agent is trained to recognize these situations. Situation 11 recognized in the northeast and a small area the shape of a square occupied by Situation 7 in the northeast is not expected. Both of these cases are in the boundary of the battlefield for which the agent is not trained to recognize situations.

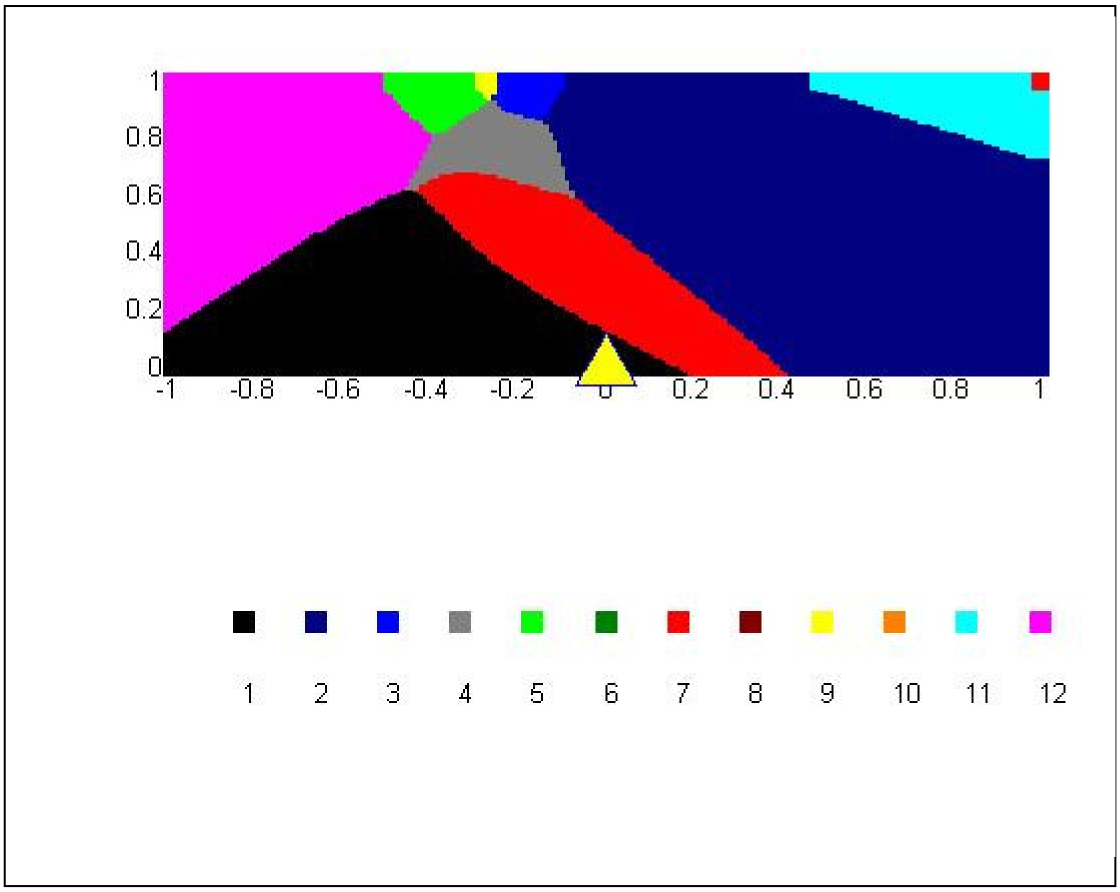


Figure 7.64 Situations for one hill fixed at (0, 0). In this case nine out of twelve Situations are produced. Strategy 2 uses the hill in the west for *AG*, Strategy 3 uses the hill in the north and west of enemy for *FS*, Strategy 4 uses the hill in the south west of the enemy for *FS*, Situations 5 and 9 use the hill in the north and west of enemy for *AG*, Strategy 12 uses the hill in the south west of the enemy for *FS* to the advantage of the attacker.

The agent is generally recognizing the situations correctly and recognizing all twelve situations. The situations recognized for positions on the boundaries away from the enemy and own starting positions on the boundaries are not explainable because the neural net is not trained for these situations as the hills located in these areas do not affect the selection of strategy. The neural net is so structured that it gives a similarity value of the presented situation to all twelve situations. These results show the situation that is recognized with the highest similarity value. The situation selected by the neural net with the highest recognition value is fed to the agent where the strategy associated with recognized situation is implemented. The *RPD-Soar* agent uses mental simulation for selecting course of action at the atomic level in order to implement the goal set by this selected strategy. All the components are available to take the *RPD-Soar* agent to a level where the strategy selected by the neural net is evaluated in a mental model and if it is not suitable then the strategy associated with the next best recognized situation is evaluated. However this has not been implemented as a part of this research.

7.7 Summary

In this chapter, the experiments discussed are aimed at demonstrating the flexibility in decision making and evaluating performance and behaviour of various types of *RPD-Soar* agents. The experiments on the agent discussed in this chapter also demonstrate behaviour variability across agents and behaviour variability within an agent across episodes, test the ability of the agent to recognize a situation in a changing context and test mental simulation capability of the agent for dynamic situations. The experiments on learning demonstrate the ability of the agent to adapt to recurring tasks and transfer the learnt knowledge to other tasks with overlapping problem spaces. The last experiment is related to integration of a trained neural network in the architecture to enhance the situation recognition ability of the agent. The conclusions of the research are provided in the next chapter.

The code required to carry out all the experiments discussed in this chapter is available in the attached CD, see Appendix C.

8 SUMMARY, CONCLUSIONS AND FUTURE WORK

In this chapter, the research work is summarized, important conclusions are listed and the future direction of this research is discussed.

8.1 Summary

The purpose of this research is to propose and implement an architecture to model command agents that addresses some of the deficiencies in decision making and learning that assist in current human behaviour representations for military simulations. In order to achieve the aim of this research, we have developed a computer implementation of the recognition primed decision making (*RPD*) model using the *Soar* cognitive architecture which is referred to as *RPD-Soar* agent in this thesis. The recognition primed decision making model is selected as the most suitable model of naturalistic decision making for the military domain as a result of very comprehensive research carried out by Klein and his associates on the decision making behaviour of military commanders and experts in similar domains. The *Soar* architecture is selected to represent human cognition because of its successful applications in representing human behaviour in the military domain. Moreover, there are many advantages with regards to the implementation of the *RPD* model in *Soar*, and these are discussed in the next paragraph in detail.

Soar provides a convenient frame work to model all three Levels of *RPD*. Recognizing patterns in the environment and proposing applicable operators is already a part of the *Soar* architecture, and if *Soar* has sufficient knowledge then it behaves like Level 1 *RPD*, with only one problem and that is that *Soar* does not allow partial matching of conditions for recognition of a production rule. Level 2 *RPD* except for the story building part, is achieved through the elaboration phase of *Soar*. In the elaboration phase all production rules are matched and fired in parallel. And in this phase any amount of reasoning and processing of the environmental variables may be carried out to understand the situation and extract cues for situation recognition. Level 3 *RPD* has its emphasis on mental simulation and the *Soar* architecture has the capability to take mental simulation to as many steps as is suitable for the application. In *Soar*, if multiple operators with equal preferences are proposed then the architecture considers it lack of sufficient knowledge and creates an *operator tie impasse*, which in turn creates a *sub-state* to bring to bear the knowledge required to resolve this conflict. This *sub-state* is used as the selection space for evaluating these operators. For each proposed operator or action as we call it in *RPD*, an abstract operator called evaluate-operator is proposed, which in turn creates another sub-space through *operator no-change impasse*, which is used as a mental model to evaluate the operator. The objects of the external world are modelled in this mental model and the action required to be evaluated is applied to this mental world to see its effects. If this action is promising then the mental model is dissolved and the action is applied to the real world. If the action does not satisfice then it is thrown away and the next action is mentally simulated by creating another mental model. This process is repeated until one action is selected. If no action is promising then the one with the highest success value among them is selected and in situations where multiple actions have equal success value then one out of them is selected at random.

In *RPD-Soar* agents, the modeller needs to code the behaviour for higher level tasks which is comparatively easier to acquire from domain experts. The knowledge is required to be elicited in the form of experiences with whatever is pertinent for that particular experience from the four components; cues, goals, expectations, and courses of action. The modeller then codes the behaviour of the agent for atomic actions such as turn, move, or fire, which are few as compared to the total number of behaviours and assistance if needed from the domain expert may be acquired. Because of the capability of mental simulation, the behaviour from a sequence of primitive actions emerges automatically, which means the modeller has to design general rules for evaluation of courses of action by modelling the effects of each atomic action on the environment. There is no requirement to give a specific course of action for every new situation from the start. These agents can be further enhanced to exhibit various levels of expertise.

The proposed implementation is evaluated using prototypical scenarios arising in command decision making in tactical situations. The *RPD-Soar* agent recognizes situations within a context and generates goals, expectations, and plausible courses of action accordingly and then *wargames* the course of action by mentally simulating it. Due to the ability of the *RPD-Soar* agent to mentally simulate applicable courses of action it is possible for the agent to handle new situations very effectively using its prior knowledge.

Experiments are developed as a whole to test the proposed implementation for flexibility in decision making strategies, behavioural variability and adaptability. The study compares the behaviour of agents with and without the capability to mentally simulate courses of action, by observing the external environment where the actions as a result of these two decision-making processes are implemented. Various experiments clearly demonstrate that the behaviour exhibited by the *RPD-Soar* agents is consistent with that of plausible human behaviour. The results show how the behaviour of agents is more human like when the agent uses mental simulation, than otherwise.

It has been demonstrated that an experienced agent takes quicker decisions and a less experienced agent may give the same behaviour but with more evaluation that will make it slow to react to situations. In these simulations the agents were taking turns to act, therefore, this effect could only be measured through the number of *Soar* decision cycles. The advantage of experience will be directly observable if the simulation is running on time steps instead of agents acting in turns. The *RPD-Soar* agent exhibits the ability to change decision making strategy with experience, which means the same agent for a situation for which the agent has sufficient knowledge adopts a Level 1 *RPD* strategy and for a situation where the agent has less experience it automatically changes its strategy to Level 3 *RPD*. This is the demonstration of the ability of the agent to possess flexibility in decision making. In this case this change of strategy can be used to either produce agents with varying experience or to represent different levels of knowledge of the same agent for different problems. But this inherent flexibility in decision making strategy can also be used to represent stress in an agent.

The variability in behaviour within an agent is a desirable characteristic. Variability in agents may be produced through randomness but randomness also introduces undesirable behaviour. The observed variability in the *RPD-Soar* agent is due to reasonable but some times sub-optimal choices made by the agent. And the preliminary results clearly demonstrate the ability of the model to represent human behaviour variability within and across individuals.

The overall variability is expected to increase in an environment where more entities are interacting with *RPD-Soar* agent.

Agents adapt using chunking provided by the *Soar* architecture which is a form of explanation based learning. Learning through chunking in *Soar* is the process of remembering the results of the *sub-goals*. In terms of *RPD-Soar* agents it is the process of changing from a Level 3 *RPD* to Level 1 *RPD*. The latter is more efficient and is an indicator of the experience of the agent.

The *RPD*-Soar agents have demonstrated the ability to transfer learnt knowledge from one task to the other. If the agent has learnt an experience from one task and a similar situation arises in the other task then the agent is observed to use this knowledge.

In rule based systems the antecedents of the production rule have to match exactly for the production to fire. If the current situation deviates from the conditions in the rule then the appropriate rule does not fire. Due to rule matching through efficient algorithms and also advances in computer technology it is possible in *Soar* to add a large number of production rules to handle generalization. But writing large number of rules is not an efficient method of solving this problem. To enhance the ability of the agent to recognize new situations an artificial neural network is integrated in the architecture. The neural net is trained on the example experiences that the *RPD-Soar* agent is likely to face in the simulation. For this experiment the neural net is trained to recognize situations presented by the terrain in order to develop a strategy to attack the enemy tank with the help of a troop of own tanks.

8.2 Conclusions

The need for realistic decision making in military simulations has been identified. Much of the *recognition primed decision making* (*RPD*) model has been successfully implemented using the *Soar* cognitive architecture. It has been demonstrated that the agents developed using *RPD* exhibit a rich variety of desirable behaviours within the domain considered. Some of the salient features of the work are summarized below:-

- *Soar* cognitive architecture provides the basic framework to model most aspects of *recognition primed decision making* (*RPD*) model. The *RPD* model implemented in the *Soar* cognitive architecture is capable of mimicking some of the decisions made by military commanders in battlefield settings.
- Level 1 *RPD* has been completely implemented for situations where sufficient knowledge is available. The agent straight away recognizes a situation as typical and selects a course of action for that situation to implement.

- That part of Level 2 *RPD* has been implemented where situations are not recognized straight away and information from the environment is required to be processed and combined with already available knowledge in order to diagnose and then recognize a situation as typical.
- The Level 2 *RPD* for very complex situations require story building to account for some of the inconsistencies in situation recognition. This part of Level 2 *RPD* has not been implemented in this model.
- Mental simulation which forms the basis of Level 3 *RPD* has been implemented in this model with such flexibility to accommodate all types of requirements that are expected to be encountered while making decisions using *RPD* model.
- Flexibility in decision making strategies based on psychological theories is achieved. Decision making strategies are based on experience and extent of knowledge.
- Variability in behaviour across individuals is a desirable characteristic in human behaviour representation. Variability in behaviour across individuals is achieved based on the type of experiences in long term memory of similar agents. *Within-entity variability* is achieved in this model not through randomness which introduces undesirable behaviour but through reasonable but sometimes sub-optimal choices made by the agent.
- Command agent of the developed model exhibits adaptability across various episodes which adds the much desired dynamism to the simulation environment. The agent learns from its experience. The learning is based on the chunking phenomenon inherent in *Soar* which is a form of explanation-based generalization.
- The agents also exhibit transfer of knowledge from one task to the other in case of overlapping problem spaces within tasks.
- Due to the ability of the agents to mentally simulate courses of action it is possible for the agent to handle new situations very effectively. This relieves the modeller from coding behaviours for all situations expected to be encountered in a simulation and this in turn reduces the development time of the agent.

- The strategies to form experiences in the long term memory of the agents are required only at a higher level with general rules to evaluate actions at lower levels which is easier for the subject matter expert to describe and less tasking for the knowledge engineer to elicit. This reduces the time and effort in the development of the agent. Following this the mental simulation and learning abilities can be used to improve the agent.
- The ability of the agent to handle new situations is further enhanced using a trained artificial neural network which is integrated in the proposed architecture. This further reduces the labour of the modeller in coding behaviours for all expected situations.
- The research also developed a simple *RPDAgent* to operate in a simple simulation environment in order to explore the affect of realistic human decision making on the outcome of the battle simulations. The study concludes that the outcome of the constructive military simulations changes if more realistic human behaviour is incorporated in these simulations, and the known mathematical and probabilistic solutions for combat modelling help in validating the start point or base line of simulations involving human behaviour.
- In order to develop an agent for a different domain based on *RPD-Soar* model following tasks are required to be completed:
    - Change the objects of this implementation to the objects of the domain of interest. The structure for tree and graph representation is already available as *SML* code in this implementation and can be utilized as per the requirements.
    - Elicit knowledge about the domain from the experiences of the subject matter expert and transform it to the form of goals, cues, expectations and courses of action in the light of the examples in this implementation.
    - Convert the experiences into *Soar* rules using the rules of this implementation as examples.
    - Set the goals as the desired state. Convert the cues and expectations into conditions, and courses of actions as operators.

- Give the success values of all courses of action applicable to a situation as numeric preferences to operators in *Soar* rules.
- Identify the objects that need to be represented in the mental model and also the level of attributes that need to be represented. Change the objects and their attributes using the *Soar* rules in *Selection space* as examples.
- Give preference to operators to select for evaluation. Write *Soar* rules to implement the selected operator (the course of action) in the mental model.
- Write *Soar* rules to evaluate the situation after the selected course of action is implemented to find out whether this action is likely to take the agent to its goal or otherwise. The factors on which a course of action is evaluated may be different for different domains but the *Soar* rules of this implementation can be used as examples.

8.3 Future work

In this section, a list of future directions of work is enumerated. This is important as it is likely to provide a clear perspective to this research work.

- The model is required to be tested in a richer context. There are two options as regards selection of the environment for rich context. The first option is to integrate the agent with *ModSAF* (or similar simulation environment), where the agent takes higher level decisions and the *ModSAF* entities implement the commands in the *ModSAF* environment. This option has some problems regarding the restriction on availability of *ModSAF* code. The second option is to integrate the *RPD-Soar* agent in some computer game application and for this option *Unreal Tournament* is a possible candidate due to the availability of its code.
- The number of steps in the mental simulation should be increased from the present implementation of a single step only. The next phase in this direction it should be related to the complexity of the decision problem and time constrained decision making.
- Decision strategy is presently related to knowledge. It should also be related to stress due to time and physiological conditions of the decision maker.

- The basic platform of variable behaviour is demonstrated in this implementation. It should be linked to behaviour moderators like fatigue, fear, morale etc.
- Reinforcement learning has been recently incorporated in *Soar* (Nason and Laird, 2005). Reinforcement learning should be incorporated to adapt the success values of courses of action. The effect of reinforcement learning on variability in behaviour should also be analyzed.
- A synthetic life of the agent should be created using the episodic memory proposed by Nuxoll and Laird (2004) and (2007).
- The plan selected by the neural network should be modified if necessary by the agent after evaluation through mental simulation.

9 REFERENCES

AGENT ORIENTED SOFTWARE LTD. (2008) JACK Intelligent Agents: Agent Manual Release 5.3.

ANDERSON, J. R. (1993) Rules of the mind, Hillsdale, N.J. ; Hove, L. Erlbaum.

ANDERSON, J. R. & LEBIERE, C. (1998) The atomic components of thought, Mahwah, NJ: Lawrence Erlbaum Associates.

ANDERSON, J. R., BOTHELL, D., BYRNE, M. D., DOUGLASS, S., LEBIERE, C. & QIN, Y. (2004) An integrated theory of the mind. Psychological Review, 111, pp. 1036-1060.

U.S. ARMY (1997) Knowledge and Speed: The Annual Report of the Army After Next Project. Washington, DC: U.S. Army Chief of Staff.

U.S. Department of the Army (1990) Terrain Analysis, FM 5-33. Headquarters, Department of the Army, Washington, D.C.

U.S. Department of the Army (1993) Staff Organization and Operations, FM 101-5. Headquarters. Fort Monroe, VA.

http://www.globalsecurity.org/military/library/policy/army/fm/101-5/f540.pdf

BANKS, J. (2005) Discrete-event system simulation, Upper Saddle River, N.J. ; London, Pearson Prentice Hall.

BEST, B. J., LEBIERE, C. & SCARPINNATTTO, K. C. (2002) Modelling Synthetic Opponents in MOUT Training Simulations using ACT-R Cognitive Architecture. Paper presented at the 11th Conference on Computer Generated Forces and behaviour Representation, Orlando.

BOOK, E. (2002) Central Fla. School promotes advanced simulation studies. National Defense. Iss. November.

http://www.nationaldefensemagazine.org/issues/2002/Nov/Central_Fla.htm

BOWDEN, F. D. J., GABRISCH, C. & DAVIES, M. (1997) C3I systems analysis using the Distributed Interactive C3I Effectiveness (DICE) simulation. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 5, 4326-4331.

BRATMAN, M. (1987) Intention, Plans, and Practical Reasoning, Harvard University Press,Cambridge, MA. Reprinted in 1999, CSLI Publications, Stanford, CA.

BURNS, K. (2000) Mental models and normal errors. Proceedings of the 5th Conference on Naturalistic Decision Making, May 26-28. Tammsvik, Sweden.

CERANOWICZ, A. (1994a) ModSAF Capabilities. Proceedings of the Forth Conference on Computer Generated Forces and Behavioral Representation, May 4-6, 1994, University of Central Florida, Orlando, pp. 3-8.

CERANOWICZ, A. (1994b) Operator Control of Behavior in ModSAF. Proceedings of the Forth Conference on Computer Generated Forces and Behavioral Representation, May 4-6, 1994, University of Central Florida, Orlando, pp. 9-16.

CAMPBELL, L., LOTMIN, A. & DERICO, M. M. G. R., C. (1997) The use of artificial intelligence in military simulations. Systems, Man, and Cybernetics, 'Computational Cybernetics and Simulation', IEEE International Conference on, Vol.3, Iss., 12-15 Oct 1997 pp. 2607-2612.

STANLEY, C., PORAC, C. & WARD, L. M. (1978) Sensation and Perception (International Edn.). New York: Academic Press.

D'INVERNO, M., KINNY, D., LUCK, M. & WOOLDRIDGE, M. (1998) A Formal Specification of dMARS. Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages, pp. 155-176. Springer-Verlag.

DEPARTMENT OF DEFENSE (1998) "DoD Modeling and Simulation (M&S) Glossary", DoD 5000.59-M, 01/1998.

DOMPKE, U. (2001) Simulation of and for Military Decision Making. RTO SAS Lecture Series, Italy, 15-16 October.

ENDSLEY, M. R. (1995) Toward a theory of situation awareness in dynamic systems. Human Factors, 37, pp. 32-64.

ERWIN, S. I. (2000) Simulation of Human Behavior Helps Military Training Models. National Defense, Vol. 85, No. 564, pp. 32-35.

ERWIN, S. I. (2001) Commanders Want Realistic Simulations. National Defense, Vol. 85, No. 567, pp. 50-51.

ESD, A. M. A. O. R. (2004) Foundations of modelling and simulation. Lecture notes, DCMT, Cranfield University.

FAN, X., SUN, S., MCNEESE, M. & YEN, J. (2005) Extending the recognition-primed decision model to support human-agent collaboration. AAMAS 05, pp945-952. ACM Press.

FERNLUND, H., GONZALEZ, A., GEORGIOPOULOS, M. & DEMARA, R. (2006) Learning tactical human behavior through observation of human performance. IEEE TRANSACTIONS ON SYSTEMS MAN AND CYBERNETICS PART B-CYBERNETICS, 36, pp. 128-140.

FITTS, P. M. (1954) The information capacity of the human motor system in controlling the amplitude of movement. JOURNAL OF EXPERIMENTAL PSYCHOLOGY, 47, pp. 381-391.

FORBUS, K. D., USHER, J. & CHAPMAN, V. (2004) Qualitative spatial reasoning about sketch maps. AI Magazine, 25, pp. 61-72.

FORGY, C. (1982) Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, 19, pp. 17 - 37.

FORGY, E. (1965) Cluster analysis for multivariate data: Efficiency vs. interpretability of classifications. Biometrics, 21.

FORSYTHE, C. & WENNER, C. (2000) Surety of human elements of high consequence systems: An Organic Model. Proceedings of the IEA 2000 / HFES 2000 Congress, Vol. 3, pp. 839-842. San Diego, CA.

FORSYTHE, C. & XAVIER, P. G. (2002) Human emulation: Progress towards realistic synthetic human agents. 11th Conference on Computer-generated Forces and Behavior Representation. pp. 257-266. Orlando, FL.

GEORGEFF, M., PELL, B., POLLACK, M., TAMBE, M. & WOOLDRIDGE, M. (1999) The belief-desire-intention model of agency. Intelligent Agents V, pp. 1-10.

GEORGEFF, M. P. & INGRAND, F. F. (1990) Real-time reasoning: The monitoring and control of spacecraft systems. Proceedings of the sixth conference on Artificial intelligence applications, pp. 198-204.

GEORGEFF, M. P. & RAO, A. S. (1996) A profile of the Australian Artificial Intelligence Institute. IEEE Expert-Intelligent Systems and their Applications, 11, pp. 89-92.

GIARRATANO, J. C. & RILEY, G. (1998) Expert systems : principles and programming, Boston ; London, PWS.

GIBSON, E. (1990) Recency preferences and garden-path effects. Proceedings of the Twelfth Annual Conference of the Cognitive Science Society, 372-379.

GLADWELL, M. (2005) Blink : the power of thinking without thinking, London, Allen Lane.

GOLDMAN, L., COOK, E. F., JOHNSON, P. A., BRAND, D. A., ROUAN, G. W. & LEE, T. H. (1996) Prediction of the Need for Intensive Care in Patients who Come to Emergency Departments with Acute Chest Pain. THE NEW ENGLAND JOURNAL OF MEDICINE, 334, pp. 1498-1504.

GONZALEZ, A. J. & AHLERS, R. (1998) Context-based representation of intelligent behavior in training simulations. Transactions of the Society for Computer Simulation, 15, pp. 153-166.

GONZALEZ, R. E. & WOODS, R. E. (2002) Digital image processing, Pearson Education Singapore.

GROSZ, B. J. & KRAUS, S. (1996) Collaborative plans for complex group action. Artificial Intelligence, 86, pp. 269-357.

GROUND, L., KOTT, A. & BUDD, R. (2002) A knowledge-based tool for planning of military operations: The coalition perspective. Proceedings of the Second International Conference on Knowledge Systems for Coalition Operations.

GURNEY, K. (1997) An introduction to neural networks, CRC Press.

HASTIE, T. J., TIBSHIRANI, R. J. & FRIEDMAN, J. H. (2001) The elements of statistical learningdata mining, inference, and prediction : with 200 full-color illustrations, New York, Springer.

HAYES, C. C., SCHLABACH, J. L. & FIEBIG, C. B. (1998) FOX-GA: An intelligent planning and decision support tool. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 3, pp. 2454-2459.

HILES, J. (2002) Innovations in computer generated autonomy. Naval Postgraduate School Technical Report NPS-MV-02-002.

HILL, J., W., R., CHEN, J., GRATCH, J., ROSENBLOOM, P. & TAMBE, M. (1997) Intelligent agents for the synthetic battlefield: a company of rotary wing aircraft. Innovative Applications of Artificial Intelligence, pp. 1006-1012.

HINTZMAN, D. L. (1984) MINERVA 2: A simulation model of human memory. Behavior Research Methods, Instruments, & Computers, 16, pp. 96-101.

HINTZMAN, D. L. (1986) Schema abstraction in a multiple-trace memory model. Psychological Review, 93, pp. 429-445.

ILACHINSKI, A. (2004) Artificial war : multiagent-based simulation of combat, River Edge, NJ, World Scientific Pub.

JI, Y., MASSANARI, R. M., AGER, J., YEN, J., MILLER, R. E. & YING, H. (2007) A fuzzy logic-based computational recognition-primed decision model. Information Sciences, 177, pp. 4338-4353.

JOHNSON, T. (1997) Control in ACT-R and soar. Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society, pp. 343-348.

JOHNSON, W. L. (1994a) Agents that explain their own actions, In Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, pp. 87-95.

JOHNSON, W. L. (1994b) Agents that learn to explain themselves, *In Proceedings of the National Conference on Artificial Intelligence*, pp. 1257-1263, AAAI press.

JOHNSON, W. R., MASTAGLIO, T. W. & PETERSON, P. D. (1993) Close combat tactical trainer program. Winter Simulation Conference Proceedings, pp. 1021-1029.

JONES, R. M., LAIRD, J. E., NIELSEN, P. E., COULTER, K. J., KENNY, P. & KOSS, F. V. (1999) Automated intelligent pilots for combat flight simulation. AI Magazine, 20, pp. 27-41.

KAEMPF, G. L., KLEIN, G., THORDSEN, M. L. & WOLF, S. (1996) Decision making in complex naval command-and-control environments. Human Factors, 38, pp. 220-231.

KAUFMAN, L. & ROUSSEEUW, P. J. (1990) Finding groups in data: An introduction to cluster analysis, New York, Wiley.

KERSHAW, T. C. & OHLSSON, S. (2001) Training for insight: The case of the nine-dot problem. Proceedings of the Twenty-third Annual Conference of the Cognitive Science Society, pp. 489-493.

KILLEBREW, R. B. (1998) Learning from Wargames: A Status Report. Parameters, Spring 1998, pp. 122-35.

KLEIN, G. A. (1998) Sources of power : how people make decisions, Cambridge, Mass. ; London, MIT Press.

KLEIN, G. & KLINGER, D. (2000) Naturalistic Decision Making. Human Systems IAC GATEWAY, XI, pp. 16-19.

KLIMESCH, W. (1996) Memory processes, brain oscillations and EEG synchronization. International Journal of Psychophysiology, 24, pp. 61-100.

KOHAVI, Z. (1978) Switching and finite automata theory, New York, McGraw-Hill.

KOLODNER, J. L. (1993) Case-based learning, Boston ; London, Kluwer Academic Publishers.

KRESS, M. & TALMOR, I. (1999) New look at the 3:1 rule of combat through Markov Stochastic Lanchester models. Journal of the Operational Research Society, 50, pp. 733-744.

KUNDE, D. & DARKEN, C. (2005) Event Prediction for Modeling Mental Simulation in Naturalistic Decision Making. Proceedings of BRIMS 2005.

KUNDE, D. & DARKEN, C. (2006) A Mental Simulation-Based Decision-Making Architecture Applied to Ground Combat. Proceedings of BRIMS 2006.

LAIRD, J. E. (2006a) The Soar 8 Tutorials 1 - 8. Electrical Engineering and Computer Science Department, University of Michigan.

http://sitemaker.umich.edu/soar/home

LAIRD, J. E. (2006b) *Soar* User's Manual Version 8.6 Edition 1. Electrical Engineering and Computer Science Department, University of Michigan.

http://sitemaker.umich.edu/soar/home

LAIRD, J. E. & NEWELL, A. (1983) A universal weak method: Summary of results. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, pp. 771-773.

LAIRD, J. E., NEWELL, A. & ROSENBLOOM, P. S. (1987) SOAR - An architecture for general intelligence. ARTIFICIAL INTELLIGENCE, 33, pp. 1-64.

LAIRD, J. E. (2000) It Knows What You're Going To Do: Adding anticipation to a QuakeBot. AAAI 2000 Spring Symposium on Artificial Intelligence and Interactive Entertainment. AAAI Technical Report SS00–02. Menlo Park, CA: AAAI Press.

LAIRD, J. E. (2000) An exploration into computer games and computer generated forces. Proc. 9th Conf. Computer Generated Forces and Behavioral Representation.

LANCHESTER, F. W. (1916) Aircraft in warfare: The dawn of the fourth arm, Constable & Co, London, England.

LAVALLE, S. M. (2006) MPlanning Algorithms, Cambridge University Press.

LEVESQUE, H. J., COHEN, P. R. & NUNES, J. H. T. (1990) On acting together. Proceedings of AAAI-90, pp. 94-99.

LIANG, Y., ROBICHAUD, R. & FUGERE, B. J. (2001) Implementing a naturalistic command agent design. Proceedings of the Tenth Conference on Computer Generated Forces, 379-386.

LIPSHITZ, R., KLEIN, G., ORASANU, J. & SALAS, E. (2001) Focus article: Taking stock of naturalistic decision making. Journal of Behavioral Decision Making, 14, pp. 331-352.

LUCAS, A. & GOSS, S. (1999) The potential for intelligent software agents in defence simulation. IN GOSS, S. (Ed.) Information, Decision and Control, pp. 579-583.

LUCHINS, A. S. (1942) Mechanization in problem solving. Psychological Monographs, 54(Whole No. 248).

MASON, C. R. & MOFFAT, J. (2001) An agent architecture for implementing command and control in military simulations. Proceedings of the Winter Simulation Conference, pp. 721-729.

MCNAUGHT, K. R. (2002) Markovian models of three-on-one combat involving a hidden defender. Naval Research Logistics, 49, pp. 627-646.

MCNETT, M. D., PHELAN, R. G., JR. & MCGINNIS, M. L. (1997) WARSIM 2000: combining multiple expert opinions from subject matter experts to generate requirements for staff training at battalion level and above. Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference, 2, pp. 1280-1284.

MITCHELL, T. (1997) Machine Learning, McGraw Hill International Editions.

NASON, S. & LAIRD, J. E. (2005) Soar-RL: Integrating reinforcement learning with Soar. Cognitive Systems Research, 6, pp. 51-59.

NEWELL, A. (1990) Unified theories of cognition, Cambridge, Mass. ; London, Harvard University Press.

NICOL, D. M., BALCI, O., FUJIMOTO, R. M., FISHWICK, P. A., L'ECUYER, P. & SMITH, R. (1999) Strategic directions in simulation research (panel).

Proceedings of the 31st conference on Winter simulation: Simulation---a bridge to the future - Volume 2. Phoenix, Arizona, United States, ACM.

NORLING, E. (2004) Folk psychology for human modelling: Extending the BDI paradigm. In Proceedings of the Third international Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1 (New York, July 19 - 23). International Conference on Autonomous Agents, pp. 202-209. IEEE Computer Society. Washington, DC.

NORLING, E. & SONENBERG, L. (2002) An approach to evaluating human characteristics in agents. Proceedings of the International Workshop on Regulated Agent-Based Systems: Theories and Applications (RASTA'02), pp. 51-60.

NORLING, E. & SONENBERG, L. (2004) Creating interactive characters with BDI agents. Proceedings of the Australian Workshop on Interactive Entertainment IE2004. Sydney, Australia.

NORLING, E., SONENBERG, L. & RONNQUIST, R. (2001) Enhancing multi-agent based simulation with human-like decision making strategies. MULTI-AGENT-BASED SIMULATION, 1979, pp. 214-228.

NUXOLL, A. & LAIRD, J. E. (2004) A Cognitive Model of Episodic Memory Integrated with a General Cognitive Architecture. Proceedings of the Sixth International Conference on Cognitive Modeling, pp. 220-225.

NUXOLL, A., LAIRD, J. E. & JAMES, M. R. (2004) Comprehensive working memory activation in Soar. Proceedings of the 6th International Conference on Cognitive Modeling, pp. 226-230.

NUXOLL, A. M. & LAIRD, J. E. (2007) Extending cognitive architecture with episodic memory. In Proceedings of the 21st National Conference on Artificial Intelligence-(AAAI), pp. 1560-1564.

NWANA, H. S. (1996) Software Agents: An Overview. Knowledge Engineering Review, Vol. 11, pp. 1-40.

PECK, M. (2004) Computer games helping to train commanding officers. National Defense. Iss. December.

http://www.nationaldefensemagazine.org/issues/2004/Dec/ComputerGamesHelping.htm

PEW, R. W. & MAVOR, A. S. (1998) Modeling human and organizational behavior : application to military simulations, Washington, D.C. ; [Great Britain], National Academy Press.

PRATT, D. R. & JOHNSON, M. A. (1995) Constructive and virtual model linkage. Winter Simulation Conference Proceedings, pp. 1222-1228.

RAO, A. S. & GEORGEFF, M. P. (1995) BDI agents: From theory to practice. Proceedings of the First International Conference on Multi-Agent Systems, pp. 312-319.

RASMUSSEN, J. (1985) The role of hierarchical knowledge representation in decisionmaking and system management. IEEE Transactions on Systems, Man and Cybernetics, 15, pp. 234-243.

RAZA, M. & SASTRY, V. V. S. S. (2007) Command Agents with Human-Like Decision Making Strategies. Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), Vol. 2, pp. 71-74. IEEE Computer Society.

RAZA, M. & SASTRY, V. V. S. S. (2008) Variability in Behavior of Command Agents with Human-Like Decision Making Strategies. Tenth International Conference on Computer Modeling and Simulation (uksim 2008), pp. 562-567. Cambridge, England.

RITTER, F. (2007) Personal communication.

RITTER, F. E., SHADBOLT, N. R., ELLIMAN, D., YOUNG, R., GOBET, F., & BAXTER, G. D. (2002) Techniques for modeling human performance in synthetic environments: A supplementary review. Wright-Patterson Air Force Base, OH: Human Systems Information Analysis Center.

ROSS, K. G., KLEIN, G. A., THUNHOLM, P., SCHMITT, J. F. & BAXTER, H. C. (2004) The Recognition-Primed Decision Model. Military Review, July-August, 6.

RUBINSTEIN, R. Y. (1981) Simulation and the Monte Carlo method, John Wiley and Sons.

RUMMERY, G. A. & NIRANJAN, M. (1994) On-line Q-learning using connectionist system. Cambridge University Engineering Department, CUED/FINENG/TR, 166.

RUSSELL, S. J. & NORVIG, P. (2003) Artificial intelligence : a modern approach, Upper Saddle River, N.J. ; [Great Britain], Prentice Hall.

SCHANK, R. C. & ABELSON, R. P. (1977) Scripts, plans, goals and understanding : an inquiry into human knowledge structures, Hillsdale, N.J., Erlbaum ; New York ; London : Distributed by Wiley.

SCHOENWALD, D., XAVIER, P., THOMAS, E., FORSYTHE, C. & PARKER, E. (2002) Simulation of a Cognitive Algorithm for a Distributed Robotic Sensing Network. World Automation Congress, 2002. Proceedings of the 5th Biannual. vol.14, pp 7-12.

SOKOLOWSKI, J. (2002) Can a composite agent be used to implement a recognition-primed decision model. Proceedings of the Eleventh Conference on Computer Generated Forces and Beharioral Representation, pp. 431-436.

SOKOLOWSKI, J. (2003a) Enhanced military decision modeling using a MultiAgent system approach. Proceedings of the Twelfth Conference on Behavior Representation in Modeling and Simulation, pp. 179-186.

SOKOLOWSKI, J. A. (2003b) Representing Knowledge and Experience in RPDAgent. 12th Conference on Behavior Representation in Modeling and Simulation (BRIMS). May 12-15, Scottsdale, AZ.

SOKOLOWSKI, J. A. (2003c) Modeling the decision process of a joint task force commander. Norfolk, Virginia, USA., Old Dominion University.

SOMMERVILLE, I. (2004) Software engineering, Boston ; London, Pearson/Addison Wesley.

STENSRUD, B. (2005) FAMTILE: An Algorithm for Learning High-Level Tactical Behavior from Observation. School of Electrical and Computer Engineering. Orlando, Florida, University of Central Florida.

STERNBERG, S. (1975) Memory scanning - new findings and current controversies. QUARTERLY JOURNAL OF EXPERIMENTAL PSYCHOLOGY, 27, pp. 1-32.

SUN, S., COUNCIL, I., FAN, X., RITTER, F. E. & YEN, J. (2004) Comparing Teamwork Modeling in an Empirical Approach. Proceedings of the sixth international conference on cognitive modeling, pp. 388-389. Mahwah, NJ: Erlbaum.

TAMBE, M. and ROSENBLOOM, P. S. (1995) RESC: An approach for real-time, dynamic agent tracking, In Proceeding of the 14th International Joint Conference, AI, pp. 103-110.

TAMBE, M. (1997) Towards flexible teamwork. JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH, 7, pp. 83-124.

THREEPENNY (2005) Soar 8 Documentation - SML quick start guide. Threepenny Software LLC.

http://sitemaker.umich.edu/soar/home

TULVING, E. (1983) Elements of episodic memory, Oxford, Clarendon.

TURING, A.M., (1936-7) On Computable Numbers, With an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, (2) 42, pp 230-265; correction ibid. 43, pp 544-546.

TURING, A. M. (1950) Computing machinery and intelligence. Mind, 59, pp. 433-460.

WAND, K. & BATHE, M. R. (2008) Personal communication.

WARWICK, W., MCILWAINE, S., HUTTON, R. & MCDERMOTT, P. (2001) Developing computational models of recognition-primed decision making. Proceedings of the Tenth Conference on Computer Generated Forces, pp. 232-331.

WRAY, R. & LAIRD, J. (2003) Variability in human behavior modeling for military simulations. Proceedings of the 2003 Conference on Behavior Representation in Modeling and Simulation.

WRAY, R. E., LAIRD, J. E., NUXOLL, A., STOKES, D. & KERFOOT, A. (2005) Synthetic adversaries for urban combat training. AI Magazine, 26, pp. 82-92.

YEN, J., YIN, J., IOERGER, T., MILLER, M., XU, D. & VOLZ, R. (2001) CAST : Collaborative agents for simulating teamwork. Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01), pp. 1135-1142.

YEN, J., FAN, X., SUN, S., HANRATTY, T. & DUMER, J. (2006) Agents with shared mental models for enhancing team decision makings. Decision Support Systems, 41, pp. 634-653.

10 APPENDIX A – LIST OF ACRONYMS

AAII	Australian artificial intelligence institute
ABM	Agent based model
ACT-R	Adaptive control of thought – rational
ACT-R/PM	Adaptive control of thought – rational/perception motor
AG	Assault group
AGL	Above ground level
AI	Artificial intelligence
ANN	Artificial neural network
BDI	Belief, desire, and intentions
BTC	Blue tank commander
CA	Composite agents
CASTFOREM	Combined arms and support task force evaluation model
CAST	Collaborative agent for simulating team behaviour
CBR	Case-based reasoning
CBS	Corps battle simulation
CCTT	Close combat tactical trainer
CFOR	Command forces
CGF	Computer generated forces
CTDB	Compact terrain database
DA	Decision agent
DEM	Digital elevation model
DIANA	DIvisive ANAlysis
DIS	Distributive interactive simulations
DL	Deterministic Lanchester square law
dMARS	Distributed Multi-Agent Reasoning System
DME	Declarative memory element
DoD	Department of defense
DSTO	Defence and science technology organisation
EBL	Explanation-based generalization
EINSTein	Enhanced ISSAC neural simulation toolkit
ESL	Exponential stochastic Lanchester

FM	Field manual
FS	Fire support
FSM	Finite state machines
FWA	Fixed wing attack
GUI	Graphical user interface
HBR	Human behaviour representation
HBM	Human behaviour model
ISAAC	Irreducible semi-autonomous adaptive combat
IA	Intelligent agent
IP	Internet protocol
IPB	Intelligence preparation of the battlefield
IFOR	Intelligent forces
LOS	Line of sight
LTM	Long term memory
MANA	Map aware non-uniform automata
MAS	Multi-agent system
MAUA	Multi attribute utility analysis
MDMP	Military decision making process
METT-T	Mission, enemy, terrain, troops, and time available
ModSAF	Modular semi-automated forces
MOUT	Military operations on urban terrain
NDM	Naturalistic decision making
Oasis	*O*ptical *a*ircraft *s*equencing using *i*ntelligent *s*cheduling
OCOKA	Observation, cover and concealment, obstacles, key terrain, and avenues of approach
PRS	Procedural reasoning system
RA	Reactive agent
R-CAST	RPD enabled collaborative agents for simulating teamwork
RL	Reinforcement learning
RPD	Recognition primed decision making
RWA	Rotary wing attack
SIMNET	Simulator networking
SAF	Semi-automated force

SARSA	State action reward state action
SCA	Symbolic constructor agents
SME	Subject matter expert
SML	*Soar* mark-up language
SSKP	Single shot kill probabilities
STRICOM	Simulation, training, and instrumentation command
STOW	Synthetic theatre of war
STEAM	Shell for teamwork
STM	Short term memory
SWARMM	Smart whole air mission model
TIN	Triangulated irregular network
TRADOC	Training and doctrine command
UTC	Unified theories of cognition
WM	Working memory
WME	Working memory element

11 APPENDIX B – GLOSSARY OF TERMS

Agent: A computer program that persists and operates as an entity in a simulation.

Assault group: A group of force within a military organization like platoon, squadron, or task force, etc that is assigned the mission of physically attacking the enemy forces or positions.

Atomic level action: An action that is not further decomposed in the model.

Attribute: A property of an entity.

Behaviour: The outcome of a continuous process of decision making by an agent operating in its environment while attempting to carry out a task.

Command agents: Intelligent agents representing human combatant or a military commander leading a group of combatants or human controlled platforms that autonomously take decisions in military simulations.

Defilade: The protection of a position, vehicle, or troops against enemy observation or gunfire.

Enfilade: A volley of gunfire directed along a line from end to end.

Entity: An entity is an object of interest in the system.

Fire support: Fire support is the support provided by one mobile or static group of combatants to the other by fire using available weapon systems.

Higher level action: An action that can be decomposed further into higher level or atomic actions.

Intelligent agent: Intelligent agent is an autonomous, learning, and cooperating agent that continuously interacts with its environment in pursuit of a mission assigned to it mimicking human intelligence.

Platform: Representation of a transport or fighting vehicle, or a weapon system with one or more crew members. The over all behaviour of the platform is controlled by the decision of its commander.

Situation variable: The elements in an environment that define a situation for an agent.

12 APPENDIX C - PROJECT SOFTWARE

The complete research work is developed in Java using Eclipse as the development environment. All types and versions of agents with their simulation environments are stored as Eclipse projects with all its necessary files. The software is included in the CD attached with the back cover.

12.1 RPD-Soar agents

All types and versions of RPD-Soar agents, their required Soar files and data link libraries to include *ElementXML.dll*, *SoarKernelSML.dll*, and *Java_sml_ClientInterface.dll* are stored as Java projects in workspace *Eclipse1* in the attached CD. The class libraries *sml.jar* and *swt.jar* need to be added from *Soar*.
All the programs required to extract data from *stats* file generated from *Soar* are also included in each project.

12.1.1 Experiment described in implementation

Advance to contact military operation is stored in the project directory *Eclipse1\RPDSoar-MentalSimVer5*. *Simulation.java* is the main class.

12.1.2 Experiment 1 – Varying performance due to experience

The code to carry out Experiment 1 is available in the project directory *Eclipse1\RPDSoar-MentalSim140207*. *Random–walk* agent, *Less-experienced RPD-Soar* agent and the *Experienced RPD-Soar* agent are all stored as separate Soar files and can be inserted in the class *Environment.java* to be loaded. *TankSimulation.java* is the main class.

12.1.3 Experiment 2 – Changing context

Changing context due to number of obstacle is available in project directory *Eclipse1\RPDSoar-MentalSimVer2.0-140207*. Changing context due to enlarged environment is available in project directory *Eclipse1\RPDSoar-MentalSimVer2.2*. *TankSimulation.java* is the main class in both cases.

12.1.4 Experiment 3 – Variability within an agent

The complete experiment for variability with in agent is set up from the project *Eclipse1\RPDSoar-MentalSimVer3.2*. The starting location of the Red agent is required to be changed according to the coordinates specified in each part of the experiment. *Simulation.java* is the main class.

12.1.5 Experiment 4 – Learning

The complete experiment for learning in an agent is set up from the project Eclipse1\RPDSoar-MentalSimVer3.3. The starting location of the Red agent is required to be changed according to the coordinates specified in each task of the experiment and code for transfer of learning is available as comments which can be uncommented to run. *Simulation.java* is the main class.

12.1.6 Experiment 5 – Recognition of situation by artificial neural network

The neural net part implemented in *Matlab* is stored in the directory *Neural Net1*. The implementation of trained net in Java and its integration with Soar as Java project RPDSoar-MentalSimVer6 in the Eclipse workspace Eclipse1. The weights in the *NeuralNetTfrFn.java* class need to be imported from NeuralNet1 for an agent. The NeuralNet1 contains programs to extract weights from learnt neural net in Matlab to be imported to Java. *Simulation.java* is the main class.

12.2 A simple RPDAgent

All types and versions of the simple *RPDAgent* discussed in Chapter 4 are stored as *RPDTankSimulationVer-**** in the workspace *Eclipse-Java in the attached CD.* Main class is Simulator.java in all the projects.

12.2.1 Verification of one-on-one combat

Exponential and triangular time distribution versions are stored as projects *RPDTankSimulationVer-1.2* and *RPDTankSimulationVer-1.1* respectively.

12.2.2 Verification of three-on-one combat

Verification of three-on-one combat is done on exponential time distribution and is stored as project *RPDTankSimulationVer-2.3*.

12.2.3 Two cases: Red and Blue agents not intelligent, and only Red agents intelligent in three-on-one combat

Three Red tanks are approaching Blue tank in line formation and both sides are not intelligent and the next experiment of only Red agents intelligent are developed in the projects mentioned in the following sentence by modifying the if then conditions of time to engagement for Red tank. If the Red tank hears the Blue tank fire and then the Red tanks out of the firing range of Blue tank manoeuvre and fire on short inter-firing time by calling the method of *timeToNextEngmnt()*. Exponential and triangular time distribution versions are stored as projects *RPDTankSimulationVer-1.5* and *RPDTankSimulationVer-1.6* respectively.

12.2.4 Both Red and Blue agents are intelligent in three-on-one combat

Exponential and triangular time distribution versions are stored as projects *RPDTankSimulationVer-1.7* and *RPDTankSimulationVer-1.8* respectively.

13 APPENDIX D – EXPLANATION OF THE KEY ELEMENTS OF THE CODE OF RPD-SOAR AGENT

Some key elements of the code used in implementing the *RPD-Soar* agent are discussed in this appendix.

13.1 The architecture

The external environment or the world is developed using the Java programming language and the agent is developed using the *Soar* Cognitive architecture. The *Soar* agent and the external environment are interfaced using *Soar* mark-up language (*SML*). Different environments based on maps for different scenarios can be loaded into the system. Agents with different behaviours may be loaded into the system as production rules in *Soar* files. In the *RPD* model it is the experience of the agent that guides its behaviour. As recognition primed decision making is modelled within the *Soar* cognitive architecture, therefore, experiences of the *RPD* model consisting of goals, courses of action, cues, and expectations are transformed into appropriate *Soar* production rules. And these *Soar*-production rules are stored in the agent's *LTM*.

13.2 The interface

The simulation environment is interfaced to the Soar kernel with the help of soar mark-up language (*SML*), as shown in Figure 13.1.

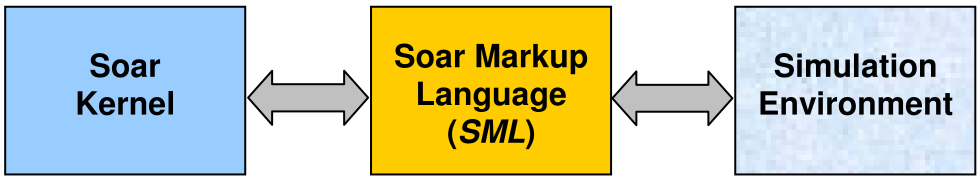


Figure 13.1 The interface

The simulation environment consists of objects or 'entities' as usually called in simulations and some of these entities are *Soar* agents. The *Soar* kernel is capable of developing and maintaining multiple agents and each can have its individual behaviour based on the *Soar* production rules loaded in that agent. *SML* was developed by the *Soar* group to provide an interface into *Soar*. The client can send and receive *Soar* XML packets through a socket maintained by *Soar*, which is port 12121 by default. Client*SML* is available in C++, Java, and Tcl. We have developed the simulation environment in Java and for a client implemented in Java, Java_sml_ClientInterface.dll, SoarKernelSML.dll, and ElementXML.dll dynamically loaded libraries are required.

13.2.1 Creating *Soar* kernel and agents

A *Soar* kernel is created in a new thread using the code shown in Figure 13.2. *Soar* kernel can also be created in the same thread but we do not use this method because we require the *Soar* Kernel to run in a separate thread from that of the environment.
In order to create an agent in this kernel and load productions in the created agent the code shown in Figure 13.3 is used. To facilitate debugging, it also prints any errors that are generated while loading the productions. Multiple agents can be created using the same process by giving each agent a different name. And every agent behaves according to the *Soar* production rules loaded in it.

```
Kernel kernel;
//create Soar kernel
try {
kernel = Kernel.CreateKernelInNewThread("SoarKernelSML");
} catch (Exception e) {
System.out.println("Exception while creating kernel: " + e.getMessage());
          System.exit(1);

if (kernel.HadError()) {
System.out.println("Error creating kernel: " + kernel.GetLastErrorDescription());
                    System.exit(1);
          }
}
```

Figure 13.2 Code to create *Soar* kernel

```
Agent agent;
agent = kernel.CreateAgent("agent name");
boolean load = agent.LoadProductions("File Name.soar");
if (!load || agent.HadError()) {
        throw new IllegalStateException("Error loading     productions:       "       +
agent.GetLastErrorDescription());
}
```

Figure 13.3 Code to create a *Soar* agent

13.2.2 Input - perception

The "input-link" of the *Soar* agent, as explained in the previous chapter, is the link of the agent to receive the information about the outside world. This information is picked up by the agent during the input phase of the next decision cycle. The client needs to acquire the identifier of the input-link in order to give all the information depicting the present situation of the world to the agent. The code to get this input-link identifier and example code of connecting an object from the environment to it, which is put as an identifier on the input-link is shown in Figure 13.4.

```
Identifier input, map;
String MAP = "map";
input = agent.GetInputLink();
map = agent.CreateIdWME(input, MAP);
```

Figure 13.4 Code to get the input-link and create an identifier *WME*

The identifier *WME*s are required when objects need to be created at the input-link, e.g., map in Figure 13.4. String and integer *WME*s are created either directly on the input-link or as part of the object represented by an identifier at the input-link. An example of a *WME* of type 'string' named as 'sound' is created directly on the input-link with the help of code shown in Figure 13.5 with an attribute named 'sound' and its value is a string type constant equal to 'silent'. A *WME* is an identifier, attribute, and value triplet. The value is either a constant or an identifier. The value is an identifier if it is not a terminal node and one or more branches are emanating out of this node. In Figure 13.5, '*bluetank*' is created as an object in the working memory at the input-link representing an entity present in the simulation environment. The object '*bluetank*' has three attributes; two of them are its location in the Cartesian coordinates and third is the direction that the '*bluetank*' is facing. The X and Y coordinates are represented with the *WME*s of type integer and the direction that the tank is facing is represented with a *WME* of type 'string'. All the objects and facts that are required by the agent to reason for situational awareness and decision making are represented in the working memory of the agent through the input-link using codes similar to the ones explained in the above paragraphs but one type named 'Shared Identifier *WME*' and it is discussed in the succeeding paragraphs.

```
Identifier blueTank;
IntElement intElmBlueX, intElmBlueY;
StringElement strElmTkFacing, sound;
String BLUETANK = "bluetank";
String X = "x";
String Y = "y";
String FACING = "facing";

//input is the identifier on input-link
sound = agent.CreateStringWME(input, "sound", "silent");

//input is the identifier on input-link
blueTank = agent.CreateIdWME(input, BLUETANK);

//attribute x and value is location of Blue tank.
intElmBlueX = agent.CreateIntWME(blueTank, X, locOfBlueTkX);

//attribute y and value is location of Blue tank.
intElmBlueY = agent.CreateIntWME(blueTank, Y, locOfBlueTkY);

//attribute facing and value is direction of Blue tank.
strElmTkFacing =agent.CreateStringWME(blueTank,FACING,tkFacing);
```

Figure 13.5 Code to create object identifier, string and integer *WME*s

The environment in this model is grid based. Each cell in the grid is surrounded by its neighbouring cells. Each cell has at least three and at most eight cells as its neighbours. These cells are represented as objects in the working memory of the agent because each cell has two attributes representing its location in Cartesian coordinates. These attributes have integer constant values and can be represented with the help of techniques discussed above. But consider an example of *Cell 5* (Figure 13.6), it has a neighbouring cell just above it *Cell 2*. To represent this environment a *WME* need to be created, which has the identifier of *Cell 5* as its identifier with an attribute *north* and the value being the identifier of the cell in the north the *Cell 2* which itself is another object. This is a case where graph is required instead of a simple tree. In order to develop a graph in working memory of the agent new identifier *WME* with the same value as that of an identifier of an existing object need to be created through '*Create shared identifier WME*' method; the code is shown in Figure 13.6.

Cell 1	Cell 2	Cell 3
Cell 4	Cell 5	Cell 6
Cell 7	Cell 8	Cell 9

```
agent.CreateSharedIdWME(Identifier of Cell 5, ”north”, Identifier of Cell 2);
```

Figure 13.6 Example of shared identifier *WME*

Cell is developed as a class in Java, part of the code is shown in Figure 13.7. The upper part of the code which is a constructor constructs the cells and gives them values for their location in Cartesian coordinates and the content and the lower part of the code which is an example of one of many methods of the same type that connect these cells to each other. The agent instantiates the cell object to create the nine cell graph structure. The code in S_Agent class that instantiates cells and then connects them is shown in Figure 13.8. The upper part of the code creates these cells and gives them the values from the map of the environment for fixed objects in the map and if the objects are dynamic during a simulation then it gives the cells the values separately as the map of the environment does not have these values. These values come when the simulation is fired like in the case of red tank in Figure 13.8. The lower part of the code is a small portion of the code that uses the method in *Cell* class (Figure 13.7) to connect these cells together in a graph.

```
public class Cell {
            .
            .
            .
   public Cell(Agent agent, Identifier map, int xvalue, int yvalue,
String contentvalue) {
            this.agent = agent;
            this.map = map;
            this.xvalue = xvalue;
            this.yvalue = yvalue;
            this.contentvalue = contentvalue;

            Cell cell = agent.CreateIdWME(map, CELL);
            IntElement xIntElm = agent.CreateIntWME(cell, X, xvalue);
            IntElement yIntElm = agent.CreateIntWME(cell, Y, yvalue);
StringElement       contentStrElm = agent.CreateStringWME(cell, CONTENT, contentvalue);
}
            .
            .
            public void setNorthSquare(Cell snorthcell) {
                        if (idNorthCell != null)
                                    agent.DestroyWME(idNorthCell);
                        if(snorthcell == null)
                                    idNorthCell = agent.CreateSharedIdWME(cell, NORTH, null);
                        else {

northcell = snorthcell;
                        idNorthCell = agent.CreateSharedIdWME(cell, NORTH, northcell.cell;
                        }
            }
}
```

Figure 13.7 Part of code for Cell class

```
List cells = new ArrayList();
.
.
int i, j;
   for (i = 0; i < 3; i++)
            for (j = 0; j < 3; j++){
                        if(x[i] == locOfRedTkX && y[j] == locOfRedTkY)
                         cells.add(new Cell(agent, map, x[i], y[j], REDTANK));
                        else
                         cells.add(new Cell(agent, map, x[i], y[j], mapEnv[x[i]][y[j]]));
            } // for loop

            for (i = 0; i < 9; i++) {
                        if(i==0 || i==3 || i==6)
                                    ((Cell)cells.get(i)).setNorthSquare(null);
                        else
(Cell)cells.get(i)).setNorthSquare((Cell)cells.get(i-1));
                        if(i==0 || i==1 || i==2)
                                    ((Cell)cells.get(i)).setWestSquare(null);
                        else
                                    ((Cell)cells.get(i)).setWestSquare((Cell)cells.get(i-3));
                        if(i==2 || i==5 || i==8)
                                    ((Cell)cells.get(i)).setSouthSquare(null);
                        else
                                    ((Cell)cells.get(i)).setSouthSquare((Cell)cells.get(i+1));
                        if(i==6 || i==7 || i==8)
                                    ((Cell)cells.get(i)).setEastSquare(null);
                        else
                                    ((Cell)cells.get(i)).setEastSquare((Cell)cells.get(i+3));   }//for loop
```

Figure 13.8 Part of code in S_Agent class to create cells and connect them

The agent sits in the centre in *Cell 5* in Figure 13.6 and therefore the value of the *content* attribute of this cell is always *bluetank*. This template of nine cells moves over the map and the value of the *x* an *y* attributes representing Cartesian Coordinates of the location of the cell on the map and the value of the *content* giving the name of the object present on the location where the cell is now keep changing accordingly. The method in Cell class that updates these values in the *WME*s are shown in Figure 13.9.

```
public class Cell {
            .
            .
 public boolean setValues(int sxvalue, int syvalue, String scontentvalue) {

     if (sxvalue < 0 || syvalue < 0 || !(scontentvalue == EMPTY ||
                         scontentvalue == OBSTACLE || scontentvalue == ROAD
                         || scontentvalue == RIVER || scontentvalue == REDTANK))
        return false;
     if (xvalue != sxvalue)
        agent.Update(xIntElm, sxvalue);
     xvalue = sxvalue;

     if (yvalue != syvalue)
        agent.Update(yIntElm, syvalue);
     yvalue = syvalue;

     if (contentvalue != scontentvalue)
        agent.Update(contentStrElm, scontentvalue);
     contentvalue = scontentvalue;
     return true;
  }
  .
  .
}
```

Figure 13.9 Method in Cell class to update values in the cell *WME*s

13.2.3 Output – command/action

The command is put at the output-link after the output phase. The code to get command from the agent is shown in Figure 13.10. The method *agent.Commands()* returns true Boolean value if the agent has put any command on the output-link. The method *agent.Command(0)* returns the identifier of the first command and if there are more commands then the sequence needs to continue to 1,2,3,… for the identifiers of other commands. The method *GetCommandName()* on the identifier of command object returns command name as a string object. The method *equals("command name")* on the string object is used to identify the command. The method *GetParameterValue("attribute")* on the identifier of command object gives the value of that *WME* with the attribute that is passed as a parameter. Its representation in working memory is shown in the bottom of Figure 13.10.

```
public boolean executeCommand() {
 if (agent.Commands()) {
  for (int i = 0; i < agent.GetNumberCommands(); ++i){
          Identifier command = agent.GetCommand(i);
   if(command.GetCommandName().equals("move")) {
                  if(command.GetParameterValue("direction").equals("north"))
                               locOfBlueTkY = locOfBlueTkY-1;
                         if(command.GetParameterValue("direction").equals("south"))
                               locOfBlueTkY = locOfBlueTkY+1;
                         if(command.GetParameterValue("direction").equals("east"))
                               locOfBlueTkX = locOfBlueTkX+1;
                         if(command.GetParameterValue("direction").equals("west"))
                               locOfBlueTkX = locOfBlueTkX-1;
   } // if(command....)

   if(command.GetCommandName().equals("turn")) {
                         if(command.GetParameterValue("direction").equals("north"))
                               dir = 'n';
                         if(command.GetParameterValue("direction").equals("south"))
                               dir = 's';
                         if(command.GetParameterValue("direction").equals("east"))
                               dir = 'e';
                         if(command.GetParameterValue("direction").equals("west"))
                               dir = 'w';
   }//if
   command.AddStatusComplete();
   agent.Commit();
   agent.ClearOutputLinkChanges();
  }//for loop
    return true;
 }//if
 else
 return false;
}//executeCommand
```

```
<Identifier of command object> ^direction north
```

Figure 13.10 Output – command

After acquiring all the information from the command at the output-link the command object is augmented with a *WME* with attribute equal to 'status' and value equal to 'complete'. This is done with the help of the method *AddStatusComplete()* on the identifier of the command object. This is a kind of a back door approach of telling the agent that the commands have been picked for implementation by writing status complete on the command at the output link. This information is used by the agent to remove the implemented commands from the output link. But nothing is passed on to the agent until *Commit()* method is used on the command identifier object. After which *ClearOutputLinkChanges()* method on the agent object is implemented. The reason for using this method is because we are using a technique to get the commands in which the changes on the output-link are monitored to pick up a fresh command therefore the output-link changes need to be cleared. Now the method *commands()* on agent object returns true after output-link is changed.

13.2.4 Event handling

In this model event handling is required to update the user interface in the environment and to connect the environment to the 'Java debugger'. The Java debugger can connect to the remote *Soar* kernel given an internet protocol (*IP*) address and a port number. The *IP* address is not required if the *Soar* kernel is running on the same machine. The user interface in this implementation is in the *Simulation* class. The *Soar* kernel is created in *Environment* class. The Simulation class is registered with the kernel through the *Environment* using its *registerForStartStopEvent()* method (Figure 13.11). The *Simulation* object is passed as second argument to *RegisterForSystemEvent()* method in *Kernel* class Figure 13.12. This argument is an object of *SystemEventInterface* type and *Simulation* object matches the type because the S*imulation* is implementing the *EnvironmentListener* interface class (Figure 13.13) and *EnvironmentListener* is extending *SystemEventInterface* class (Figure 13.11).

```
//Class: Environment
public class Environment implements Runnable, Kernel.UpdateEventInterface {
//This allows us to either run the environment directly or from a debugger and get correct behaviour
  int updateCallback = kernel.RegisterForUpdateEvent(
      smlUpdateEventId.smlEVENT_AFTER_ALL_OUTPUT_PHASES, this, null) ;

public void registerForStartStopEvents(EnvironmentListener listener, String methodName) {
              if (kernel != null){
                  int startCallback = kernel.RegisterForSystemEvent(
                           smlSystemEventId.smlEVENT_SYSTEM_START, listener, null) ;
          int stopCallback  = kernel.RegisterForSystemEvent(
                           smlSystemEventId.smlEVENT_SYSTEM_STOP, listener, null) ;
              }
    }
/** This method is called when the "after_all_output_phases" event fires,
* at which point we update the world */
public void updateEventHandler(int eventID, Object data, Kernel kernel, int runFlags){
                try{
                           if (m_StopNow) {
                                        m_StopNow = false ;
kernel.StopAllAgents() ;
                           }//if
                           updateWorld() ;
                }//try
                catch (Throwable t){
                           System.out.println("Caught a throwable event" + t.toString());

                }
              }
}

//Class: EnvironmentListener
public interface EnvironmentListener extends Kernel.SystemEventInterface {

public void tankMoved(Environment env, int x, int y, char dir, int redx, int redy);



              public void atGoalState(Environment env);
} //EnvironmentListener ends
```

Figure 13.11 Code to handle events in Environment class

```
public class Kernel{
  //Class: SystemEventInterface within Kernel
  public interface SystemEventInterface {
public void systemEventHandler(int eventID, Object data, Kernel kernel);
  }
  //Class: UpdateEventInterface within Kernel
  public interface UpdateEventInterface {
public void updateEventHandler(int eventID, Object data, Kernel kernel, int runFlags);
  }
   public  int  RegisterForSystemEvent(smlSystemEventId  id,  SystemEventInterface        handlerObject,  Object
callbackData) {
return smlJNI.Kernel_RegisterForSystemEvent(swigCPtr, id.swigValue(),   this, handlerObject, callbackData);
  }

  public int RegisterForUpdateEvent(smlUpdateEventId id, UpdateEventInterface handlerObject, Object callbackData){
return smlJNI.Kernel_RegisterForUpdateEvent(swigCPtr, id.swigValue(), this, handlerObject, callbackData) ;
  }
} // ends Kernel Class

//Class: smlUpdateEventId
public final class smlUpdateEventId {
public     final     static     smlUpdateEventId     smlEVENT_AFTER_ALL_OUTPUT_PHASES     =     new
smlUpdateEventId("smlEVENT_AFTER_ALL_OUTPUT_PHASES",
smlXMLEventId.smlEVENT_LAST_XML_EVENT.swigValue() + 1);
} // ends Class: smlUpdateEventId

//Class: smlSystemEventId
public final class smlSystemEventId {
public      final      static      smlSystemEventId      smlEVENT_SYSTEM_START      =      new
smlSystemEventId("smlEVENT_SYSTEM_START");
public      final      static      smlSystemEventId      smlEVENT_SYSTEM_STOP      =      new
smlSystemEventId("smlEVENT_SYSTEM_STOP");
}
```

Figure 13.12 Methods and fields in Kernel and other sml classes for event handling

The *Environment* is registered with kernel through *RegisterForUpdateEvent()* method in the *Kernel* class (Figure 13.12). The *Environment* is passed as second argument to the method which is *UpdateEventInterface* class but it matches because the *Environment* is implementing *UpdateEventInterface* class (Figure 13.12). The *UpdateEventInterface* and *smlUpdateEventId* classes are used to update the world and *SystemEventInterface* and *smlSystemEventId* classes are used to update the buttons in the *GUI* (Figure 13.12).

```
public class Simulation implements EnvironmentListener {
Environment env = new Environment(BlueTkX, BlueTkY, RedTkX,fRedTkY,       mapArray);
              env.addEnvironmentListener(this);
              env.registerForStartStopEvents(this, "systemEventHandler") ;

 public void systemEventHandler(int eventID, Object data, Kernel kernel){
          if (eventID == sml.smlSystemEventId.smlEVENT_SYSTEM_START.swigValue()) {
// The callback comes in on Soar's thread and we have to update the //buttons on the UI thread, so
switch threads.
dpy.asyncExec(new Runnable(){
public void run() { updateButtons(true) ; } } ) ;
          }

          if (eventID == sml.smlSystemEventId.smlEVENT_SYSTEM_STOP.swigValue()) {
                    dpy.asyncExec(new Runnable(){
public void run(){ updateButtons(false) ; } } ) ;
          }
 }
}
```

Figure 13.13 Code in Simulation class for event handling

13.3 Graphical user interface (*GUI*)

The code for the *GUI* is in the *Simulation* class. The interface has four buttons Run, Stop, Step, and Reset to control the simulation Figure 13.14. The Run button when pressed runs the agents forever until either the stop button is pressed or the agent achieves its goal. All the buttons are enabled and disabled appropriately. The *GUI* is updated whenever the agent makes a decision to take an action in the world. The simulation and the *GUI* are running in separate threads and therefore the *GUI* is updated independently of the simulation.

13.4 The Environment

The environment is grid based (Figure 13.14). The perimeter has obstacles and the agent's world is restricted to these boundaries. There is a *Map* class which contains the location of obstacle and initial location of the red tank, and is responsible to place the appropriate map for the task. The agent is a tank commander who is commanding a single tank. There are two types of sensors in the tank, one is a visual sensor that looks only one adjacent cell around itself, and the other is a radar sensor that can see up to five cells in the direction that the tank is facing. The radar sensor can not see beyond any obstacle. Past observations from the radar are retained in the memory of the agent and it can use this information in decision making. This environment is more or less common in all the experiments but the changes, if any, are mentioned in the experiments.

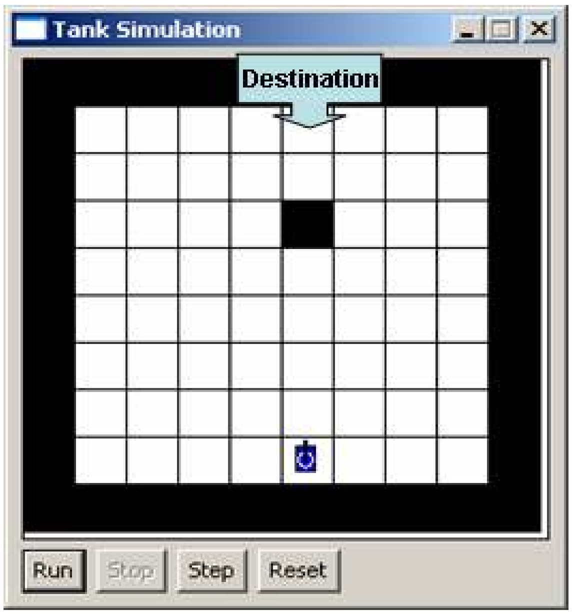


Figure 13.14 The Environment

13.5 Working of *RPD-Soar* agent

The implementation and working of the *RPD-Soar* agent is explained with the help of a vignette. The context is an advance-to-contact military land operation. In a 10 x 10, grid based environment (Figure 13.14), the tank has to start from the south and advance towards north to reach the destination. The environment has only one obstacle which is a hill that gives protection from observation and fire. The agent has radar and visual sensors as described in Section 6.6. The agent has been given the location of the destination cell and has been tasked to advance to that location. Enemy tanks are expected on the route to delay the advance. The firing range of an enemy tank is three kilometres while, that of the agent is four kilometres. In this experiment one cell represents one kilometre. In this thesis the scales for representation of terrain, if required, are mentioned with the experiment.

Most tasks are performed within a larger context that includes higher-level goals. In this case the main context is an advance-to-contact military land operation. There are three high level contexts in this experiment and each is represented with an experience. The experience has goals, cues, expectations, and a course of action. These high level contexts are mutually exclusive and the agent at one time is in any one of them. These experiences are shown in Figure 13.15, Figure 13.16, and Figure 13.17.

Experience: Advance

- **Goal**
  - **Reach the destination**
- **Cues**
  - **High ground: not visible**
  - **Incoming missile: none**
  - **Enemy tank: none visible**
  - **Distance to the destination**
- **Expectations**
  - **No incoming missile**
  - **No enemy tank visible**
  - **No high ground within four kilometres**
- **Course of Action**
  - **Move towards destination**

Figure 13.15 Experience – advance

The *goal* is the state of affairs that is intended to be achieved and may also be defined as the end state to which all efforts are directed. The *cue* is the perception of a set of patterns that gives the dynamics of the situation, and making distinctions in these patterns. This pattern is formed by the features of a situation or elements in an environment. The *expectation* is the belief of the agent that an event will or will not occur in a given situation. The *course of action* is the strategy or plan that the agent intends to implement.

Recognition of a situation not only means recognizing a typical response but also indicating what goals make sense, what cues are important and what is expected next. During advance an important cue is *high ground*. The agent expects to see no high ground within four kilometres of it. Now if the agent finds high ground within four kilometres then this expectation is violated and a fresh evaluation of the situation is necessary. If the agent finds high ground within four kilometres of itself and is facing north, which is the direction of its destination, then it recognizes this situation and changes its state to *manoeuvre*. During *manoeuvre* the agent does not expect to see an enemy tank. If it sees a tank an expectation is violated and the situation is evaluated again.

Experience: Manoeuvre

- **Goals**
  - **Expose the enemy tank at the longest range**
  - **Do not expose own tank to enemy within enemy tank's firing range**
- **Cues**
  - **High ground: at a distance <= 4 kilometres**
  - **Direction of own tank: facing destination (north)**
  - **Incoming missile: none**
  - **Enemy tank: none visible**
- **Expectations**
  - **No incoming missile**
  - **No enemy tank visible**
  - **Enemy tank behind high ground on completion of manoeuvre**
- **Course of Action**
  - **While taking cover from the high ground, move to a location four kilometres east of expected enemy tank**

Figure 13.16 Experience – manoeuvre

Experience: Attack

- **Goal**
  – **Destroy the enemy**
- **Cues**
  – **Enemy tank: visible**
- **Expectations**
  – **Enemy tank remains visible**
- **Course of Action**
  – **Engage the enemy tank with fire**

Figure 13.17 Experience - attack

If we set up the simulation with the map representing the environment displayed in Figure 13.14, load the agent with the behaviour required to accomplish the mission for *advance-to-contact* operation, connect it with *Soar* Java debugger and then run it for a single step then the agent will start to develop working memory contents as shown in Figure 13.18. Running the simulation one step also makes the agent run through one decision cycle. The information generated by the radar and the visual sensors is put in the working memory through the input-link of the agent. This information is shown in Figure 13.18. The agent is facing north and is five cells south of the high ground therefore the radar sensor of the agent sees an obstacle at location represented in Cartesian coordinates as (5, 3). This information is represented in working memory as *(S1 ^io I1) (I1 ^input-link I2) (I2 ^radar R1) (R1 ^obstacle O1) (O1 ^x 5 ^y 3).* The visual sensor as we know can see only one cell around itself and therefore, sees three obstacles in the south, south-west, and south-east of the agent represented in the working memory as *(M1 ^cell C9 ^cell C8 ^cell C7 ^cell C6 ^cell C5 ^cell C4 ^cell C3 ^cell C2 ^cell C1) (C9 ^content obstacle) (C6 ^content obstacle) (C3 ^content obstacle).* The rest of the five cells around the agent are empty and are displaying their contents as empty in the working memory.

```
(S1 ^bluetank B1 ^directions E15 ^directions N1 ^directions W1 ^directions S2
    ^io I1 ^map M1 ^operator O2 + ^operator O2 ^radar R1 ^super-state nil
    ^super-state-set nil ^top-state S1 ^type state)
 (I1 ^input-link I2 ^output-link I3)
   (I3)
  (I2 ^bluetank B1 ^incoming no ^map M1 ^radar R1 ^sound silent)
   (B1 ^facing north ^x 5 ^y 8)
   (R1 ^empty E14 ^empty E13 ^empty E2 ^empty E12 ^empty E11 ^empty E10
        ^empty E9 ^empty E8 ^empty E7 ^empty E6 ^empty E5 ^empty E4
        ^empty E3 ^empty E1 ^obstacle O1)
    (E14 ^x 6 ^y 3) (E13 ^x 4 ^y 3) (O1 ^x 5 ^y 3) (E12 ^x 6 ^y 4)
    (E11 ^x 4 ^y 4) (E10 ^x 5 ^y 4) (E9 ^x 6 ^y 5) (E8 ^x 4 ^y 5)
    (E7 ^x 5 ^y 5)  (E6 ^x 6 ^y 6)  (E5 ^x 4 ^y 6) (E4 ^x 5 ^y 6)
    (E3 ^x 6 ^y 7)  (E2 ^x 4 ^y 7)  (E1 ^x 5 ^y 7)
(M1 ^cell C9 ^cell C8            ^cell C7 ^cell C6 ^cell C5 ^cell C4 ^cell C3 ^cell C2 ^cell C1)
    (C9 ^content obstacle ^north C8 ^north-west C5 ^west C6 ^x 6 ^y 9)
    (C5 ^content empty ^east C8 ^north C4 ^north-east C7 ^north-west C1
          ^south C6 ^south-east C9 ^south-west C3 ^west C2 ^x 5 ^y 8)
    (C6 ^content obstacle ^east C9 ^north C5 ^north-east C8
          ^north-west C2 ^west C3 ^x 5 ^y 9)
    (C8 ^content empty ^north C7 ^north-west C4 ^south C9 ^south-west C6
          ^west C5 ^x 6 ^y 8)
    (C7 ^content empty ^south C8 ^south-west C5 ^west C4 ^x 6 ^y 7)
    (C4 ^content empty ^east C7 ^south C5 ^south-east C8 ^south-west C2
          ^west C1 ^x 5 ^y 7)
    (C3 ^content obstacle ^east C6 ^north C2 ^north-east C5 ^x 4 ^y 9)
    (C2 ^content empty ^east C5 ^north C1 ^north-east C4 ^south C3
          ^south-east C6 ^x 4 ^y 8)
    (C1 ^content empty ^east C4 ^south C2 ^south-east C5 ^x 4 ^y 7)
 (N1 ^opposite south ^value north) (E15 ^opposite west ^value east)
 (S2 ^opposite north ^value south) (W1 ^opposite east ^value west)
 (O2 ^name initialize-rpd-soar)
```

Figure 13.18 Working memory of the *RPD-Soar* agent

The *bluetank, map, cell, radar, obstacle,* and *empty* are objects in the working memory that have been put there through the input-link by the environment. *Operator* and *direction* objects are produced by the production rules loaded in the agent. The *state* object is automatically created in the working memory of the agent. The production rule *propose*initialize-rpd-soar* (Figure 13.19) checks for a task for the agent by checking the absence of *name* of the state and proposes an operator named *initialize-rpd-soar*. This being the only operator proposed is selected in the decision phase and is applied in the application phase by the rule *apply*initialize-rpd-soar* (lower half of Figure 13.19). Firing of this rule places the mission of this *advance-to-contact* operation as the desired state in the working memory of the agent.

The simulation is run through the next step and conditions based on the cues of experience for *advance* (Figure 13.15) as the suitable course of action is selected. There is no red tank in sight, the obstacle is five kilometres away, and there is no incoming missile. The presence of red tank and incoming missile are straight forward cues but in order to observe the cue of relative distance of tank to the obstacle some elaborations is required which is Level 2 *RPD* and is done with the help of productions in Figure 13.20. The *advance* course of action is an abstract operator. Therefore an operator no-change impasse occurred and a new *sub-state* is created to implement it.

```
sp {propose*initialize-rpd-soar
  (state <s> ^super-state nil
           -^name)
-->
  (<s> ^operator <o> +)
  (<o> ^name initialize-rpd-soar)
}

sp {apply*initialize-rpd-soar
  (state <s> ^operator <op>)
  (<op> ^name initialize-rpd-soar)
-->
  (<s> ^name rpd-soar
       ^desired <d>)
  (<d> ^bluetank <btk>)
  (<btk> ^x 5 ^y 1)
}
```

Figure 13.19 Production: initialize-rpd-soar

```
sp {elaborations*elaborate*state*dist-obs-tank
  (state <s> ^operator.name initialize-rpd-soar)
-->
  (<s> ^dist-obs-tank <dot>)
}

sp {elaborations*elaborate*state*dist-obs-tank*coords
  (state <s> ^name rpd-soar
           ^bluetank <bt>
           ^radar.obstacle <obs>
           ^dist-obs-tank <dot>)
  (<bt> ^x <btx> ^y <bty>)
  (<obs> ^x <ox> > 0 < 9   ^y <oy> > 0 < 9)
-->
  (<dot> ^x (- <btx> <ox>)   ^y (- <bty> <oy>) )
}
```

Figure 13.20 Productions: elaborate distance of tank to obstacle

In this context with *advance* as its major task the agent has four actions to choose from: move in the direction that the tank is facing and turn in the other three directions. And as all four are applicable in the situation then an operator tie impasse is generated (Figure 13.21). This is the situation of *RPD* model where the decision maker can not select a course of action from a pool of courses of action that he knows can apply. Now the decision maker develops a mental model of the environment and mentally simulates the courses of actions serially to select the one which seems satisficing.

```
: ==>S: S1
   :   O: O3 (advance)
   :   ==>S: S3 (operator no-change)
   :      ==>S: S5 (operator tie)
```

Figure 13.21 Operator tie impasse

Now among the candidate operators in experiments discussed later, the agent has the experience to prefer one operator over the other for evaluation and the experience to judge when an operator is satisficing but this agent evaluates each and every candidate serially and randomly selects one to evaluate first. Therefore, one operator is selected in the selection space *S5* for evaluation at random, shown in Figure 13.22.

```
: ==>S: S1
   :    O: O3 (advance)
   :    ==>S: S3 (operator no-change)
```

Figure 13.22 Operator: evaluate-operator

This operator named *evaluate-operator* is also abstract and therefore another space *S7* is created to implement evaluation and this is the mental model for simulating a course of action as of *RPD* model (Figure 13.23). In this space, all the objects in the environment are modelled again and the operator representing the course of action to be evaluated is selected to be applied.

```
: ==>S: S1
   :    O: O3 (advance)
   :    ==>S: S3 (operator no-change)
   :       ==>S: S5 (operator tie)
   :          O: O10 (evaluate-operator)
   :          ==>S: S7 (operator no-change)
   :             O: C10 (turn)
```

Figure 13.23 Space for mental simulation

The operator application is not on the real world rather on the model world created in the agent's head. In this case the course of action is being evaluated for advance which means a better action is the one that can take the agent close to the destination given in the original mission. In order to evaluate the candidate actions, the Manhattan distance is calculated after applying each action and the numeric value is recorded as evaluation factor. Manhattan distance between two points (x1, y1) and (x2, y2) is defined in terms of X and Y as X = x2 - x1, and Y = y2 - y1. And then the action with the least numeric value is selected. This is achieved through the use of *selection space* implementation provided by *Soar* group (Laird, 2006a) and the production rules written for copying the objects and the application of operators in the mental model for this implementation. Some of the production rules written for this purpose are shown in Figure 13.24.

The majority of the production rules provided as *selection space* productions are being used as such in this implementation for mental simulation while some of them are modified to suite the requirements of this model. The first rule in Figure 13.24 is used by the production rules of *selection space* for copying objects in the mental model; the second rule is used to calculate the Manhattan distance; and the third rule applies a north move operator for mental simulation after checking the absence of *io* object which is an indicator that this is the mental model and not the real world.

```
1           sp {advance*elaborate*problem-space
            (state <s> ^name advance)
            -->
            (<s> ^problem-space <p>)
            (<p> ^name advance ^default-state-copy yes
                        ^two-level-attributes bluetank)
            }


2           sp {advance*elaborate*state*manhattan-distance
            (state <s> ^name advance
             ^desired <d>    ^bluetank <bt>   ^tried-tied-operator)
            (<d> ^bluetank <dbt>)
            (<bt> ^x <bx> ^y <by>)
            (<dbt> ^x <dbx> ^y <dby>)
            -->
            (<s> ^mhdistance <mhd>)
            (<mhd> ^mhx ( abs ( - <dbx> <bx>))   ^mhy ( abs ( - <dby> <by>)))
            }
```

Figure 13.24 Example productions used to implement mental simulation

After evaluating each action the *sub-states* of the mental model and thus all the *WMEs* related to them are removed from the working memory of the agent and only the evaluated value is kept in the higher state evaluating these actions.

After evaluating all the candidate actions *move north* operator is selected because it is taking the agent close to the destination and is applied to the real world. It is done through the output-link and with the help of the model for acquisition of commands from the agent explained earlier in the same chapter. The new location of the *Blue* agent in the environment after moving north is shown in Figure 13.25.

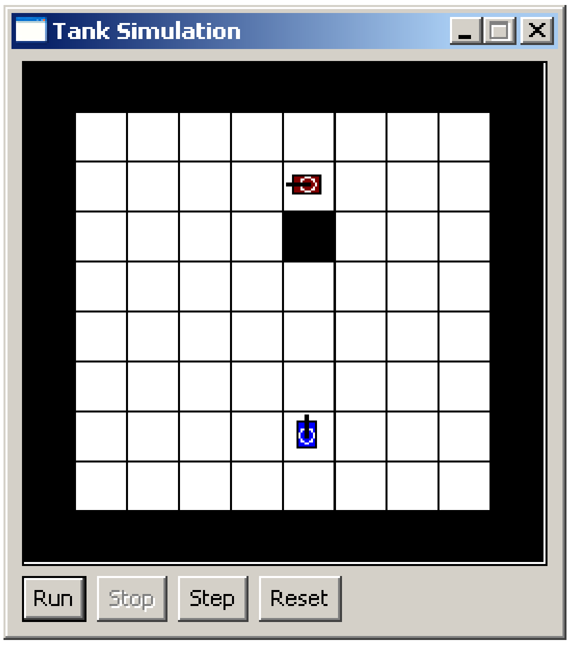


Figure 13.25 Situation after moving north

Now the distance to the high ground is equal to four kilometres and one of the expectations of the advance experience is not met, therefore the situation is re-evaluated and this time the experience manoeuvre is recognised as its conditions are met. The course of action for the experience manoeuvre is represented graphically in Figure 13.26. In this case the blue agent sees high ground on its approach to its destination and expects an enemy tank behind it. Similar approach has been adopted by Tambe and Rosenbloom (1995) where the pilot agent observes the actions of the enemy aircrafts and by observing the observable actions infers their unobserved actions, plans, goals, and behaviours.

The course of action manoeuvre is also at higher level of abstraction and creates an operator no-change impasse (Figure 13.27).

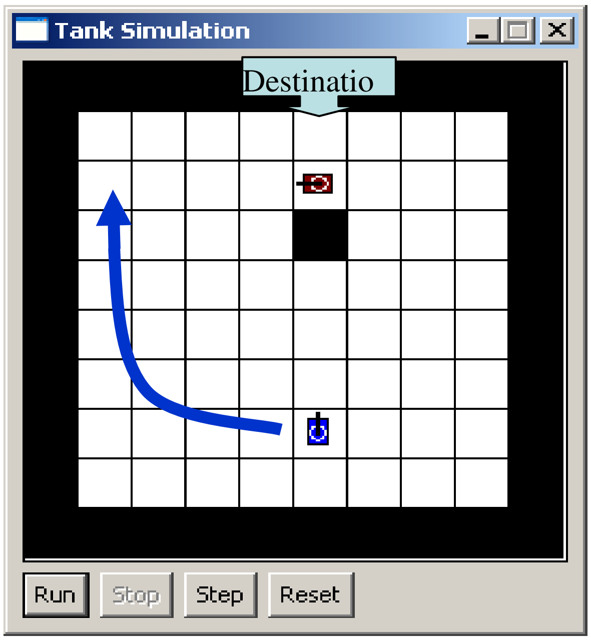


Figure 13.26 Experience – manoeuvre

Just like advance, this course of action for experience manoeuvre is implemented through atomic actions of move and turn but now the destination is the location pointed by the head of the arrow representing the planned path for movement of blue tank. This desired state set as the goal of experience manoeuvre is set by the production rule shown in Figure 13.28.

```
: ==>S: S1
   :    O: O12 (mnvr)
```

Figure 13.27 Manoeuvre - an abstract action

This location as destination for completing the manoeuvre action is kept so as the Blue tank appears at a distance of four kilometres from the Red tank and therefore is out of the firing range of the enemy while the Red tank is within the firing range of Blue tank. The Blue tank commander is exploiting the weakness of the enemy to achieve own aim of destroying the enemy forces as secondary mission while reaching the destination which is the main mission. In this situation it would have not been possible for the Blue tank to reach its destination without destroying the Red tank or making it retreat from its present location as the area would have been unsafe to advance.

```
sp {mnvr*initialize*desired*state
   (state <s> ^name mnvr    ^radar.obstacle <obs>)
   (<obs> ^x <ox> > 0 < 9   ^y <oy> > 0 < 9)
-->
   (<s> ^desired <d>)
   (<d> ^bluetank <btk>    ^better lower)
   (<btk> ^x (- <ox> 4)      ^y (- <oy> 1)       ^facing east)
}
```

Figure 13.28 Production: set the goal for manoeuvre

The selection of the atomic actions in experience manoeuvre is through mental simulation as is the case of experience advance. It is not necessary for all the experiences to have all the components of situations as represented in the *RPD* model. It is understandable that the recognition of a situation requires more processing of information for comparatively high level contexts; therefore, it is expensive in time and resources to repeat the process with every single change in the world. It is also true that not all changes in the world are likely to change the higher context. It is also observed that the behaviours at a higher level persist for a comparatively longer time and consist of a combination of low level behaviours. There may not be a requirement to associate expectations with the courses of action in the experiences at atomic level behaviours where an action is taken that changes the world and then the situation is re-evaluated to select the next action. This is because the selected course of action does not persist long enough to require watching expectations while the action is under progress. The same is true for the goal at atomic level. The goal is the result of the action itself. Therefore, in this implementation of the *RPD* model, the goals and expectations are part of the experiences representing behaviour at a higher level of abstraction. At atomic level the experiences consist of only cues and the action. The success value or preference of one action over the other accompanies the experiences at even atomic level in most cases. This success value is used in two ways: the first, is the selection of a course of action straight away without mentally simulating it if one candidate is distinctly better than the others; and the second, is the selection of a course of action as the first one to consider for mental simulation when the chances of success of candidate courses of action are similar.

In *Soar*, it is effortless to model the phenomenon of watching the expectations while carrying out a course of action. In *Soar*, all the states are active at all times. Any change in a state at a higher level removes all the *sub-states* which are responsible for the creation of these *sub-states.* In the vignette under discussion (see Figure 13.14), the advance behaviour is selected and the course of action is under progress when the blue tank moves north and the distance between the blue tank and the obstacle reduces to four kilometres (Figure 13.25). The agent is expecting no obstacle this close while advancing thus an expectation is violated and the situation needs to be re-evaluated. In *Soar*, the re-evaluation of a situation given the violation of expectations is almost automatic if the conditions for selection of the concerned operators are set correctly. The abstract advance operator that creates the *sub-state* where this course of action is being implemented is removed due to one of its conditions for selection being violated and thus the *sub-states* implementing it are also removed. The situation therefore is re-evaluated to recognize new situations in order to find courses of action from other experiences to proceed with the task.

During the manoeuvre context the blue tank keeps moving by selecting actions that reduce its distance from the destination recognized as a goal with the present situation until it reaches the destination. To accomplish its goal completely the blue tank also turns east as shown in Figure 13.29. Now the blue agent finds the red tank on its radar sensor (Figure 13.30). The only cue in the *attack experience* is red tank (Figure 13.17) and for its selection the condition to be satisfied is red tank's presence. As the condition is met therefore the proposal to select attack as a context is fired by the production rule shown in Figure 13.31 and as attack is the only operator proposed therefore it is selected. Attack is an action at a higher level of abstraction therefore a new *sub-state* is created through an operator no-change impasse (Figure 13.32) to implement this abstract action. In this context a fire action is proposed, selected and applied and the red tank is destroyed.

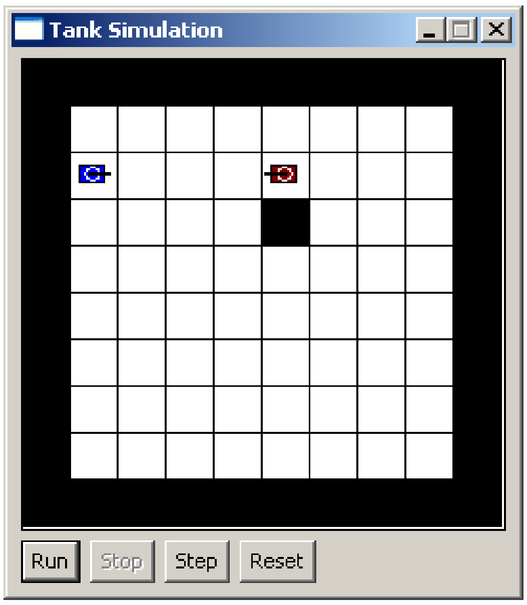


Figure 13.29 Situation after completing manoeuvre

```
(S1 ^bluetank B1 ^desired D2 ^directions E15 ^directions W1 ^directions S2
    ^directions N1 ^dist-obs-tank D1 ^io I1 ^map M1 ^mnvr-situation yes
    ^name rpd-soar ^operator O215 + ^operator O215 ^radar R1
    ^super-state nil ^super-state-set nil ^top-state S1 ^type state)

    (R1 ^empty E84 ^empty E83 ^empty E82 ^empty E1 ^obstacle O61
        ^obstacle O45 ^obstacle O44 ^obstacle O43 ^obstacle O62
        ^obstacle O71 ^obstacle O72 ^obstacle O73 ^obstacle O120
        ^obstacle O1 ^obstacle O121 ^obstacle O122 ^obstacle O123
        ^obstacle O124 ^obstacle O125 ^obstacle O140 ^obstacle O141
        ^obstacle O142 ^obstacle O143 ^obstacle O144 ^obstacle O60
        ^redtank R2)
```

Figure 13.30 State of working memory showing red tank on radar sensor

```
sp {rpd-soar*propose*attack
   (state <s> ^name rpd-soar   ^radar.redtank )
-->
   (<s> ^operator <op> + =)
   (<op> ^name attack)
}
```

Figure 13.31 Production - propose attack

```
: ==>S: S1
   :    O: O215 (attack)
```

Figure 13.32 Attack – an abstract action

The *attack experience* expects to see the red tank all the time but as the simulation removes the destroyed tank it is not visible on the radar sensor. The expectation of the situation is violated in the *RPD* model and situation is required to be re-evaluated and in *Soar* it is implemented by putting it as a condition in the production that proposes attack operator as shown in Figure 13.31. As the conditions for the proposal of the *attack* operator are not satisfied therefore attack operator is removed and so is the *sub-state* created because of it.

The situation is re-evaluated and advance is selected which as discussed earlier is an abstract operator and creates an operator no-change impasse to create a *sub-state* to implement it Figure 13.33.

```
: ==>S: S1
   :    O: O217 (advance)
```

Figure 13.33 Advance – an abstract action

The agent, repeating *move* and *turn* actions after selecting them by evaluating through mental simulation reaches its destination shown in Figure 13.34.

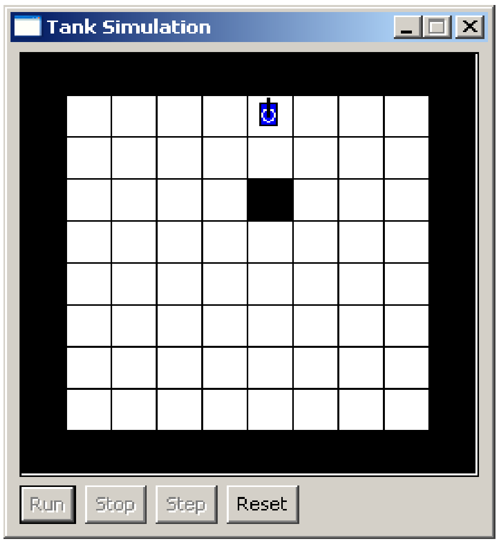


Figure 13.34 Blue tank reaches its destination

On completing the mission as in military operations and reaching the goal state as in *Soar*, the agent needs to halt and the simulation is required to either stop or reset for another run. In case only the agent needs to be stopped, it may be done with the help of *Soar* production rules and the method to implement this option is discussed later, but if the simulation needs to be stopped and reset for another run then it may be done with the help of the code shown in Figure 13.35 and Figure 13.36.

```
public class Environment implements Runnable,
 Kernel.UpdateEventInterface {
.
.
public void updateWorld() {
if(blueTk.executeCommand()) {
                    .
                    .
                              firetankMoved();
                              if (isAtGoalState())
                              fireAtGoalState();
}//if(blueTk.exec....)
 }//updateWorld()
public boolean isAtGoalState() {
                    return (locOfBlueTkX == 5 && locOfBlueTkY == 1);
}
* Notifies any registered listeners that this <code>Environment</code>
   * has reached the goal state. */
          protected void fireAtGoalState() {
    Iterator i = listeners.iterator();
    while (i.hasNext())
      ((EnvironmentListener)i.next()).atGoalState(this);
  }
* Runs Soar until interrupted
          */
          public void run() {
                    if (isAtGoalState())
                              return;
                    m_StopNow = false;

                    // Start a run
                    kernel.RunAllAgentsForever();
          }
          public void step() {
                    if (isAtGoalState())
                              return;
                    // Run one decision
                    kernel.RunAllAgents(1);
          }
/** Stop a run (might have been started here in the environment or in the debugger)*/
          public void stop() {
                    // issue StopSoar() in a callback.
                    m_StopNow = true;
          }
```

Figure 13.35 Environment – methods to handle goal state

```
public class Simulation implements PaintListener,
ControlListener, EnvironmentListener {
.
.
public void atGoalState(Environment env) {
                    env.stop();
                    System.out.println("Goal State reached.");
}
public void systemEventHandler(int eventID, Object
data, Kernel kernel) {
if (eventID == sml.smlSystemEventId.smlEVENT_SYSTEM_START.swigValue()) {
// The callback comes in on Soar's thread and we have to //update the buttons on the UI thread, so switch
threads.
dpy.asyncExec(new Runnable() {
public void run() { updateButtons(true) ; } } ) ;
                    }

if (eventID == sml.smlSystemEventId.smlEVENT_SYSTEM_STOP.swigValue()) {
                    simRun++;
                              if(simRun<totalSimRuns) {
                                        env.reset();
                                        env.run();
                    }
// The callback comes in on Soar's thread and we have to //update the buttons on the UI thread, so switch
threads.
                    dpy.asyncExec(new Runnable() {
public void run() { updateButtons(false) ; } } ) ;
}
}
}
```

Figure 13.36 Simulation – method to handle goal state

If agent has some command to execute, it is executed and then it is checked with the help of the method *isAtGoalState()* whether the goal state is reached or not. If the goal state is reached then the *Environment* with the help of the protected method *fireAtGoalState()* fires the *atGoalState()* method in all the registered listeners (Figure 13.35). The *Simulation* with the help of its *atGoalState()* method calls the *Stop()* method in *Environment* which sets the Boolean variable *m_StopNow* true. The update event handling method in *Environment*, shown in Figure 13.11, checks for *m_StopNow* variable and if true the agents are stopped. When the agent stops the *systemEventHandler()* method in *Simulation* is evoked which resets the environment and runs the simulation again until the number of simulations required is reached (Figure 13.36).

The agent can be stopped using halt command inherent in *Soar*. The halt command irreversibly terminates the execution of the *Soar* program and should not be used when the agent needs to be restarted. The production rules that implement this method of stopping the execution of *Soar* program is shown in Figure 13.37. But this has not been used in most cases in this implementation because the *Soar* program is run within a simulation which needs to be restarted most of the times.

```
# This production sets the goal for the agent.

sp {rpd*apply*initialize-rpdsoar-ms1
  (state <s> ^operator <op>)
  (<op> ^name initialize-rpdsoar-ms1)
-->
  (<s> ^name rpdsoar-ms1
       ^desired <d>)
  (<d> ^bluetank <btk>)
  (<btk> ^x 5 ^y 1)
}

# This production tests the goal state and
# halts the execution of Soar program when
# the goal is achieved.

sp {rpd*detect*desired-state*reached
 (state <s> ^name rpdsoar-ms1
           ^io
           ^desired <d>
           ^bluetank <bt>)
 (<d> ^bluetank <dbt>)
 (<dbt> ^x <x> ^y <y>)
 (<bt> ^x <x> ^y <y>)

-->
  (write (crlf) |Success!|)
  (halt)
}
```

Figure 13.37 Productions: to set the goal, test the goal and halt the agent

14 APPENDIX E – LEARNT CHUNKS

The chunks learnt by the *RPD-Soar* agent in Experiment 4 are stored in the file named *learnt chunks* in the attached CD.

15 APPENDIX F – PUBLICATIONS

The following papers have been published based on the work in this thesis:

- Raza, M. & Sastry, V. V. S. S. (2007) Command Agents with Human-Like Decision Making Strategies. *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence - (ICTAI 2007),* Vol. 2, pp. 71-74. IEEE Computer Society.

- Raza, M. & Sastry, V. V. S. S. (2008) Variability in Behavior of Command Agents with Human-Like Decision Making Strategies. *Tenth International Conference on Computer Modeling and Simulation (uksim 2008),* pp. 562-567. Cambridge, England.