2011

Enhancing automated red teaming with Monte Carlo Tree Search

Daniel Beard
Edith Cowan University

Edith Cowan University

Copyright Warning

Enhancing Automated Red Teaming with Monte Carlo Tree Search

Bachelor of Computer Science Honours Thesis

By: Daniel Beard
Student ID: 10045549

Faculty of Computing, Health and Science
Edith Cowan University

Supervisor(s): Associate Professor Philip Hingston
Dr Martin Masek

Date of submission: November 2011

Abstract

This study has investigated novel Automated Red Teaming methods that support replanning. Traditional Automated Red Teaming (ART) approaches usually use evolutionary computing methods for evolving plans using simulations. A drawback of this method is the inability to change a team's strategy part way through a simulation. This study focussed on a Monte-Carlo Tree Search (MCTS) method in an ART environment that supports re-planning to lead to better strategy decisions and a higher average score.

Table of Contents

1. Introduction

The term “red teaming” refers to playing or thinking like an opponent or adversary. “Red teams are established by an enterprise to challenge aspects of that very enterprise’s plans, programs, assumptions” (Schneider, 2003). Red teaming has practical applications in many fields including network security, business, and military. Red teaming usually consists of two teams, the red team that is challenging a plan or organisation and the blue team, the team or organisation that is being challenged.

There are generally three different types of red teaming:

- Surrogate adversaries and competitors of the enterprise
- Devil’s Advocates
- Sources of judgment independent of the enterprise’s “normal” processes (Schneider, 2003)

Surrogate adversaries generally try to expose weaknesses and vulnerabilities in an organisation as an attempt to understand how a real adversary may use these weaknesses to their advantage. An outcome from this process is a list of potential responses to an adversary. “The setting could be a military training, experimentation or gaming environment where the red team plays the ‘Opposing Force’, using the adversary’s presumed tactics and equipage” (Schneider, 2003). Surrogate adversaries and competitors of the enterprise are the most common type of red teaming used in simulation and will be the focus of this study.

A devil’s advocate approach is used to challenge an assumption that an organisation has that the organisation cannot afford to be wrong about. “By deliberately challenging the organisation’s own plans, programs and assumptions, Red Teaming can identify strengths, weaknesses, opportunities and threats that were not considered or given proper review” (Nettles, 2010)

Schneider (2003) describes the sources of independent judgement method as an analysis from “team members with experience from positions at higher levels in industry or government” (Schneider, 2003). This method has been criticised as “The objective is often to be a sounding board and ‘kitchen cabinet’ for the sponsor” (Schneider, 2003) and “Even the most talented group of planners and thinkers cannot identify their own oversights and sometimes are unable to see the overall big picture” (Nettles, 2010).

Computational, or Automated Red Teaming (ART) is a natural progression from red teaming that involves using computer simulated environments and involves executing many simulations over a short period of time. Automated red teaming is usually used in military applications to find strategic weaknesses and shape battlefield strategies. Automated red teaming usually uses an evolutionary algorithm to tune certain parameters.

Evolutionary algorithms define a fitness function, which measures the effectiveness of a simulation after execution is complete. Given that the simulation has to run to completion before it can be evaluated, the teams cannot change their strategies part of the way through a simulation and makes implementing re-planning difficult with an evolutionary based algorithm. Rather than having to rely on a heuristic evaluation function to evaluate how well each team is performing, which can often be very difficult to implement for non-terminal game states (G Chaslot, Bakkes, Szita, & Spronck, 2008)an alternative method is explored in this project that support re-planning.

1.1. The background to the study

Red teaming simulations involve complex interactions between an environment and sets of agents, which are well suited for agent-based modelling. Agent based modelling typically consists of three parts.

1. A set of agents, their attributes and behaviours
2. A set of agent relationships and methods of interaction:
   An underlying topology of connectedness defines how and with whom agents interact.
3. The agents' environment: Agents interact with their environment in addition to other agents. (Macal & North, 2010)

A red teaming scenario is similar to a game in several aspects. Each side (player) has strategies and tactics, the scenario has certain rules that must be adhered to, each side is trying to defeat an opponent, and teams have scores.

Therefore in analysing a red teaming scenario, we can use tools such as:

- Game theory
  - Linear programming
- Artificial intelligence methods designed for use in games
  - Game tree search
  - Evolutionary algorithms to search for strategies

1.1.1. Game Theory

A branch of mathematics that is concerned with studying and analysing games is known as game theory. A game is "a situation in which several individuals have choices to make" (Cave, 1987). Generally, the study of games has two objectives: "the descriptive goal of understanding why the parties ('players') in competitive situations behave as they do" and "being able to advise the players of the game the best way to play" (Morris, 1994).

The scenarios in the proposed study are two-person zero-sum games. A two-person zero-sum game is "one in which the two players have precisely opposite preferences. It is, therefore a game in which cooperation and collusion can be of no value. Any improvement for one player necessitates a corresponding loss for the other" (Luce & Raiffa, 1989). An example of a two-person zero sum game is tic-tac-toe.

One method commonly used in game theory of representing the payoffs for a two-person zero sum game is a payoff matrix. E.g. Matrix A = (a_{ij}) could contain the final scores of the red team given a set of strategy choices for each team. The rows are player 1's strategies (the red team) the columns are player 2's strategies (the blue team). "The rows and columns of A are called pure (or deterministic) strategies that are available for the two players to choose" (Zafra, 2010).

Solving a payoff matrix with linear programming gives values known as the Nash Equilibrium. The Nash Equilibrium is where "no team has any incentive to deviate [from their chosen strategies]" (Sailer, et al., 2007). Finding the Nash equilibrium gives us the ratio of strategies each team should choose to get the maximum payoff. E.g. Red team should play strategy 1 100% of the time, while the Blue team should play strategy 1 38% of the time and strategy 2 62% of the time.

1.1.2. Game Tree Search

A common method used for representing turn-based games is the game tree. “Each node in the tree represents a board position, and each branch represents one possible move. Each player gets to move at alternating levels of the tree” (Millington & Funge, 2009).

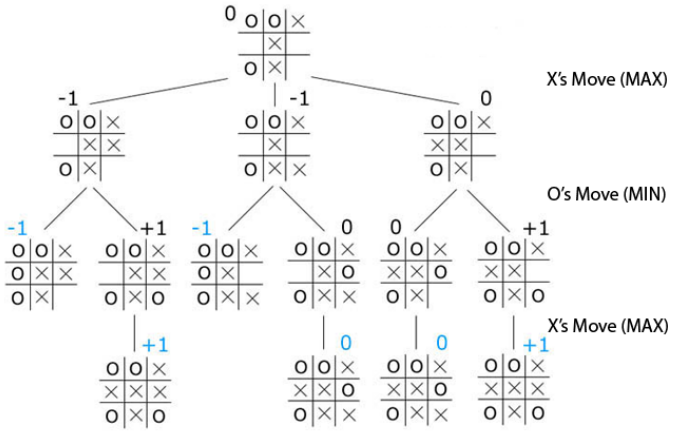


Figure 1. Minimax Tree. Adapted from (Lin, 2003)

An algorithm that is commonly used for artificial intelligence in board games such as chess, checkers and tic-tac-toe is known as minimax. A simple example of a minimax tree from a tic-tac-toe game is shown in Figure 1. “The minimax algorithm is an algorithm for finding an optimal strategy in a certain game state for deterministic two-player zero-sum games” (Kleij, 2010). Minimax assigns a score to each node of the tree through the use of a heuristic evaluation function. When choosing a move to make, an assumption is made that the opposing team will choose the move that puts the current team in the worst possible position (lowest score). “We are maximising our score, while our opponent is minimising our score. This changing between maximising and minimising, as we search the game tree is called minimaxing” (Millington & Funge, 2009).

The minimax algorithm only works for deterministic, turn-based two-person zero-sum games. When games have an element of randomness or players can make simultaneous moves, minimax is not effective. A known problem is that “in several games building an evaluation function based on heuristic knowledge for a non terminal position is a difficult and time-consuming issue; the most notorious example is the game of Go” (G. Chaslot, Winands, Uiterwijk, Herik, & Bouzy, 2007).

1.1.3. Monte-Carlo Tree Search

An alternative to minimax is the Monte-Carlo Tree Search algorithm. Monte Carlo Tree Search (MCTS) is based on the Monte Carlo method, which relates to random sampling of methods that have some randomness or uncertainty and is used in the fields of artificial intelligence, statistics, and simulating physical systems. Monte Carlo is "a method of using repeated random sampling to estimate the solutions to problems that are very hard or impossible to find analytically" (Kleij, 2010).

Monte-Carlo Tree Search does not use a heuristic evaluation function when scoring nodes in a tree. Instead, random iterated sampling of choices occurs and each choice is executed to the end of the simulation. A node's score represents the best possible outcome from that game position. As shown in Figure 2, MCTS has four steps: selection, expansion, simulation and back-propagation.

Selection: Navigate down the tree through the child nodes, starting at the root node using a combination of exploitation and exploration. Exploitation is where the "higher scoring" nodes are most focused on. Exploration is the processes of investigating nodes that don't score particularly well, but might offer an advantage at a lower depth. A balance must be made between exploration and exploitation and there are several algorithms available to help this choice. The most common of these is Upper Confidence Bounds for Trees (UCT). UCT focuses on the more promising nodes of the tree, while still exploring the lower scoring nodes.
Expansion: The first state found that isn't present in the search tree is added as a new node to the tree.
Simulation: The game or simulation is then executed using random moves to gain knowledge of the average outcome for the selected child node. The node is then given a score that indicates the strength of that choice based on the average outcome of that action.
Back-propagation: The calculated score is then recursively propagated upwards through the nodes of the tree until the root node is reached.

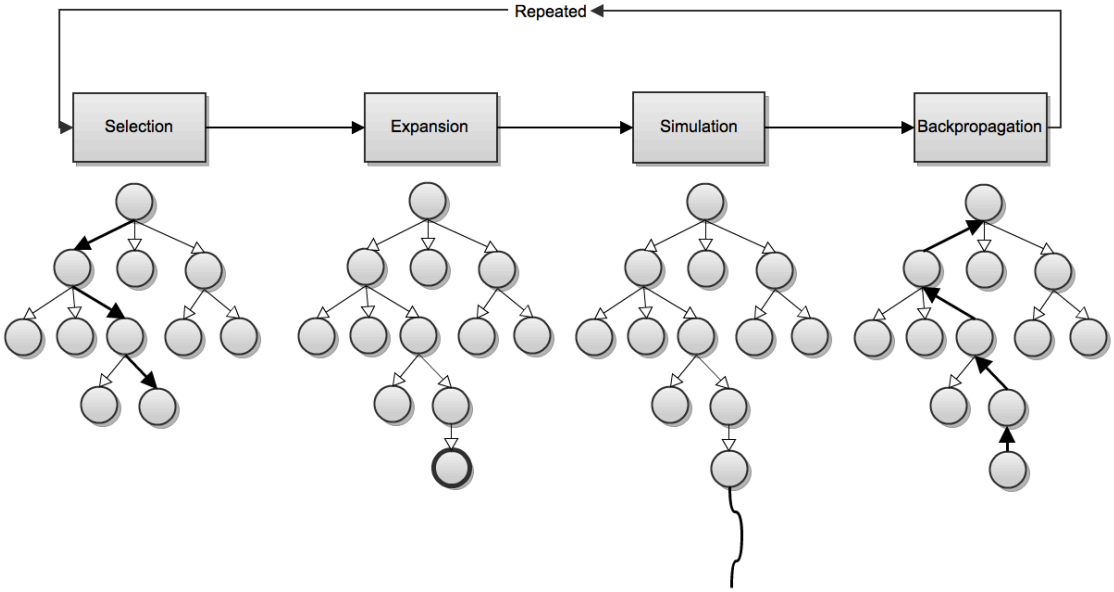


Figure 2. The four steps of the Monte Carlo Tree Search Algorithm. Adapted from (G Chaslot, et al., 2008)

1.1.4. Evolutionary Algorithms

The most common method used in Automated Red Teaming to find an optimal solution to a given problem is the use of evolutionary algorithms. Binary representations of variables are usually used in evolutionary algorithms to represent a problem. This bit string is known as a chromosome and the individual bits are named genes. A collection of chromosomes is known as a population.

The actual algorithm itself is quite simple; the initial population of chromosomes is randomly generated. Each individual is then evaluated using a fitness function and is given a fitness value. At each generation, the algorithm performs selection and reproduction. Selection is the process of choosing a sample of individuals to breed together. “This sample is obviously biased towards better individuals, i.e., good – according to the fitness function- solutions should be more likely in the sample than bad solutions” (Alba & Cotta, 2006).

A pair of selected parent chromosomes then produces a child chromosome using methods known as mutation and crossover. A mutation operation in a binary string chromosome can be a bit flip in a random position. A crossover operation is where a random point in the parent’s bit strings is chosen and the child is a combination of the information before the split point in the first parent and the information after the split point in the second parent. Figure 3 shows an example of the crossover operation.

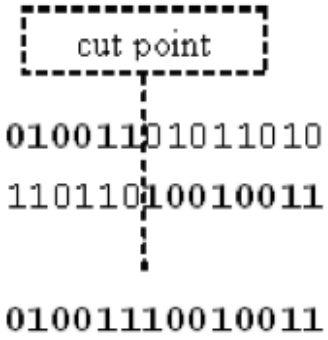


Figure 3. Crossover operation (Alba & Cotta, 2006)

Iteration continues until termination criteria are met. A terminating condition can be minimum fitness level, maximum number of generations reached, maximum computation time is met. The most common method is usually a combination of the above criteria.

In Automated Red Teaming, “the parameter values (e.g. troop clustering/cohesion, response to injured teammates, etc.) Defining the behaviour or personality of the red team are evolved to optimise its efficiency (e.g. maximise damage to target facilities) against the blue team” (Decraene, Zeng, Low, Zhou, & Cai, 2009).

1.2. The significance of the study

Most traditional automated red teaming simulations use genetic algorithms to evaluate the score or performance of a simulation with a particular set of parameters. A disadvantage of this approach is that a simulation must complete execution before the effectiveness of that set of parameters can be evaluated. Combining this with the fact that creating a heuristic evaluation function for non terminal game states can be extremely difficult to implement (G Chaslot, et al., 2008), means that evolutionary methods are not suited for use when implementing re-planning.

The proposed Monte-Carlo Tree Search algorithm would work well within an Automated Red Teaming environment by supporting re-planning and therefore longer simulations. A Monte-Carlo Tree Search algorithm does not rely on a heuristic evaluation function to score game states. While MCTS methods have been used in fields such as chess, board games, Real Time Strategy (RTS) games (G. M. J.-B. Chaslot, 2010), they have not been used in an ART environment combined with a linear programming approach before. This should lead to more information gained from longer simulation leading to more intelligent decisions.

1.3. The purpose of the study

The primary purpose of the study is to improve current Automated Red Teaming methods by including a Monte-Carlo Tree Search component that support replanning and provide an algorithm that chooses the best strategy for the simulation at the current time step in a two-person stochastic zero-sum simulation.

The purpose of this study was to determine if combining a Monte-Carlo Tree Search method with a linear programming scoring approach in an ART environment would provide better results than random strategy choices. If the method provides better results than random strategy selection, then the next step was to determine which methods worked better than other and if re-planning provided a benefit to the average score and how the MCTS approach handled extra randomness in the simulation.

1.4. Research questions

How do Monte-Carlo Tree Search methods perform in an automated red teaming environment compared to random strategy choices?

How do Monte-Carlo Tree Search methods using re-planning perform in an automated red teaming environment when compared to random strategy choices? Is it better to use re-planning methods rather than methods without re-planning?

How do Monte-Carlo Tree Search methods perform given an element of uncertainty caused by Improvised Explosive Devices (IEDs) in an Automated Red Teaming environment?

2. Review of the literature

The literature review will consider the fields of Monte-Carlo Tree Search, red teaming, data farming and the use of Monte-Carlo Tree Search methods in Real Time Strategy (RTS) games.

2.1. Studies into Monte-Carlo Tree Search

The term Monte-Carlo Tree Search was defined by Chaslot in 2006, and came from work in the artificial intelligence field. Monte-Carlo Tree Search was first applied to the game Go (G Chaslot, Saito, Bouzy, Uiterwijk, & van der Herik 2006), but has since been used for many purposes in areas such as Kriegspiel (Ciancarini & Favini, 2010), Texas Hold'em Poker (Kleij, 2010), an RTS game (M. Chung, Burro, & Schaeffer, 2005).

Chaslot states that "building an *adequate* [authors' emphasis] evaluation function based on heuristic knowledge for a non-terminal game state is a domain-dependent and complex task" (G Chaslot, et al., 2008) and also backs this up with "MCTS does not require any heuristic positional evaluation function." MCTS does not require a heuristic evaluation function because the algorithm itself evaluates the score of a move by playing the simulation to a point as talked about by Takeuchi "Monte Carlo Go utilizes the results of random sampling in evaluating positions, instead of hand-coded evaluation functions of heuristic search" (Takeuchi, Kaneko, & Yamaguchi, 2010).

Monte-Carlo Tree Search seeks to solve the problem where building an evaluation function would be extremely difficult or unfeasible such as the game of Go. There are several different methods available for choosing a new node in Monte-Carlo Tree Search. These methods try to strike some balance between exploration and exploitation. The algorithm Upper Confidence Bound for Trees (UCT) was first introduced in 2006 (Kocsis & Szepesvári, 2006). A UCT algorithm is based on the Upper Confidence Bound (UCB) algorithm, which tries to maximise a score or payoff while "occasionally the algorithm might decide to do exploration which improves the knowledge about the reward generating process, but which is not necessarily maximising the current reward" (Auer, 2003).

In 2011, Fern and Lewis presented a method called Ensemble Monte-Carlo Planning. This method involves using UCT methods to create multiple Monte-Carlo search trees for a single problem. The algorithm then uses a weighted voting system to make a decision. The authors test the algorithm on Backgammon, Yahtzee and Connect 4 games successfully showing computational benefits over traditional Monte-Carlo Tree Search methods.

> "Our main observations are the following: 1) Ensembles can significantly improve performance per unit time in a parallel model, 2) Ensembles can significantly improve performance per unit memory in a single-core model, and 3) Contrary to some prior observations, we did not observe a significant improvement in performance per unit time in a single-core model" (Fern & Lewis, 2011)

2.2. Studies into red teaming, data farming and re-planning

Red teaming has a long history in the military and has been used in areas of network security (Markham & Payne, 2001). Automated Red Teaming was first introduced in 2004 by Upton, Johnson and McDonald as a complement to manual red teaming and is described as "we automate this vulnerability discovery process using a combination of evolutionary algorithms and agent-based simulations" (Upton, Johnson, & McDonald, 2004).

Choo, Chua and Tay later built upon this work to create an automated red teaming framework (ART Framework) specifically for military application. Their study showed that there was a clear benefit from using evolutionary algorithms, as apposed to manual red teaming. "Results showed that Red Force survivability can be improved by 27% just by modifying behavioural parameters alone. These findings could be used by Blue force to refine their tactics and strategy thereby ensuring robustness of plans and mission success." (Choo, Chua, & Tay, 2007). Figure 4 shows the ART framework

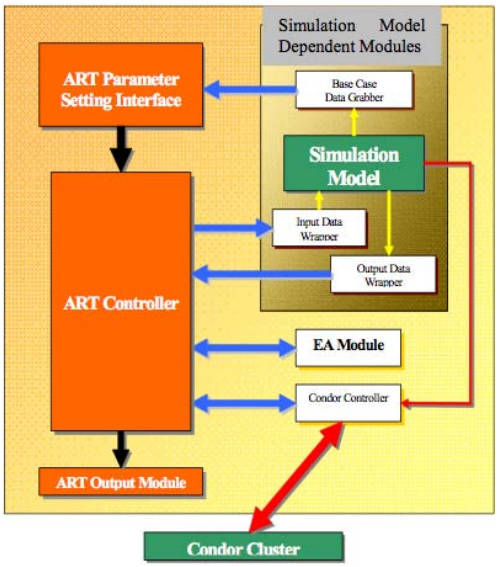


Figure 4. ART Framework (Choo, et al., 2007)

Other work performed in the Automated Red Teaming field showed that ART was able to "enhance the Red performance from their manual red teaming tactics" (S. W. Chung, Choo, Martinez-Tiburcio, & Lin, 2006) and that the ART had produced alternate plans that were not discovered during the MRT process by exploring paths that had initially been dismissed.

Others recognise the problem associated with many-objective problems. "Solving real world problems commonly involves the simultaneous optimization of many objectives which often conflict with each other" (Decraene, et al., 2009). They propose several different techniques to handle multi-objective problems, with future research to focus on "to develop new heuristic techniques capable of efficiently addressing many-objective ART problems" (Decraene, et al., 2009).

In 2008, a study looked at using a game tree structure for decision making in data farming. This technique is named "Strategic Data Farming" and uses a hybrid of real-world war games and computer-based simulations. "Strategic Data Farming uses iteration between human and computer to gain the best of both worlds" (Duong,

2008). Strategic data farming uses a Minimax game tree and a heuristic evaluation function to score moves, as well as an alpha-beta optimisation.

Duong further built upon this work in 2010 and showed that re-planning was effective using a Minimax-based algorithm for choosing a different Course of Action (COA). “In strategic data farming, for each set of parameter values, the simulation is run for, the moves that would have been in a script change according to what achieves the agent’s objectives” (Duong et al., 2010).

Duong et al (2010) state that for a simulation to support decision making it must support three things:

- Indicators that measure how far agents are from their goals at specific time intervals
- A way to save and precisely restore the simulation state (checkpoint/restart)
- A way to make moves

(Duong, et al., 2010)

2.3. Studies similar to this study

In 2005 Chung, Burro and Schaeffer applied a Monte-Carlo based technique to planning in RTS games using the Open RTS software. They created a general Monte-Carlo planning framework named MCPlan. Their technique was to look ahead at each planning point using simulations with random choices and the authors' note that the importance of the research is the level of abstraction they are able to obtain from the game. "MCPlan is similar to the stochastic sampling techniques used for other games" (M. Chung, et al., 2005). The authors discuss the fact that this kind of automated planning is still reliant on expert knowledge for creating plans and creating a good evaluation function that works effectively. "For most application domains, including RTS games, there is no easy way around this dependence on an expert" (M. Chung, et al., 2005)

A similar study by Sailer, Buro and Lanctot focused on adversarial planning in RTS games. They discuss the six main problems associated with planning in RTS games, many of which also apply to automated red teaming situations.

- Complex unit types and actions
- Real-time constraint
- Large game maps and number of units
- Simultaneous moves
- Several opponents and allies
- Incomplete information

(Sailer, Buro, & Lanctot, 2007)

The authors overcome the difficulty of creating an effective evaluation function by simulating to the end of the game and checking if they have won or lost the game. They combine ideas from previous studies on RTS planning with ideas from game theory. "In a zero-sum two-player setting with simultaneous moves the natural move-selection choice then would be to determine a Nash equilibrium strategy by mapping the payoff matrix r into a linear programming problem" (Sailer, et al., 2007). One of the problems associated with this method is the small set of strategies that are used.

In 2009 a Monte-Carlo based tactical planner was implemented for use in RTS games. The main feature of the planner was that it uses UCT for creating a simulation tree. "The key idea behind UCT is to intelligently bias the rollout trajectories toward ones that appear more promising based on previous trajectories, while maintaining sufficient exploration" (Balla & Fern, 2009). The authors discuss that UCT is usually used in sequential moves, however they have applied it to non-sequential, continuous games. The authors presented a domain independent planner that does not require expert knowledge.

There are several studies that build upon this work and try to reduce the search space of particular problems by using other artificial intelligence techniques. Laviers (2010) notes that one of the issues with multi-agent adversarial games is "the size of the search space can be prohibitively large when the actions of all players are considered simultaneously" (Laviers, 2010). The study used a K* classifier to rank groups of moves along with a UCT Monte-Carlo planning approach to create better plans. They concluded that using this method "doubles the offensive team's performance in the Rush 2008 football simulator over prior methods" (Laviers, 2010).

Branavan, Silver and Barzilay (2011) presented a method of reducing the search space in for large sequential planning problems in the game Civilization II. “We approximate the value function by a neural network, augmented by linguistic knowledge that is extracted automatically from the official game manual” (Branavan, Silver, & Barzilay, 2011). They show a clear benefit by using a different, enhanced evaluation function from what would usually be used with a ‘vanilla’ Monte-Carlo Tree Search implementation “Our non-linear Monte-Carlo search algorithm wins over 78% of games against the built-in AI of Civilization II” (Branavan, et al., 2011).

Szita, Chaslot and Spronck (2010) apply a Monte-Carlo Tree Search method to a non-deterministic board game called Settlers of Catan, which is a multiple agent environment with more than two players. The authors show that, when provided with a small amount of domain knowledge, “The playing strength of our agent is notable: it convincingly defeats the hand-coded AI of JSettlers, and is a reasonably strong opponent for humans” (Szita, Chaslot, & Spronck, 2010).

A 2003 study examined utilising game theory concepts for military decision-making in two-person zero sum games. The study outlines a ten step process for calculating a payoff matrix using manual red teaming then solving the problem using software linear programming solvers. “The intent is to translate the abstract concepts of game theory to a well-defined process for organising information to enhance military decision-making. This study offers a model for the military commander to augment the military decision-making process.” (Cantwell, 2003)

In 2009, Cazenave introduced a method called “nested monte-carlo search”. This method deals with the problem where no easy heuristic method of scoring a game exists. At each level of the algorithm, the best set of moves is stored. Then random playouts occur at the lower level. “If none of the moves improves on the best sequence, the move of the best sequence is played, otherwise the best sequence is updated with the newly found sequence and the best move is played” (Cazenave, 2009). The algorithm has been tested successfully on Morphion Solitaire, SameGame and 16x16 Sudoku. This algorithm was further extended to execute in parallel with successful results “The parallel algorithm run at level 3 has found sequences of length 80 which is the current world record at Morphion Solitaire disjoint version” (Cazenave & Jouandeau, 2009). In 2010, it was shown that a new algorithm named MAX, a combination of the Nested Monte-Carlo and UCT methods outperformed the separate methods “MAX gets the performance of both algorithms by taking at each step the move with the best independent evaluation” (Mehat & Cazenave, 2010).

3. Theoretical Framework

The philosophical approach that was used in this study was quantitative methods based on the positivist paradigm. According to Creswell, quantitative methods are “a means for testing objective theories by examining the relationships among variables. These variables, in turn, can be measured, typically on instruments, so that numbered data can be analysed using statistical procedures” (Creswell, 2009). Therefore, the research questions that are developed in a study with positivist views can be answered by using experimental research to determine the effect that the independent variables have on the dependent variables.

Experimental design is the method most commonly used to answer a hypothesis in the fields of automated red teaming and data farming. “Experimental research provides a systematic and logical method for answering ‘If this is done under carefully controlled conditions, what will happen?’” (Best & Kahn, 1989). Taking into account the previous work performed in this field, the most suitable method of testing the research questions is experimental design.

3.1. Identification of variables

The independent variables in this study are the variables that are modified to determine if they have an effect on the dependent variable. The independent variables are the methods of choosing a strategy. There are four different methods that are used to choose a strategy; these are further explained in chapter 5.3:

- Random strategy choices
- Monte-Carlo methods without re-planning (level 1)
- Monte-Carlo methods with re-planning (level 1)
- Monte-Carlo methods with re-planning (level 2)

Another method that will be modified is the Improvised Explosive Device (IED) presence in the scenario. This is also described in more detail in chapter 5.3.

- IEDs present in the scenario
- IEDs not present in the scenario

The dependent variable in this study is the variable that will be measured to determine if the research questions can be proven. The dependent variables is:

- Red team score

The studied simulation is asymmetric because changing the identities of the players will not give the same payoff matrix or final score. This is because we are only measuring the red team’s final score. Other parameters of the simulation include:

- Number of re-planning points
- Squad numbers per team
- Agents per squad
- Objective point scores

These parameters are fixed so they don’t affect the dependent variable. Although the results of modifying these parameters is not analysed in this study, it would be interesting to see the effects in future work.

3.2. Phases of the study

There are six phases that were used in the study. These are based on the recommendations made by Crawford and Stucki (1990). The phases are shown in Figure 5 and described below.

1. Design
2. Implementation
3. Test
4. Collect data
5. Analyse
6. Report

(Crawford & Stucki, 1990).

The software development process of the study was similar to the agile software development method. The agile development process uses iteration with relatively short time limits. Breaking the project down into sections allows faster development time by allowing testing and reworking of discrete parts of the project. The iteration is shown in Figure 5. “That people make mistakes is exactly why iterative and incremental development were invented. Iterative refers to a scheduling and staging strategy that allows rework of pieces of the system” (Cockburn, 2000).

The design phase of the project involved designing the software system that will carry out the set of experiments. Initially a simplified UML version of the overall structure was created. This was enhanced as the project progressed and more details of implementation and design were worked out. Some parts of the final system were written in discrete parts during the implementation phase, but most of the parts of the system were developed at the same time, whilst being tested.

The testing phase included not only testing to check if the parts of the system worked well together, but also tracking down bugs. Fixing bugs was a large part of this phase. Once a part of the system passed the testing phase, the next part was developed. Some time was spent profiling and optimising during this phase.

Data collection occurred in the fourth phase. Many simulations were executed and the resulting scores written to text files. The output from this stage was a results file per combination of search algorithms. These log files were then analysed to give a mean, standard deviation and error of the mean to give the final results (as described in chapter 7). Throughout the separate stages the data was reported and collated into the final information present in this document.

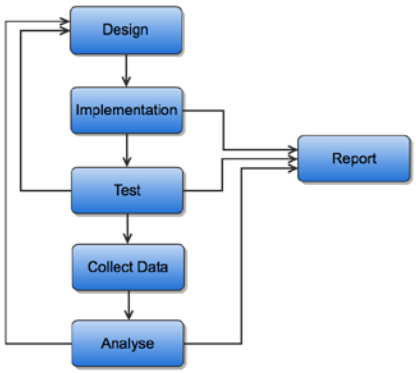


Figure 5. Phases of the project. Iterative Agile Process.

3.3. Software framework

Appendix A shows the main structure of the simulation developed for the study. The simulation framework is loosely based on other Automated Red Teaming frameworks and military style chain of command. Each simulation has two teams; these teams each have a commander class. Each commander class controls a set of squads. Each squad controls a set of agents. After the search algorithms have picked a strategy, the strategy is passed to the commander class, which then sets the strategy for each squad and the agents that make up the squad are given a list of goals (from the strategy). The agents are responsible for the path finding between goals.

The rest of the software framework is loosely based on the Automated Red Teaming Framework shown in Figure 4. ART Framework (Choo, et al., 2007). A controller class is responsible for configuring and executing the search algorithms. The result of the final execution is the red scores from each simulation stored in text files that are analysed later.

Each search algorithm inherits methods from the PlannerInterface class. Similarly, the objects in each search algorithm that depend on the mason simulation are implemented from interfaces. Examples of these are the SimulationInterface, StateInterface, ChoicesInterface and the linear programming SolverInterface.

Each search algorithm is given the initial state for the simulation. It is then responsible for building a search tree, growing the tree, analysing the results and choosing a strategy to play. This is achieved through the execution of simulations and building payoff matrices for each simulation tree. The simulation trees are solved with linear programming to give a chosen strategy. In the case of the random search algorithm, a random strategy is picked and the grow operation is ignored because it does not take into account scores when choosing a strategy to play. For the two search algorithms that can replan, they are given the chance to replan when the main simulation reaches the re-planning point.

4. Materials

Throughout the study the following software instruments were utilised:

- Netbeans Java IDE – Version 6.9.1
- MASON Multi-agent Simulation Toolkit – Version 15
- JFreeChart graphing library
- Lp_solve (5.5.20) – Mixed integer linear programming (MILP) solver
- Classmexer – java memory profiler toolkit

The primary reason for using the above tools is that they are all open source software and cross platform. This turned out to be a useful choice as the finished project was executed on Windows, Linux and Mac OSX operating systems. Open source software has gained a reputation for reliable, portable software. “Open source software is well known today for its high degree of reliability and portability” (Bonaccorsi & Rossi, 2003). Netbeans was used because it contains a powerful profiler and debugger and supports many different programming languages.

Mason was chosen because it provides a solid cross-platform agent-based simulation toolkit, whilst being open-source and was designed to be a “general-purpose single-process discrete-event simulation library intended to support diverse multi-agent models across the social sciences, artificial intelligence and robotics” (Balan, Cioffi-Revilla, Luke, Panait, & Paus, 2003). Mason separates the simulation core and the GUI of a simulation. This means that simulations written using mason can easily be executed without a user interface and run as a single process. This made scaling up significantly less complex.

The mason toolkit has a feature called checkpointing that allows the state of a simulation to be saved and written to disk where it can be resumed easily. Checkpointing also enables the simulation to be resumed on a different system than the one on which it was started. The applications of saving a simulation state like this are to enable front-end visualisation of a simulation that was running on a cluster / high performance computing environment (Balan, et al., 2003) or to save the state of a longer simulation incrementally so it can be resumed at a later point if any errors occur.

Although the checkpointing system of mason wasn’t used directly in this study, the fact that was implemented made saving the state in memory incredibly easy. Rather than writing the state to disk, for performance reasons, states were stored in memory. If a significantly larger scenario is required for future research, checkpointing to disk will be a useful feature.

JFreeChart is an open-source graphing library for java. JFreeChart was used for tracking agent scores and casualties throughout the project. It was chosen because it is included as a required library for mason and many tutorials for using JFreeChart and mason together are readily available.

Lp_solve is a free, open-source Mixed Integer Linear Programming (MILP) solver and was a crucial part of the project. Lp_solve was used in the study for measuring the effectiveness of strategy choices at intermediate simulation tree nodes. This software was chosen because it is well documented, and provides the source code for the java bindings as well as the core software. This was very important because the core Lp_solve library had to be recompiled for each system is was used on.

The hardware instruments that were utilised in the study are:

- Laptop (2.66GHz Dual Core Intel i7 with 8GB RAM)
- SGI Altix 1300. 512-core cluster High Performance Computing environment with 2GB RAM per core named XE.
- SGI Altix 3700 Bx2 (192 x 1.5GHz Itanium2 with 384GB RAM) named COGNAC.
- Desktop computer (3.8GHz Quad Core Intel i7 with 6GB RAM)

The software required to execute the simulations was developed on a laptop. Most of the execution of the simulations was performed on the XE machine with the other machines being used mostly for testing.

5. Experiment

This chapter contains the description of the scenarios and how they are designed to answer the research questions that were introduced in 1.4. The main basis of the search algorithm is described in section 5.1. This chapter also introduces the scenario used in the simulations, how the scenario is used to test the research questions and shows how linear programming is used to give a team a score in the intermediate steps of the algorithm. Also discussed are the Nash Equilibrium solutions for several of the search algorithms.

5.1. Scenario

For this study, a scenario was created that supports a single re-planning point. The simulation is non-deterministic and consists of two teams: red and blue. Figure 6 shows the abstract representation of the scenario used in this study. The red team starts at node A, while the blue team starts at node I. There are 4 distinct paths from the start nodes to goal nodes (D, E, F) for each team. This gives a total of 16 total strategy choices at the first level of the tree. The scores for the scenario are fixed, with the scores for nodes d, e, and f being 2, 3, and 5 respectively. Figure 7 shows the mason implementation of the scenario. Scenarios are stored in an xml format, which makes modifying and creating new scenarios easy.

The teams are limited to a small number of strategies that they can choose from. A strategy is simply a list of nodes leading to a goal node. E.g. (I, H, F). Limiting the strategy choices also limits the computational complexity of the simulation, which was essential for this study as time was a principal limitation. As the depth of a scenario increases, the memory consumption and execution times increase exponentially. Limiting the depth allowed more simulations to be executed and lead to higher accuracy in the final results.

Each team has a commander class that is responsible for selecting strategy choices using the MCTS methods and controlling squads. The selected strategy choice is then passed on to the squads present in the team. Each squad is made up of multiple agents that act autonomously based on their given strategy selection. Each agent has a pathfinder that calculates the path to the goal node. The squad is governed by a number of rules such as flocking rules that keep the agents in a close formation with some separation. An agent can have three different states: idle, pathfinding and attacking as shown in Figure 6. An agent can have an idle state when there are no further goals to move towards. An agent has a pathfinding state when moving towards goal nodes. An agent has an attacking state when the agent is within a certain distance of an enemy agent. When an agent is within attacking distance of an enemy agent, the agent has a 50% chance of either killing the enemy agent or being killed.

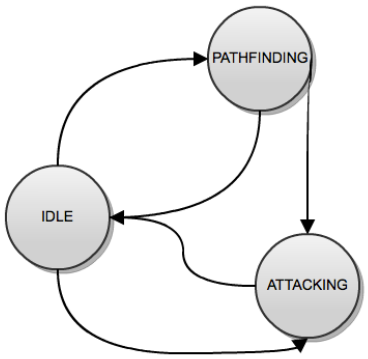


Figure 6. Agent state diagram

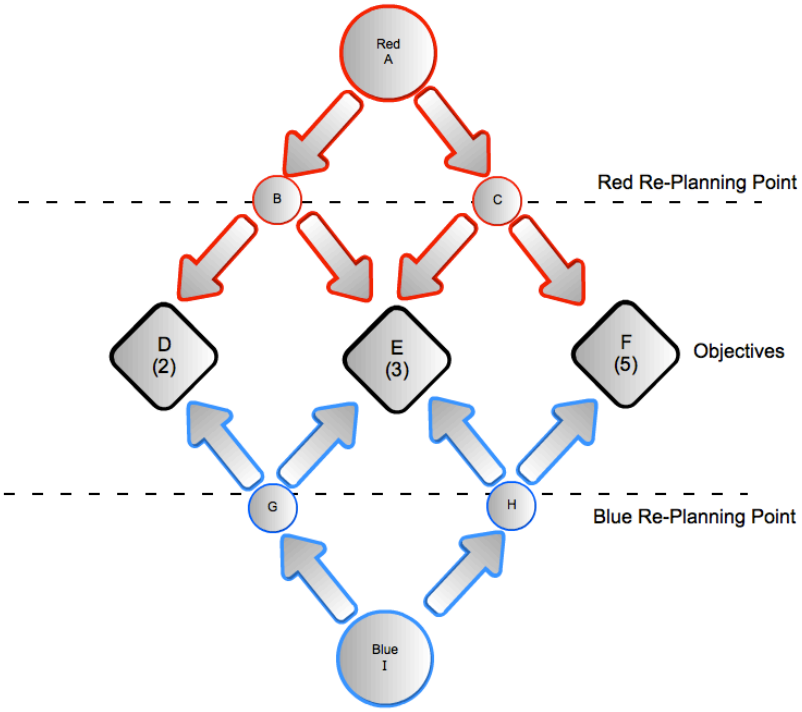


Figure 7. The Scenario.

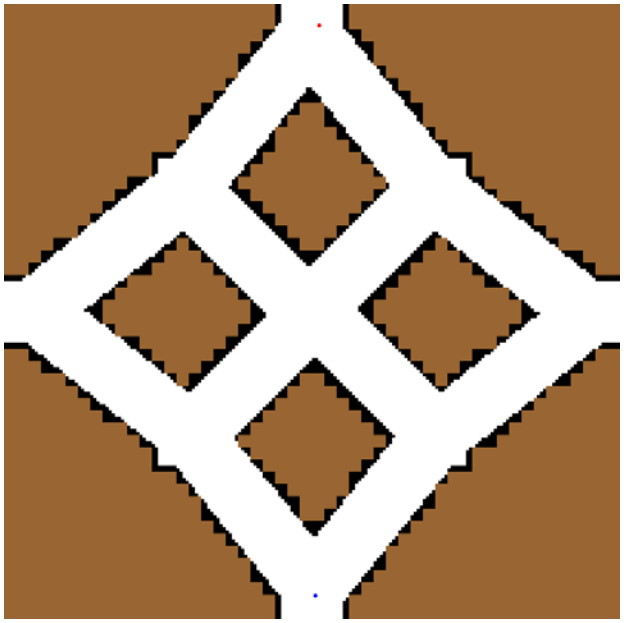


Figure 8. The implementation of the scenario.

5.2. Search Algorithm

The search algorithm makes use of several data structures. A simulation can be thought of as a matrix that can be evaluated and subsequently scored. The cells of the matrix store a data structure named a continuation. Each continuation can stored multiple simulation trees.

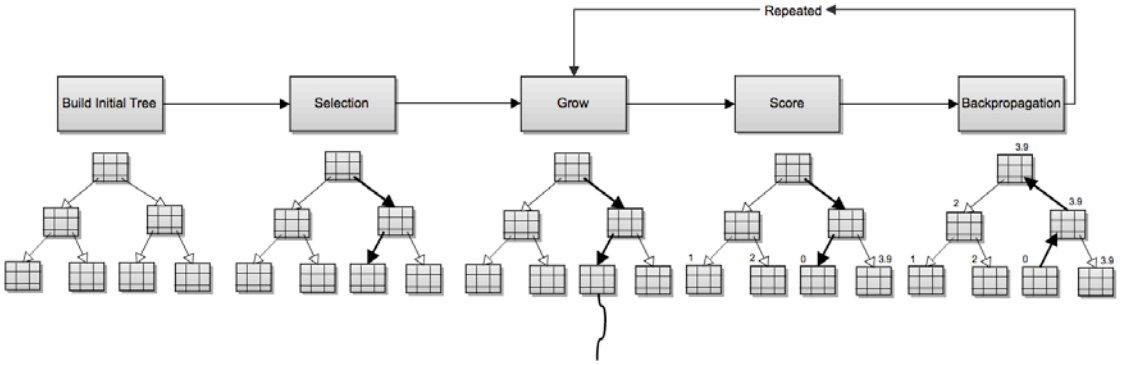


Figure 9. MCTS with Nash Solution Scoring

The search algorithm used in this project has five steps. First, an initial simulation tree is built using mutual recursion. For each pair of red/blue strategy choice combinations, we add a continuation to the tree. Each continuation is passed the level of the simulation, the current simulation state and the strategy choices for each team. During this initial tree construction, continuations are simulated to a certain point given their level. For example, a continuation with a level of 2 or higher will simulate to a re-planning point and a continuation with a level of 1 will execute the simulation to the end point.

If another re-planning point exists, another simulation tree is added a level below, with each set of choices containing continuations, the algorithm recursively continues until the last level contains a continuations with no simulation trees attached.

The second step selects a node at random to grow. This study used a random selection method, however, future work on this algorithm could explore how changing the selection strategy changes the outcome of the simulations and whether there is any benefit to adjusting the selection method.

The grow step uses the selected node and executes the simulation with the strategy selection for each team. The result is added to the parent simulation tree. This execution of the simulation only requires the input simulation state, the team strategies and whether or not to replan. Given that this method is discrete and does not rely on any other simulations this makes it viable to separate simulations into separate threads. This was not focussed on in this study because the simulations that are being executed are embarrassingly parallel and the methods being compared were simply separated out into different executions. However, this would make interesting further work to support huge simulations. The grow step is performed at least 100 times in this study.

The scoring step scores the individual simulation trees. Each simulation tree acts as a payoff matrix that stores the scores for each pair of red/blue strategy choices in the tree. For the leaf node simulation trees we use the scores calculated from the measures of effectiveness.

For this study, the scores for a terminal state are simply a summation of the scores for each squad from the red team. This means that each squad in the red team can capture a goal node and get the full score, or the blue team defeats the red team and the red team gets a score of 0. Using this method we expect the payoff where two teams meet to be half of the goal node's score. For simulation trees that contain multiple continuations, these scores are averaged.

The final step is an implicit method from the scoring process. After the leaf nodes of the tree are scored, the result scores populate the payoff matrices above them in the tree. The intermediate nodes are then scored using linear programming to determine the Nash Equilibrium.

Throughout the simulation each team has it's own planner that either uses the algorithm from Figure 8, or a random planner. This allows the comparison of search algorithms. Each team can then estimate expected payoffs and plan accordingly.

5.3. Procedure

To answer the research questions, four different search algorithms are used. These are as follows:

1. Random – The search algorithm builds a search tree of level 1 that ignores re-planning points and results in strategies from the start nodes to the goal nodes. The simulation tree is not grown at all. The search algorithm picks an initial random strategy, at a replanning point, the random planner is asked for another strategy that is chosen at random from the available choices.

2. Without Replan Depth 1 – This search algorithm builds a search tree of level 1, which does not take into account the re-planning points The tree is then grown a minimum of 100 times and the search algorithm picks a strategy based on the results from the linear programming solution. No replanning occurs and the search algorithm cannot change its chosen strategy.

3. With Replan Depth 1 – This search algorithm builds a tree of depth 1 that ignores the replanning points. The tree is then grown a minimum of 100 times. A strategy is then picked based on the results of the linear programming solution. At the replanning point, this search algorithm builds another tree of level 1 and replans. A new strategy can then picked based on the results of the linear programming solution for this new tree.

4. With Replan Depth 2 - This search algorithm builds a tree of depth 2 that includes the replanning points. This tree is then grown a minimum of 100 times. A strategy is then picked based on the results of the linear programming solution. At the replanning point this search algorithm is given the chance to replan, by creating a search tree of level 1, which is then grown a minimum of 100 times. A new strategy then may be chosen based on the results of the linear programming solution for this tree.

Given that the red team score is the dependent variable and all results are measured in final red score, the notation used in this study when two search algorithms are being compared always lists the red search algorithm first. For example Random vs. With Replan Depth 1 refers to the red team using the random search algorithm and the blue team using the With Replan Depth 1 search algorithm. This notation is also interchangeable with numbers instead of names. E.g. 1,3 could represent the previous example.

The first research question states – “Can Monte-Carlo Tree Search methods be used for planning in automated red teaming?” This research question will by comparing the results of the random search algorithm and the MCTS search algorithm that does not use replanning.

First the theoretical results are calculated and then compared to an abstract problem to test if the implementation of the MCTS algorithm is correct. Then a baseline score can be calculated by comparing the Random search algorithm against itself (where both teams use the random search algorithm). This baseline score can then be compared against the scores of MCTS without replanning vs. the Random search algorithm. It is expected that the MCTS without replanning scores will be higher than the random baseline as the red team is trying to maximise it’s own score. We can also compare the score when the blue team uses the MCTS with replanning methods to the random baseline.

The second research question states – “Can Monte-Carlo Tree Search methods be used for re-planning in Automated Red Teaming? How effective are they compared to methods where no re-planning is employed?” This research question will be answered in two parts.

For the second part of the question we can compare the results of the monte-carlo methods that use replanning to the monte-carlo methods that do not use replanning. If the results have higher red scores, then the conclusion can be drawn that Monte-Carlo Tree Search methods with replanning are more effective than methods where no replanning is employed.

The idea of Improvised Explosive Devices (IEDs) was used to test how the search algorithms handle extra randomness in the scenario. When IEDs are present in the simulation and a team reaches a replanning point, an Improvised Explosive Device could kill a random number of agents in that team. The deaths possible per team for this study was up to half of the agents. Two methods are then proposed, how the search algorithms perform normally and how they perform with the added randomness of IEDs.

The data will be collected from executing many different simulations designed to test the research questions as described above. It is important to determine a baseline score against which other scores can be measured calculated where both teams use random strategy selection. Each team tries to maximise their own score and minimise their opponent’s score, but globally, the red team is the minimising team and the blue team is the maximising team in the simulations. The data will be outputted into a number of text files for analysis.

Each simulation will use the scenario shown Figure 6. Each experiment will be repeated 100 times, to ensure accuracy. We can then build matrices that compare the different results.

No Replanning

		Blue	
		Random	Level 1
Red	Random		
	Level 1		

With Replanning

		Blue		
		Random	Level 1	Level 2
Red	Random			
	Level 1			
	Level 2			

5.4. Linear programming method

The strategies produced for a two-person zero-sum game can be solved with linear programming. The solution gives us what is known as the Nash equilibrium for the game where "no team has any incentive to deviate [from their chosen strategies]" (Sailer, et al., 2007). Finding the Nash equilibrium gives us the ratio of strategies each team should choose to get the maximum payoff.

This section covers the linear programming method used in the study and shows how the results are calculated for a payoff matrix that contains the red player score for each combination of red and blue strategy choices. We build a strategy matrix where the rows represent the red team's (player 1's) strategy choices, and the columns represent the blue team's (player 2's) strategy choices. The following payoff matrix can be built when either team has the choices {d, e, f} and shows how strategy pairs can be combined into a payoff matrix. Each strategy pair has a score associated with it. For example: The score {d, d} would be the averaged payoff for all simulations where both teams choose d as a strategy choice.

	Blue: d	Blue: e	Blue: f
Red: d	{d, d}	{d, e}	{d, f}
Red: e	{e, d}	{e, e}	{e, f}
Red: f	{f, d}	{f, e}	{f, f}

Luce and Raiffa state the method for determining the Nash Equilibrium for Player 1 (red) is:

Let *U* be the set of all *m*-tuples **u** = (u_1, u_2, …, u_m) such that

$$u_i \geq 0, \quad for\ i = 1, 2, \ldots, m$$

In this study the u values are the strategy pairs for the matrix, referred to as x1, x2, and x3 for this example.

and

$$\sum_{i=1}^{m} a_{ij} u_i \geq 1 \quad for\ j = 1, 2, \ldots, n$$

To find those **u** belonging to *U* such that $\sum_{i=1}^{m} u_i$ is a minimum. (Luce & Raiffa, 1989) This means we are minimising the u values (x1, x2 and x3) for this example. Combining this with the scores from the payoff matrix we get the following linear equations:

```
minimise: x1 + x2 + x3

subject to:
x1 >= 0
x2 >= 0
x3 >= 0

{d, d} x1 + {e, d} x2 + {f, d} x3 >= 1
{d, e} x1 + {e, e} x2 + {f, e} x3 >= 1
{d, f}  x1 + {e, f} x2 + {f, f} x3  >= 1
```

Luce and Raiffa (1989) state that "Player 1 should attempt to find a **u** in U which *maximises* $\frac{1}{\sum_i Ui}$ or, equivalently, minimises $\sum_i u_i$" (Luce & Raiffa, 1989). This means that player 1 (the red team) can either be the minimising player using the standard payoff matrix, or can be the maximising player using the transpose of the payoff matrix. In this study, the red team is the minimising player.

Solving this set of equations gives us four outputs – the objective value and the values for each strategy choice. From these numbers we can determine how many times to choose each objective to get the maximum payoff possible. The inverse of the objective value is then used as a score to propagate up the simulation tree. The values returned for each variable can be used to calculate the probabilities for each strategy choice as follows:

$$x1\ probability = \ x1 * \frac{1}{ObjectiveValue}$$

$$x2\ probability = \ x2 * \frac{1}{ObjectiveVa\ \ lue}$$

$$x3\ probability = \ x3 * \frac{1}{ObjectiveValue}$$

Solving this linear programming problem for the blue team is similar. The payoff matrix from the red team's point of view is transposed. The blue team is the maximising team, so we are trying to maximise x1, x2, and x3. When maximising, the constraints are set to less than or equal to 1. Chapter 5.5 solves for the payoffs expected from the different search algorithms.

5.5. Nash equilibrium for the scenario

This section will use linear programming to solve the scenario by hand to determine the Nash Equilibrium for several of the search algorithm combinations. The methods evaluated will be random strategy choices, MCTS without replanning and MCTS with replanning. If the two teams meet at a goal node, the red payoff is half of the actual goal node score because the probability of a team winning a battle is 50%. The full solutions are present in Appendix B (10.2).

5.5.1. Expected outcome with random choices

This section will calculate the expected red score given both teams using random strategy choices. The scenario in Figure 6 has 4 distinct paths from the start nodes to the end nodes for each team. Given that two of the paths lead to the E node, the probability of ending up on that node is higher than landing on the other two nodes. The probability is calculated for each goal node where the red team reaches the goal node and the blue team chooses a different node as well as the case when the two teams meet.

D Node Probability:

Red team chooses D. Blue team chooses a different goal node.
$\frac{1}{4} * \frac{3}{4} * 2$

Red team chooses D. Blue team chooses D.
$\frac{1}{4} * \frac{1}{4} * 1$

E Node Probability:

Red team chooses E. Blue team chooses a different goal node.
$\frac{2}{4} * \frac{2}{4} * 3$

Red team chooses E. Blue team chooses F.
$\frac{2}{4} * \frac{2}{4} * 1.5$

F Node Probability:

Red team chooses F. Blue team chooses a different goal node.
$\frac{1}{4} * \frac{3}{4} * 5$

Red team chooses F. Blue team chooses F.
$\frac{1}{4} * \frac{1}{4} * 2.5$

Expected Red Score:

0.375+0.0625+0.75+0.375+0.9375+0.15625 = **2.656**

5.5.2. Nash equilibrium ignoring replanning

This section will calculated the Nash Equilibrium solution for where both players are using a MCTS search algorithm where re-planning is not used. Each team has 4 strategy choices so we can build a payoff matrix of size 4x4. Each strategy choice is a complete path from the start node to a goal node. The payoff matrix is shown below.

	Blue: I, G, D	Blue: I, G, E	Blue: I, H, E	Blue: I, H, F
Red: A, B, D	1	2	2	2
Red: A, B, E	3	1.5	1.5	3
Red: A, C, E	3	1.5	1.5	3
Red: A, C, F	5	5	5	2.5

This gives us the following linear equation for the Red team:

```
/* Objective function */
min: +X1 +X2 +X3 +X4;

/* Constraints */
R1: +X1 >= 0;
R2: +X2 >= 0;
R3: +X3 >= 0;
R4: +X4 >= 0;

+X1   +3 X2   +3 X3   +5 X4 >= 1;
+2 X1 +1.5 X2 +1.5 X3 +5 X4 >= 1;
+2 X1 +1.5 X2 +1.5 X3 +5 X4 >= 1;
+2 X1 +3 X2   +3 X3   +2.5 X4 >= 1;
```

Solving this gives us the following values:
Objective Value: 0.35555555
x1: 0
x2: 0.2222222
x3: 0
x4: 0.1333333

This means that the Nash Equilibrium score for this scenario ignoring replanning is 1.0/objective value = **2.8125**.

To achieve this score, the red team should make the following choices:
Percentages:
x1: 0%
x2: 62.5%
x3: 0%
x4: 37.5%

To solve for the blue team, we transpose the matrix and the blue team becomes the minimising player. This gives us the following linear equation:

```
/* Objective function */
max: +X1 +X2 +X3 +X4;

/* Constraints */
R1: +X1 >= 0;
R2: +X2 >= 0;
R3: +X3 >= 0;
R4: +X4 >= 0;

+X1 +2 X2 +2 X3 +2 X4 <= 1;
+3 X1 +1.5 X2 +1.5 X3 +3 X4 <= 1;
+3 X1 +1.5 X2 +1.5 X3 +3 X4 <= 1;
+5 X1 +5 X2 +5 X3 +2.5 X4 <= 1;
```

Solution for blue:
Objective Value: 0.355555555
x1: 0
x2: 0.04444444444
x3: 0
x4: 0.31111111111

The optimal strategy for the blue team is then to play the strategies with the following probabilities:
x1: 0%
x2: 12.5%
x3: 0%
x4: 87.5%

5.5.3. Nash equilibrium with replanning

This section solves for the Nash Equilibrium where both players are using MCTS methods with depth 2 replanning. When we include re-planning, we have two initial choices for each team, which gives us a 2x2 payoff matrix as shown below. We need to populate the values from the payoff matrix by solving the linear equations at the lower level. This means there are four other linear programming problems to solve before we can solve the top-level matrix.

	Blue: G	Blue: H
Red: B		
Red: C		

Red Chooses B, Blue Chooses G:

	Blue: D	Blue: E
Red: D	1	2
Red: E	3	1.5

This gives the following linear equation for the red team:

```
/* Objective function */
min: +X1 +X2;

/* Constraints */
R1: +X1 >= 0;
R2: +X2 >= 0;

+X1 +3 X2 >= 1;
+2 X1 +1.5 X2 >= 1;
```

Objective Value: 0.5555
x1: 60%
x2: 40%
Score: 1.8

Red Chooses B, Blue Chooses H:

	Blue: E	Blue: F
Red: D	2	2
Red: E	1.5	3

This gives the following linear equation for the red team:

```
/* Objective function */
min: +X1 +X2;

/* Constraints */
R1: +X1 >= 0;
R2: +X2 >= 0;

+2 X1 +1.5 X2 >= 1;
+2 X1 +3 X2 >= 1;
```

Objective Value: 0.5
x1: 100%
x2: 0
Score: 2

Red Chooses C, Blue Chooses G:

	Blue: D	Blue: E
Red: E	3	1.5
Red: F	5	5

This gives the following linear equation for the red team:

```
/* Objective function */
min: +X1 +X2;

/* Constraints */
R1: +X1 >= 0;
R2: +X2 >= 0;

+3 X1 +5 X2 >= 1;
+1.5 X1 +5 X2 >= 1;
```

Objective Value: 0.2
x1: 0%
x2: 100%
Score: 5

Red Chooses C, Blue Chooses H:

	Blue: E	Blue: F
Red: E	1.5	3
Red: F	5	2.5

This gives the following linear equation for the red team:

```
/* Objective function */
min: +X1 +X2;

/* Constraints */
R1: +X1 >= 0;
R2: +X2 >= 0;
+1.5 X1 +5 X2 >= 1;
+3 X1 +2.5 X2 >= 1;
```

Objective Value: 0.3555
x1: 62.5%
x2: 37.5%
Score: 2.8125

We can then combine the scores from the previous steps into the first matrix and solve to get the expected score for the scenario:

Top Level of the Scenario:

	Blue: G	Blue: H
Red: B	1.8	2
Red: C	5	2.8125

```
/* Objective function */
min: +X1 +X2;

/* Constraints */
R1: +X1 >= 0;
R2: +X2 >= 0;
+1.8 X1 +5 X2 >= 1;
+2 X1 +2.8125 X2 >= 1;
```

Objective Value: 0.35555
x1: 0%
x2: 100%
Score: **2.8125**

So the expected score for this scenario when replanning is used is 2.8125 and red should always choose C at the top level.

6. Limitations

The primary limitation of this study was time. This limitation has had an effect on most areas of the study. Other limitations include scope, depth of simulation and memory limitations. In this section I will outline how these limitations affected areas of this study and how they were counteracted.

The study had a number of simulations that had to be executed in a relatively timely manner. One possible solution to this problem was more computational power. I had an initial budget of 5000 hours booked on a 512-core SGI Altix machine that would allow for quicker execution in parallel compared to sequentially on a laptop or desktop machine with the backup plan of leaving myself enough time to run on the laptop and desktop machine if something went wrong. Some problems occurred with memory issues on the Altix machine and some of the earlier simulations had to be executed on a desktop machine. This did not adversely affect the timing of the results.

Downtime and system outages can also be a potential problem when it comes to computing resources. To manage the risk of outages affecting the study, I spread out the total computation over a slightly larger period of time so I was less likely to be affected by a small outage.

Another limitation of the study was the replanning depth of the simulation or the number of squads per team. The simulation that was chosen had a depth of 2. Given that adding another level of depth or adding another squad per team to the simulation increases the number of simulations that need to be executed exponentially, this limitation was closely related to computational time and memory consumption.

Memory consumption was an issue for the study, because a naïve method of storing states was used. A trade off was eventually made so that the stored states would take more memory, but execute faster. An option was included to compress states so that for huge simulations where much more time is available for execution can still run in a reasonable amount of memory. Another potential fix for this limitation was to optimize the states being saved, however, given the time available this wasn't possible or necessary to complete the project.

Scope was a large limitation for this study as many other scenarios could have easily been included as well as other areas of study such as just analysing which pair of choices to explore / exploit in the grow method of the search algorithms. Feature creep was a real risk for the study and was limited by placing limits on the number of scenarios / number of squads / objectives of the study. Some feature creep did occur, but was within limits.

7. Results

This section contains the results from the experiments introduced in chapter 5.3.

7.1. Theoretical Results

This section contains the results calculated in chapter 5.5 and these represent the scores expected from the abstract scenario and mason scenarios.

No Replanning

		Blue	
		Random	Level 1
Red	Random	2.656	2.609
	Level 1	3.047	2.8125

Figure 10

With Replanning

		Blue		
		Random	Level 1	Level 2
Red	Random	2.656	2.312	2.281
	Level 1	2.652	2.392	2.305
	Level 2	3.90625	3.086	2.8125

Figure 11

7.2. Abstract Scenario

This section contains the results from the initial abstract scenario that is a version of the same scenario used in the mason simulation but the agents and simulation have been simplified down to the simplest possible representation. The abstract scenario was used to test the validity of the search algorithm against the theoretical results.

The values in the tables are shown with the mean value for 1000 iterations and the second value is the error of the mean for that run of simulations.

No Replanning

		Blue	
		Random	Level 1
Red	Random	2.656 ± 0.01	2.613 ± 0.01
	Level 1	3.044 ± 0.01	2.805 ± 0.01

Figure 12

With Replanning

		Blue		
		Random	Level 1	Level 2
Red	Random	2.656 ± 0.01	2.323 ± 0.01	2.279 ± 0.005
	Level 1	2.644 ± 0.01	2.408 ± 0.01	2.304 ± 0.01
	Level 2	3.896 ± 0.01	3.083 ± 0.01	2.827 ± 0.01

Figure 13

The results in Figure 11 and Figure 12 match (within standard error) the theoretical calculations in Figure 9 and Figure 10. The trend of scores shows an increase as they move down the rows and a decrease as they move across the columns of the table. This indicates that more planning is beneficial to the red team, and as the level of planning increases for the blue team they defend better and limit the red team's possible payoff.

Given that the results from the abstract scenario show values matching the theoretical results extremely closely, it can be concluded that the search algorithm is working correctly and the mason simulation can be executed.

7.3. Mason Scenario

This section contains the results from the chosen scenario when the mason toolkit was used. Each set of simulations was executed with multiple agents, with 100 iterations and growing the tree 100 times.

No Replanning

		Blue	
		Random	Level 1
Red	Random	2.620 ± 0.170	2.680 ± 0.125
	Level 1	2.960 ± 0.198	2.790 ± 0.182

Figure 14

With Replanning

		Blue		
		Random	Level 1	Level 2
Red	Random	2.620 ± 0.170	2.300 ± 0.164	2.130 ± 0.159
	Level 1	3.460 ± 0.175	2.770 ± 0.165	2.530 ± 0.142
	Level 2	3.900 ± 0.162	2.990 ± 0.171	2.790 ± 0.185

Figure 15

Figure 13 shows the results from the Mason scenario where no re-planning is used. This table can be used to answer the first research question. The first research question states “Can Monte-Carlo Tree Search methods be used for planning in automated red teaming?” The values in Figure 13 show an increasing trend as the red team’s search algorithm is changed from random to a level 1 search-algorithm without re-planning. Where the blue team is using random strategy selection, the red values change from 2.620 to 2.960, which are both within the standard error of the mean from the calculated theoretical results. Similarly, where the blue team is employing the level 1 search algorithm, the red values change from 2.680 to 2.790. These values are also within the standard error of the mean from the calculated theoretical results and have the same trend as the values in Figure 11. From these results we can conclude that methods that use Monte-Carlo Tree Search based planning perform better than random methods.

The first research question is then confirmed as true and Monte-Carlo Tree Search methods can be used for planning in an automated red teaming environment.

The second research question states “Can Monte-Carlo Tree Search methods be used for re-planning in Automated Red Teaming? How effective are they compared to where no re-planning is employed?” The values from Figure 14 can be used to answer this research question. The table shows that as the red search algorithm’s move from random to level 1 and finally to level 2 with replanning, the trend is an increasing value. This trend holds true for each of the different blue team’s search algorithms that are tested. As expected, the red payoffs increase as the red team’s level of planning in increased and decreases as the blue team uses more in depth planning methods.

The first part of the second research question can be answered by comparing the results in Figure 14 to Figure 13. Examining the values where the red team is using a level 1 method and the blue team uses random strategy selection, the re-planning method gives a higher final score than the scenario where no replanning is used. Comparing the opposite situation where the red team is using random strategy selection and the blue team is using a level 1 method, the re-planning value is lower

than the value where no re-planning is used. This is because the blue team can better minimise the red team's payoffs when re-planning is employed.

This can answer the first part of the second research question and the conclusion can be made that Monte-Carlo Tree Search methods can be used for re-planning in Automated Red Teaming because the values from the re-planning search algorithms are greater than random chance.

Comparing level 2 re-planning methods to the method without replanning where the blue team is using random strategy selection shows that the re-planning method gives a significantly higher red payoff. Similarly, comparing the level 2 re-planning method where the red team is using random strategy selection shows a much lower score than the no-replanning method. From these results the conclusion can be made that Monte-Carlo Tree Search methods with re-planning are more effective than methods where no re-planning is employed.

7.4. Mason Scenario with IEDs

This section contains the results using the same parameters from section 7.3 with the addition of IEDs. An IED explodes just before a re-planning point so that the information can be used by re-planning methods. When an IED explodes, it has the potential to kill up to half of a squad's agents. The results of this section are used to answer the third research question.

No Replanning

		Blue	
		Random	Level 1
Red	Random	2.580 ± 0.160	2.630 ± 0.159
	Level 1	3.370 ± 0.187	2.880 ± 0.184

Figure 16

With Replanning

		Blue		
		Random	Level 1	Level 2
Red	Random	2.580 ± 0.160	2.490 ± 0.150	2.130 ± 0.159
	Level 1	3.470 ± 0.166	2.730 ± 0.176	2.590 ± 0.163
	Level 2	3.240 ± 0.203	3.040 ± 0.160	2.770 ± 0.165

Figure 17

The third research question states, "How do Monte-Carlo Tree Search methods handle extra randomness in an Automated Red Teaming environment?" The results in Figure 15 and Figure 16 compared to the results where no IEDS are used show similar results.

8. Conclusion

This study has shown in several ways that Monte-Carlo Tree Search methods can be used successfully in an Automated Red Teaming environment using calculated theoretical results, results from an abstract scenario and the results from the Mason simulation present in chapter 7.3. The theoretical results were essential to checking if the algorithm was feasible and would show an improvement over current methods.

The abstract scenario was created to test if the MCTS algorithm was working correctly without introducing potential bugs from the Mason simulation. The abstract simulation showed similar results to the theoretical results.

Chapter 7.3 also shows that Monte-Carlo Tree Search methods that use re-planning can be used in an Automated Red Teaming environment and give better results than other methods such as random strategy choices and MCTS based strategy choices without re-planning.

Chapter 7.4 shows the results from the Mason experiment that uses IEDs. The results are similar to the results shown in chapter 7.3 so for this experiment with the parameters used, introducing IEDs into the scenario does not provide a significant example for either team. Further work is required to test if different scenarios using differing amounts of agents per squad and a higher IED death rate would make a difference to this score.

This project has shown that MCTS based methods provide a clear increase in red team score in an automated red teaming environment when compared to other methods. An extensible and generic MCTS framework was built successfully.

9. Further Work

There are many areas of future work that could be explored. One of these is parallelising the MCTS algorithm itself. When executing the search algorithm, the largest amount of time is spent in building and growing the tree including running the simulation itself. Chaslot (2010) states that “MCTS benefits substantially from parallelization” (G. M. J.-B. Chaslot, 2010).

Another area to be explored is the method that is used to grow the simulation tree. Both a random grow method and a method that was biased towards exploration were used for the algorithm in this study. The random method was eventually used because the speed of execution was increased as the other method required scoring the entire simulation tree before each grow operation. This area should be explored to test if a better grow method is available to produce better results and faster execution.

The search algorithm should be tested with scenarios that have more re-planning points. Given that the search algorithm has been proven to be effective on the smaller scenario by comparing the results from the implementation to the theoretical results, a larger scenario that has a higher depth could be explored. The difficulty of exploring larger scenarios is the increase in required memory and computational power, as well as it being very difficult or impossible to calculate the theoretical results of a larger simulation.

The IED experiment could be tested with varying levels of randomness to check how the different levels affect the end results. For example, instead of up to half of the

team having a chance of being killed by an IED, situations where all of the team could potentially be killed by an IED could be explored.

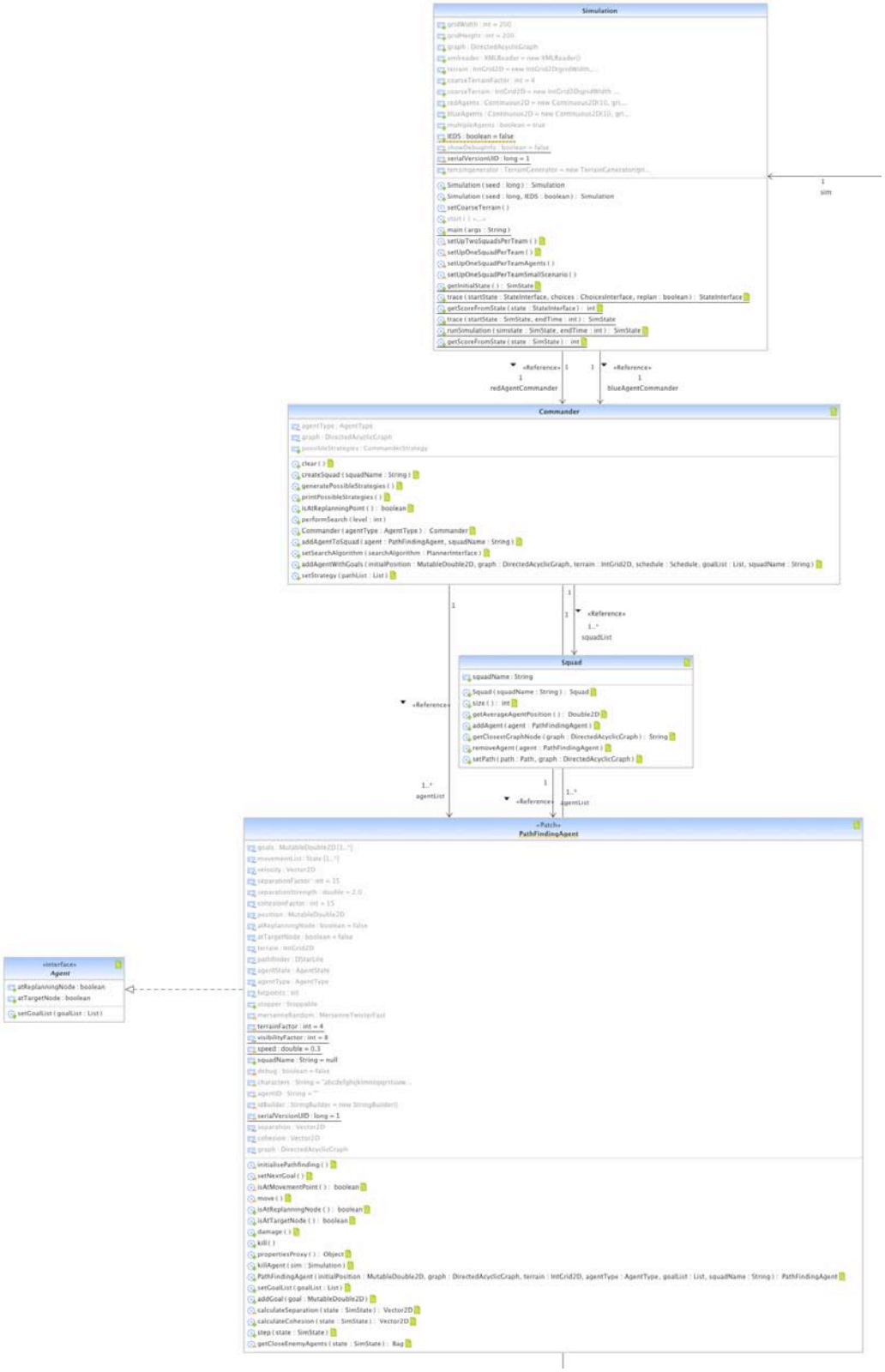
Other changes to the scenario would be interesting to test, such as increasing the number of squads per team, and increasing the number of agents per squad and determining what effect this has on the final payoffs. Changing the goal nodes payoffs could also be tested.

10.Glossary

Term	Description	Source
Algorithm	"A well specified sequence of steps to solve a particular problem that accepts an input and produces an output. Algorithms can be expressed in any language."	(Ali & Wasimi, 2007)
Stochastic	"Stochastic is often used as counterpart of the word 'deterministic', which means that random phenomena are not involved. Therefore, stochastic models are based on random trials."	(Origlio, 2011)
Monte-Carlo	"A method of using repeated random sampling to estimate the solutions to problems that are very hard or impossible to find analytically"	(Kleij, 2010)
Real Time Strategy (RTS) Game	"Real-time strategy (RTS) games are popular commercial computer games involving a fight for domination between opposing armies. There is no notion of whose turn it is to move."	(M. Chung, et al., 2005)
Two-person zero-sum games	"[A game] in which the two players have precisely opposite preferences. It is, therefore a game in which cooperation and collusion can be of no value. Any improvement for one player necessitates a corresponding loss for the other"	(Luce & Raiffa, 1989)
Nash Equilibrium	"A set of strategies, one for each player, such that no player has incentive to unilaterally change her action. Players are in equilibrium if a change in strategies by any one of them would lead that player to earn less than if they remained with their current strategy"	(Shor, 2005)

11. Appendixes

11.1. Appendix A



11.2. Appendix B

Linear Programming Solutions:

Total Solutions

With Replanning

		Blue		
		Random	Level 1	Level 2
Red	Random	2.656	2.312	2.281
	Level 1	2.652	2.392	2.305
	Level 2	3.90625	3.086	2.8125

No Replanning

		Blue	
		Random	Level 1
Red	Random	2.656	2.609
	Level 1	3.047	2.8125

Random vs. Random

D Node Probability:

Red team chooses D. Blue team chooses a different goal node.
$\frac{1}{4} * \frac{3}{4} * 2$

Red team chooses D. Blue team chooses D.
$\frac{1}{4} * \frac{1}{4} * 1$

E Node Probability:

Red team chooses E. Blue team chooses a different goal node.
$\frac{2}{4} * \frac{2}{4} * 3$

Red team chooses E. Blue team chooses F.
$\frac{2}{4} * \frac{2}{4} * 1.5$

F Node Probability:

Red team chooses F. Blue team chooses a different goal node.
$\frac{1}{4} * \frac{3}{4} * 5$

Red team chooses F. Blue team chooses F.
$\frac{1}{4} * \frac{1}{4} * 2.5$

Expected Red Score:

0.375+0.0625+0.75+0.375+0.9375+0.15625 = **2.656**

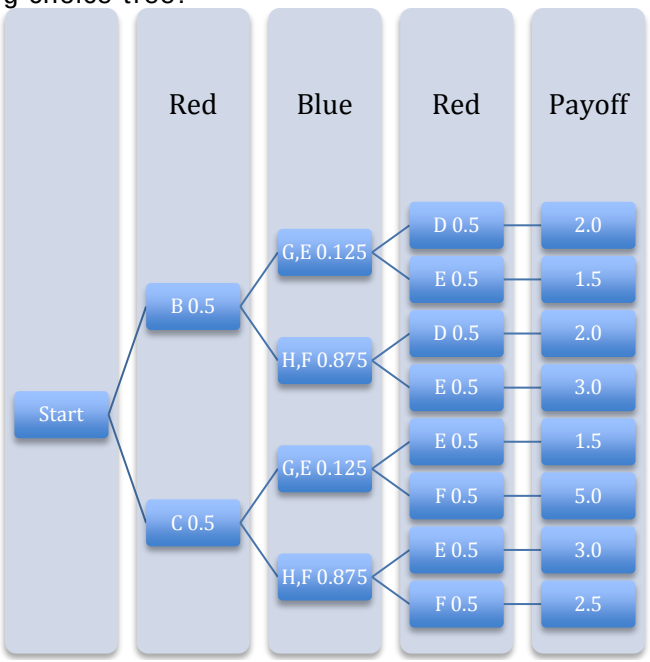
Random vs. No Replan Level 1

Red probabilities: 0.5
Blue initially solves a tree of level 1:

	Blue: I, G, D	Blue: I, G, E	Blue: I, H, E	Blue: I, H, F
Red: A, B, D	1	2	2	2
Red: A, B, E	3	1.5	1.5	3
Red: A, C, E	3	1.5	1.5	3
Red: A, C, F	5	5	5	2.5

Result: Blue chooses I,G,E 0.125 and I,H,F 0.875

Build the following choice tree:



Payoffs:
0.5*0.125*0.5*2.0+
0.5*0.125*0.5*1.5+
0.5*0.875*0.5*2.0+
0.5*0.875*0.5*3.0+
0.5*0.125*0.5*1.5+
0.5*0.125*0.5*5.0+
0.5*0.875*0.5*3.0+
0.5*0.875*0.5*2.5

Final Red Score: 2.609

NoReplan1 vs. Random

Red solves a single matrix:

	Blue: I, G, D	Blue: I, G, E	Blue: I, H, E	Blue: I, H, F
Red: A, B, D	1	2	2	2
Red: A, B, E	3	1.5	1.5	3
Red: A, C, E	3	1.5	1.5	3
Red: A, C, F	5	5	5	2.5

Red Score: 2.8125.
Percentages:
x1: 0%
x2: 62.5%
x3: 0%
x4: 37.5%

Combining this with random blue strategy choices:

D Node:
Red chooses D, Blue chooses other
0 * ¾ * 2
Red Chooses D, Blue chooses D
0 * ¼ * 1

E Node:
Red chooses E, Blue chooses other
0.625 * 2/4 * 3
Red chooses E, Blue chooses other
0.625 * 2/4 * 1.5

F Node:
Red chooses F, Blue chooses other
0.375 * ¾ * 5
Red chooses F, Blue chooses F
0.375 * ¼ * 2.5

Total Score:
(0 * ¾ * 2) + (0 * ¼ * 1) + (0.625 * 2/4 * 3) + (0.625 * 2/4 * 1.5) + (0.375 * ¾ * 5) + (0.375 * ¼ * 2.5)
Final red score = **3.046875**

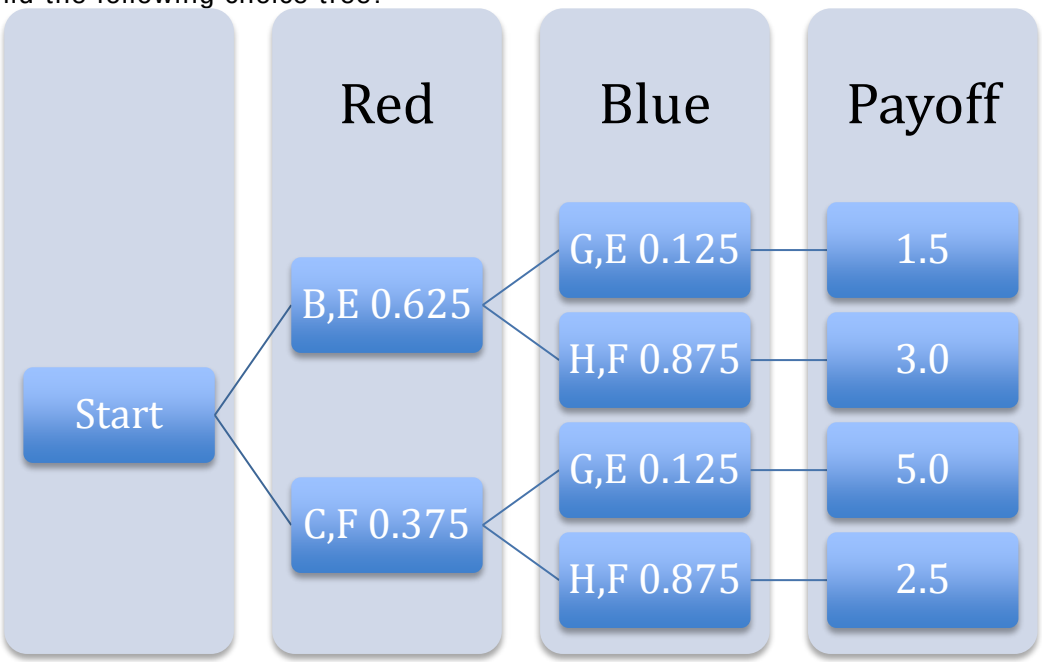
No Replan Level 1 vs. No Replan Level 1

Red and blue initially solve a tree of level 1:

	Blue: I, G, D	Blue: I, G, E	Blue: I, H, E	Blue: I, H, F
Red: A, B, D	1	2	2	2
Red: A, B, E	3	1.5	1.5	3
Red: A, C, E	3	1.5	1.5	3
Red: A, C, F	5	5	5	2.5

Results:
Red chooses A,B,E 0.625 and A,C,F 0.375
Blue chooses I,G,E 0.125 and I,H,F 0.875

Build the following choice tree:



Payoffs:
0.625*0.125*1.5+
0.625*0.875*3.0+
0.375*0.125*5.0+
0.375*0.875*2.5

Final red score: 2.8125

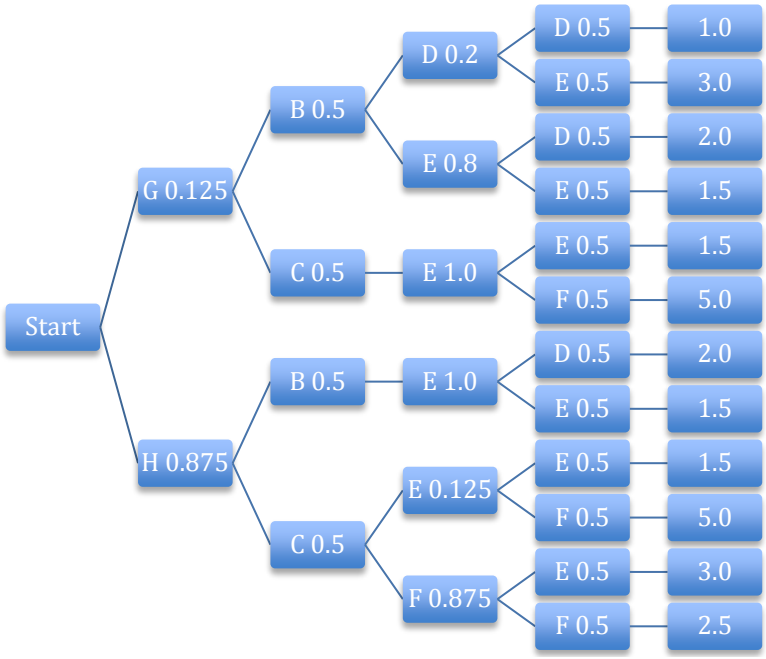
Random vs. Replan Level 1

Red probabilities = 0.5
Blue solves initial depth 1 matrix:

	Blue: I, G, D	Blue: I, G, E	Blue: I, H, E	Blue: I, H, F
Red: A, B, D	1	2	2	2
Red: A, B, E	3	1.5	1.5	3
Red: A, C, E	3	1.5	1.5	3
Red: A, C, F	5	5	5	2.5

Result: Blue chooses I,G,E 0.125 and I,H,F 0.875.

Blue can replan when they reach the replanning points.



Payoffs:
0.125*0.5*0.2*0.5*1.0+
0.125*0.5*0.2*0.5*3.0+
0.125*0.5*0.8*0.5*2.0+
0.125*0.5*0.8*0.5*1.5+
0.125*0.5*1.0*0.5*1.5+
0.125*0.5*1.0*0.5*5.0+
0.875*0.5*1.0*0.5*2.0+
0.875*0.5*1.0*0.5*1.5+
0.875*0.5*0.125*0.5*1.5+
0.875*0.5*0.125*0.5*5.0+
0.875*0.5*0.875*0.5*3.0+
0.875*0.5*0.875*0.5*2.5

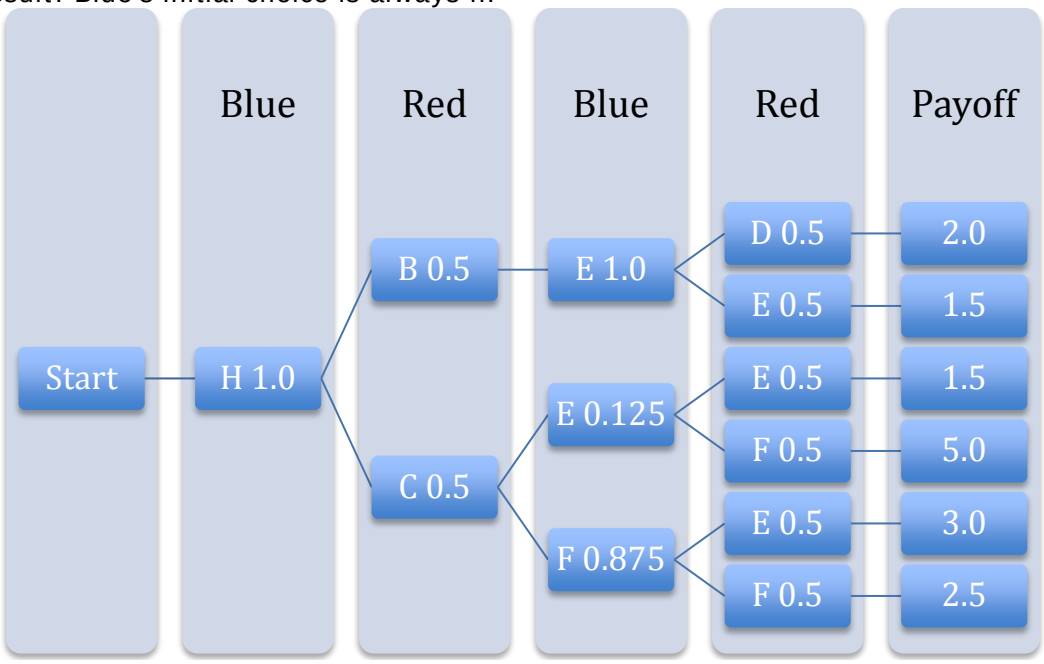
Total red payoff = 2.312

Random vs. Replan Level 2

Red probabilities = 0.5
Blue Initial tree:

1.8	2.0
5	2.8125

Result: Blue's initial choice is always h.



Payoffs:
1.0*0.5*1.0*0.5*2 = 0.5
1.0*0.5*1.0*0.5*1.5 = 0.375
1.0*0.5*0.125*0.5*1.5 = 0.046875
1.0*0.5*0.125*0.5*5 = 0.15625
1.0*0.5*0.875*0.5*3 = 0.65625
1.0*0.5*0.875*0.5*2.5 = 0.546875

Total red payoff = 2.281

Level 1 Replan vs. Random

Blue probabilities: 0.5
Red solves a level 1 tree initially:

	Blue: I, G, D	Blue: I, G, E	Blue: I, H, E	Blue: I, H, F
Red: A, B, D	1	2	2	2
Red: A, B, E	3	1.5	1.5	3
Red: A, C, E	3	1.5	1.5	3
Red: A, C, F	5	5	5	2.5

Result:
Red chooses A,B,E 0.625
Red chooses A,C,F 0.375

Payoffs:
0.625*0.5*0.6*0.5*1.0+
0.625*0.5*0.6*0.5*2.0+
0.625*0.5*0.4*0.5*3.0+
0.625*0.5*0.4*0.5*1.5+
0.625*0.5*1.0*0.5*2.0+
0.625*0.5*1.0*0.5*2.0+
0.375*0.5*1.0*0.5*5.0+
0.375*0.5*1.0*0.5*5.0+
0.375*0.5*0.625*0.5*1.5+
0.375*0.5*0.625*0.5*3.0+
0.375*0.5*0.375*0.5*5.0+
0.375*0.5*0.375*0.5*2.5

Expected red payoff:
2.65234375

Level 1 Replan vs. Level 1 Replan

Red and Blue both initially solve a tree of level 1

	Blue: I, G, D	Blue: I, G, E	Blue: I, H, E	Blue: I, H, F
Red: A, B, D	1	2	2	2
Red: A, B, E	3	1.5	1.5	3
Red: A, C, E	3	1.5	1.5	3
Red: A, C, F	5	5	5	2.5

Red chooses B 0.625, C, 0.375
Blue chooses G 0.125, H 0.875

Payoffs:
0.625*0.125*0.6*0.2*1.0+
0.625*0.125*0.6*0.8*2.0+
0.625*0.125*0.4*0.2*3.0+
0.625*0.125*0.4*0.8*1.5+
0.625*0.875*1.0*1.0*2.0+
0.375*0.125*1.0*1.0*5.0+
0.375*0.875*0.625*0.125*1.5+
0.375*0.875*0.625*0.875*3.0+
0.375*0.875*0.375*0.125*5.0+
0.375*0.875*0.375*0.875*2.5

Final red score: 2.3916

Level 1 Replan vs. Level 2 Replan

Red initially solves a tree of depth 1:

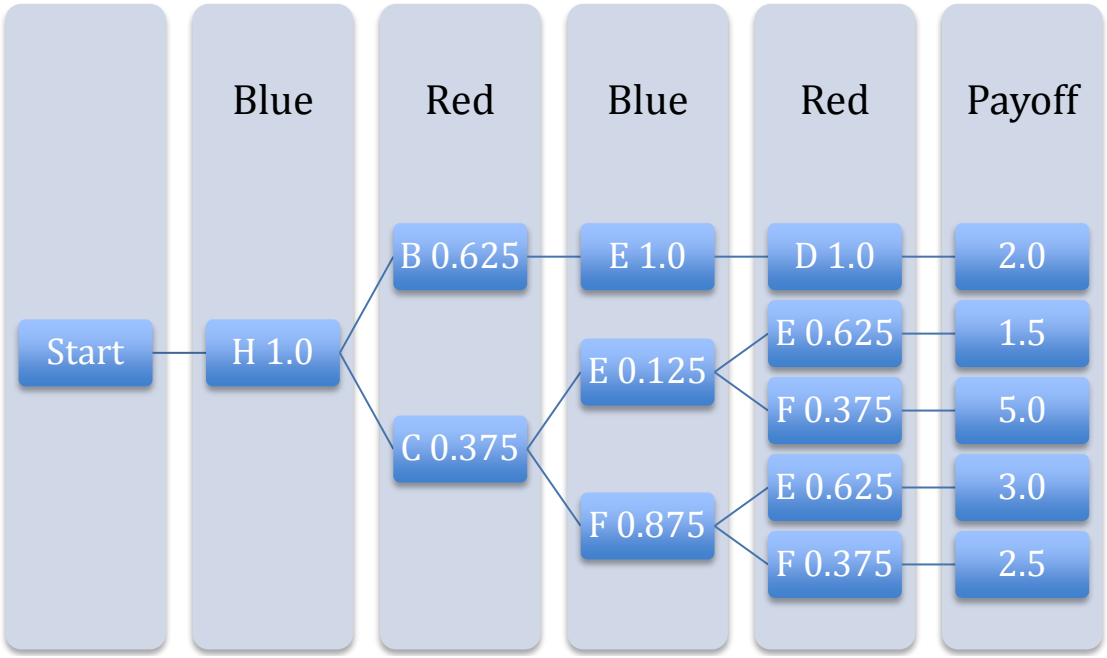
	Blue: I, G, D	Blue: I, G, E	Blue: I, H, E	Blue: I, H, F
Red: A, B, D	1	2	2	2
Red: A, B, E	3	1.5	1.5	3
Red: A, C, E	3	1.5	1.5	3
Red: A, C, F	5	5	5	2.5

Red chooses B,E 0.625 and C,F 0.375

Blue initially solves a tree of depth 2:

1.8	2.0
5	2.8125

Blue chooses H 100% of the time



Payoffs:
0.625*1.0*1.0*2.0+
0.375*0.125*0.625*1.5+
0.375*0.125*0.375*5.0+
0.375*0.875*0.625*3.0+
0.375*0.875*0.375*2.5

Final red score: 2.3046875

Level 2 Replan vs. Random

Blue Probabilities: 0.5
Red Solves an initial tree of level 2:

	Blue: G	Blue: H
Red: B	1.8	2.0
Red: C	5	2.8125

Result: Red chooses C 100% of the time.

Payoffs:
0.5*1.0*0.5*5.0+
0.5*1.0*0.5*5.0+
0.5*0.625*0.5*1.5+
0.5*0.625*0.5*3.0+
0.5*0.375*0.5*5.0+
0.5*0.375*0.5*2.5

Final Red Score: 3.90625

Level 2 Replan vs. Level 1 Replan

Red solves a tree of level 2:

	Blue: G	Blue: H
Red: B	1.8	2.0
Red: C	5	2.8125

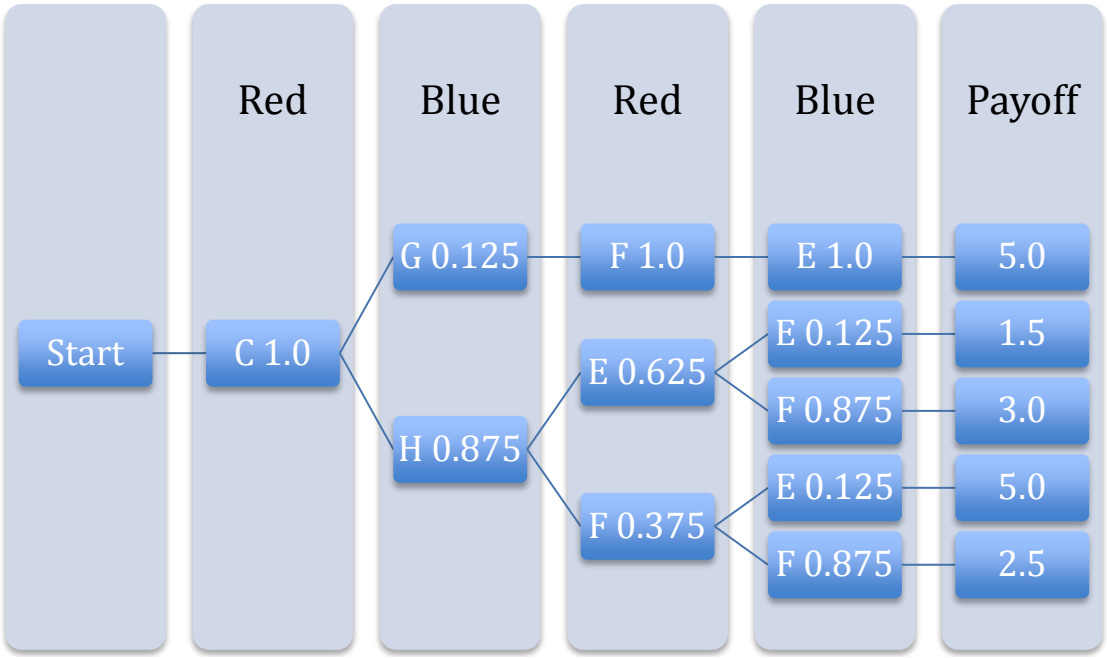
Results: Red chooses C 100% of the time

Blue solves a tree of level 1:

	Blue: I, G, D	Blue: I, G, E	Blue: I, H, E	Blue: I, H, F
Red: A, B, D	1	2	2	2
Red: A, B, E	3	1.5	1.5	3
Red: A, C, E	3	1.5	1.5	3
Red: A, C, F	5	5	5	2.5

Blue chooses G 0.125 and H 0.875

Decision Tree:



Payoffs:
0.125*1.0*1.0*5.0+
0.875*0.625*0.125*1.5+
0.875*0.625*0.875*3.0+
0.875*0.375*0.125*5.0+
0.875*0.375*0.875*2.5

Final red score: 3.086

Level 2 Replan vs. Level 2 Replan

Both teams solve a tree of depth 2

	Blue: G	Blue: H
Red: B	1.8	2.0
Red: C	5	2.8125

Red: Chooses C 100% of the time
Blue: Chooses H 100% of the time

Final red score: 2.8125

12.References

Alba, E., & Cotta, C. (2006). Evolutionary Algorithms *Handbook of Bioinspired Algorithms and Applications* (pp. 3-19): Chapman & Hall/CRC.

Ali, A. B. M. S., & Wasimi, S. A. (2007). *Data Mining: Methods and Techniques*: Thompson - Nelson Australia.

Auer, P. (2003). Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res., 3*, 397-422.

Balan, G. C., Cioffi-Revilla, C., Luke, S., Panait, L., & Paus, S. (2003). *MASON: A Java Multi-Agent Simulation Library*. Paper presented at the Agent 2003 Conference.

Balla, R.-K., & Fern, A. (2009). *UCT for Tactical Assault Planning in Real-Time Strategy Games.* Paper presented at the International Joint Conference on Artificial Intelligence, Pasadena, California, USA.

Best, J. W., & Kahn, J. V. (1989). *Research in Education* (6th ed.). New Jersey: Prentice-Hall Inc.

Bonaccorsi, A., & Rossi, C. (2003). Why Open Source software can succeed. *Research Policy, 32*(7), 1243-1258.

Branavan, S. R. K., Silver, D., & Barzilay, R. (2011). *Non-linear monte-carlo search in Civilization II.* Paper presented at the IJCAI.

Cantwell, L. C. G. L. (2003). *Can two person zero sum game theory improve military decision-making course of action selection?*

Cave, J. (1987). *Introduction to Game Theory*. Santa Monican, California: Rand.

Cazenave, T. (2009). *Nested Monte-Carlo Search.* Paper presented at the 21st International Joint Conference on Artificial Intelligence (IJCAI09), San Francisco.

Cazenave, T., & Jouandeau, N. (2009, 23-29 May 2009). *Parallel Nested Monte-Carlo search.* Paper presented at the Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on.

Chaslot, G., Bakkes, S., Szita, I., & Spronck, P. (2008). *Monte-Carlo tree search : a new framework for game AI*. Paper presented at the The 4th Artificial Intelligence and Interactive Digital Entertainment Conference.

Chaslot, G., Saito, J., Bouzy, B., Uiterwijk, J., & van der Herik , H. (2006). *Monte-Carlo Strategies for Computer Go.* Paper presented at the 18th Belgian-Dutch Conference on Artificial Intelligence.

Chaslot, G., Winands, M., Uiterwijk, J., Herik, H. J. v. d., & Bouzy, B. (2007). *Progressive strategies for Monte-Carlo tree search.* Paper presented at the 10th Joint Conference on Information Sciences, Salt Lake City, USA.

Chaslot, G. M. J.-B. (2010). *Monte-Carlo Tree Search.* Maastricht, Maastricht.

Choo, C. S., Chua, C. L., & Tay, S. H. V. (2007). *Automated red teaming: a proposed framework for military application.* Paper presented at the GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, New York, NY, USA.

Chung, M., Burro, M., & Schaeffer, J. (2005). *Monte-Carlo planning in RTS games.* Paper presented at the IEEE Symposium on Computational Intelligence and Games.

Chung, S. W., Choo, C. S., Martinez-Tiburcio, F., & Lin, K. (2006). *Applying Automated Red Teaming in a Maritime Scenario*. Paper presented at the International Data Farming Workshop 14 (IDFW14).

Ciancarini, P., & Favini, G. P. (2010). Monte Carlo tree search in Kriegspiel. *Artificial Intelligence, 174*(11), 670-684.

Cockburn, A. (2000). *Agile Software Development*: Addison-Wesley.

Crawford, S., & Stucki, L. (1990). Peer review and the changing research record. [10.1002/(SICI)1097-4571(199004)41:3<223::AID-ASI14>3.0.CO;2-3]. *Journal of the American Society for Information Science, 41*(3), 223-228.

Creswell, J. W. (2009). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (Third ed.). Thousand Oaks, California: SAGE Publications.

Decraene, J., Zeng, F., Low, M. Y. H., Zhou, S., & Cai, W. (2009, 9/3/2011). *Research Advances in Automated Red Teaming.* Paper presented at the Spring Simulation Multi-Conference (SpringSim), Orlando, FL, USA.

Duong, D. (2008). *Strategic Data Farming*. Retrieved from http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA489884&Location=U2&doc=GetTRDoc.pdf.

Duong, D., Brown, R., Schubert, J., McDonald, M., Makovoz, D., & Singer, H. (2010). *Strategic Data Farming of Military and Complex Adaptive Simulations for COA Optimization*. Paper presented at the International Data Farming Workshop 20 (IDFW20).

Fern, A., & Lewis, P. (2011). *Ensemble Monte-Carlo Planning: An Empirical Study.* Paper presented at the Twenty-First International Conference on Automated Planning and Scheduling, University of Freiburg.

Kleij, A. A. J. v. d. (2010). *Monte Carlo Tree Search and Opponent Modeling through Player Clustering in no-limit Texas Hold'em Poker.* University of Groningen, Groningen, The Netherlands.

Kocsis, L., & Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. In J. Fürnkranz, T. Scheffer & M. Spiliopoulou (Eds.), *Machine Learning: ECML 2006* (Vol. 4212, pp. 282-293-293): Springer Berlin / Heidelberg.

Laviers, K. (2010). *Multi-agent plan adaptation using coordination patterns in team adversarial games*. Paper presented at the Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1.

Lin, Y. (2003). Game Trees. Retrieved from http://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html
Luce, R. D., & Raiffa, H. (1989). *Games and Decisions: Introduction and Critical Survey*: Dover Publications.
Macal, C. M., & North, M. J. (2010). Tutorial on agent-based modelling and simulation. *Journal of Simulation, 4*, 151-162.
Markham, T., & Payne, C. (2001, 2001). *Security at the network edge: a distributed firewall architecture.* Paper presented at the DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01. Proceedings.
Mehat, J., & Cazenave, T. (2010). Combining UCT and Nested Monte Carlo Search for Single-Player General Game Playing. *Computational Intelligence and AI in Games, IEEE Transactions on, 2*(4), 271-277.
Millington, I., & Funge, J. (2009). *Artificial Intelligence for Games* (2 ed.). Burlington: Morgan Kaufmann.
Morris, P. (1994). *Introduction to Game Theory*: Springer.
Nettles, A. B. (2010). *The President Has No Clothes: The case for broader application of red teaming within homeland security.* Monterey, California.
Origlio, V. (2011). Stochastic. Retrieved from http://mathworld.wolfram.com/Stochastic.html
Sailer, F., Buro, M., & Lanctot, M. (2007, 1-5 April 2007). *Adversarial Planning Through Strategy Simulation.* Paper presented at the Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on.
Schneider, W. (2003). *The Role and Status of DoD Red Teaming Activities*. Retrieved from http://www.fas.org/irp/agency/dod/dsb/redteam.pdf.
Shor, M. (2005). Nash Equilibrium. Retrieved from http://www.gametheory.net/dictionary/NashEquilibrium.html
Szita, I. n., Chaslot, G., & Spronck, P. (2010). Monte-Carlo Tree Search in Settlers of Catan
Advances in Computer Games. In H. van den Herik & P. Spronck (Eds.), (Vol. 6048, pp. 21-32): Springer Berlin / Heidelberg.
Takeuchi, S., Kaneko, T., & Yamaguchi, K. (2010). Evaluation of Game Tree Search Methods by Game Records. *Computational Intelligence and AI in Games, IEEE Transactions on, 2*(4), 288-302.
Upton, S. C., Johnson, S. K., & McDonald, M. J. (2004). *Breaking Blue: Automated Red Teaming Using Evolvable Simulations*. Paper presented at the GECCO: Proceedings of the 6th annual conference on Genetic and evolutionary computation.
Zafra, P. (2010). Linear Programming and Two-Person Zero-Sum Games. Kean University.