Search and Recall for RTS Tactical Scenarios

Jason Traish, James Tulip and Wayne Moore [1]

Abstract. The success of a Real-Time Strategy agent is heavily dependent on its ability to respond well to a large number of diverse tactical situations. We present a novel method of tactical decision making called Search and Recall (S&R) which is a hybrid of Search and Case Based Reasoning (CBR) methods. S&R allows an agent to learn and retain strategies discovered over the agent's history of play, and to adapt quickly in novel circumstances.

The sense of memory that S&R provides an RTS AI agent allows it to improve its performance over time as better responses are discovered. S&R demonstrates an minimum win rate of 92% in standard scenarios evaluated in this paper.

S&R decouples search from the main game loop which allows arbitrary computational complexity and execution time for search simulations. Meanwhile in-game decision making is based on CBR and remains fast and simple.

This paper presents an S&R model which extends the ability of an RTS AI agent to deal with complex tactical situations. These situations include special unit abilities, fog of war, path finding, collision detection and terrain analysis.

1 Introduction

Real-time strategy (RTS) games are a popular genre of commercial games that require substantial practice, skill and experience to master. In order to conquer an opponent, a player must manage a number of in-game systems with precision, using a large number of possible commands. In-game systems include research, economics, exploration, managing an army, and executing a strategy with the potential of defeating the opponent's strategy. On top of all this complexity a player is expected to complete all these tasks in a real-time environment of uncertainty.

The number of in-game systems and possible commands illustrate the complexity of the RTS genre and form the basis of it's appeal to players and researchers alike. Developing an RTS agent poses many challenges that are not present in traditional strategy board game environments such as GO and Chess. In particular, the large number of units and possible commands, the uncertain environment, the effect of terrain, and the real-time execution constraints are unique to the computer based RTS genre.

It is very difficult to write scripted agents that vary their responses in different situations. This results in easily exploited AI agents which fail to give experienced players an enjoyable challenge.

Case Based Reasoning is one approach that has been used to create adaptive RTS AI agents [1, 2, 8, 9]. Search based methods have also became a point of interest to the RTS research community [3, 4, 12]. However, both approaches have intrinsic limitations.

[1] Charles Sturt University, Mining Lab, Australia, email: {jtraish & jtulip} @csu.edu.au & wmoore@lisp.com.au

1.1 Case Based Reasoning and Search in RTS AI

Case based reasoning (CBR) methods have been used successfully to create adaptive RTS agents. In general, such methods store plans with an associated game state and use this data to reason about future encounters.

Aha et al. [1] demonstrated a CBR agent capable of identifying and adapting to a randomly selected opponent which demonstrated good results. Their agent relied on the availability of a set of pre-generated responses, each capable of winning against an opponent from a given position.

McGinty et al. [8] improved CBR approaches by changing the structuring and case retrieval approach, leading to significantly better results. Their agent demonstrated a high win rate in experiments with imperfect information. Other CBR methods have focused on the use of recorded human player interactions to make decisions [2, 9].

However, while CBR has been successful in creating adaptive RTS agents, they face a number of challenges. Responses derived from human players can be of inconsistent quality due to the diversity of human player skills and the nature of human play. Standard CBR approaches are also ill equipped to make decisions if there is no similar recorded context.

As a result, search based methods, and in particular Monte Carlo simulations have gained the interest of the RTS AI research community [3, 4, 5, 6, 12]. Search based methods enable an agent to adapt in real-time to whatever circumstances it is currently facing, assuming the simulator can correctly predict the outcome of a given response action. Significant research on adaptive agents using search based techniques has been performed in the context Chess and GO [7] and the application of such techniques to RTS games is an attractive prospect.

However, complexities such as path finding and collision detection are required for an agent to appropriately handle commercial game type tactical situations. Such situations include moving units in a environment affected by terrain, or engaging armies of many varied unit types, some with special abilities.

The complexity inherent in commercial RTS games places huge computational demands on the simulations required to perform a search for a tactical solution. For this reason most of the published search simulation approaches are very simple relative to the demands of fully realised commercial game agents and ignore issues such as terrain, path finding, and collisions between units.

The problem is that simulations conducted within the game loop are heavily constrained to execute in an extremely limited amount of time, due to the demands of other aspects of the game loop such as animation and rendering.

In the rest of this paper we present a hybrid search/CBR approach called Search and Recall (S&R) which enables simulations capable of dealing with commercial grade RTS game complexity, while offering CBR level in-game performance. We demonstrate these capabilities in the context of Starcraft Broodwar; a commercial RTS which has become a popular RTS AI research platform.

The main contribution of this work is to demonstrate the utility of responses generated using Search simulations as recorded responses in a CBR-like database. The technique was inspired by case base reasoning literature that focused on constructing databases using player responses [10]. We also demonstrate an approach for making computationally intensive search simulations feasible in the context of a real-time game.

2 Search and Recall - Overview

Search and Recall (S&R) is a novel method of tactical decision making which is a hybrid of Search and Case Based Reasoning (CBR) methods. It allows an agent to learn and retain strategies discovered over the agent's history of play, and to adapt quickly in novel circumstances.

Similarly to CBR methods, S&R uses a database of previously discovered successful responses associated with a collection of identified game states. S&R agents use these responses to quickly identify a solution without extensive simulation within the game loop. However, unlike other CBR methods, S&R does not populate it's response database with a static set of game states identified from previously played games. Rather, it populates the database dynamically with the results of search simulations conducted in response to actual game states encountered during play.

By combining the adaptive learning of MCS with the memory of CBR, S&R allows an agent to improve the quality of its responses over the course of multiple games.

In essence, we decouple the search tasks from the game loop by allowing them to execute asynchronously and in parallel with the game loop. Searches are pushed into concurrent threads, allowing them to take as long as necessary without delaying game rendering. The agent makes its decisions based on its current database of solutions, and the search tasks update that database asynchronously with the results of new simulations based on possible responses to the current game state. As many searches can be carried out as are appropriate to the CPU resources available to the game.

Search time is limited only by the length of a game or an arbitrary stopping condition, and is substantially longer than the 5ms generally allocated for an agent's decision making process within the standard game loop. The downside is that the longer it takes to evaluate potential decisions the more likely it is that the response will come too late to be useful in the current situation. However, the next time a similar situation is encountered, the simulation results will be available in the CBR database (response library) ready for near instant access.

Search results are used to update a CBR like database as they become available, and the AI task within the game loop is reduced to selecting the appropriate response as in a conventional CBR system.

We apply this architecture in the context of the commercial game Starcraft Broodwar. Starcraft is an immensely popular and sophisticated RTS game, famous for its balanced asymmetric game play and status as a professional spectator sport in Korea. Starcraft Broodwar is a version of Starcraft for which an external programming interface has been developed called the Brood War API (BWAPI). The availability of BWAPI has made Broodwar an attractive platform for RTS AI research.

3 Search and Recall - Agent Components

The S&R agent is composed of a recall-playback component (RPC), a search component (SC), and a response library (RL). This basic architecture is illustrated in Figure 1. The RPC component acts as coordinator for the agent and interacts with the BWAPI interface. As soon as a Broodwar game begins the S&R agent starts the recall-playback component and initialises the search component with a number of threads.

3.1 Recall/Playback Component (RPC)

The RPC matches the current game state against the game states currently recorded in the response library. Game states in the database are identified by a simplified descriptor containing only the number and types of unit present.

The RPC then retrieves the response associated with the current game state from the response library. The response associated with a game state is always the most favourable response generated by the search simulations carried out in the search component. If no matching game state is found, the RPC assigns random behaviours to the agent's units. If a response was loaded earlier from a previous game state then those previous behaviours are not changed.

The RPC has a simulator similar to those being used for searching. It uses this to simulate a single time step using the unit actions specified in the response. This step is carried out in order to map from the actions specified in the response to a set of Broodwar commands that must be issued through the BWAPI interface. The raw actions that the units must perform are recorded (e.g. move[x,y], attack[unitId]) and forwarded to the BWAPI.

Although games states are identified in the RL only by the number and type of units present, actual game state is defined with considerably more information on unit positions, current unit states, what projectiles have been created, which units are damaged, and which weapons have entered their cool down periods. All of this information is captured from the BWAPI and sent through to the search component (SC) in addition to the number and type of units present in the scenario. The RPC buffers these changes in actual game state for the SC, updating the information used by that component as a basis for simulation only after 200 simulations have completed. This allows a sufficient number of searches associated with a particular game state to complete to be useful in subsequent games.

The execution of the RPC is constrained to take less than 5ms per frame since it executes as a part of the main game loop. This constraint is easily achieved since the simulator used to calculate the BWAPI commands simulates only a single time step.

3.2 Search Component (SC)

The search component is represented in Figure 1 as the Concurrent Search Simulators (CSS). It consists of a number of search threads which repeatedly run simulations for the combat scenario utilizing the current actual game state, a simulator engine, and a set of actions assigned to each unit in the scenario.

At the beginning of a simulation, each search thread is given the current identifying game state (unit numbers and types) as well as information describing the actual game state (terrain, unit positions, unit health, current unit action states, etc).

We randomly assign behaviours to each unit for each simulation so we can evaluate the effect of utilising different tactics on the outcome of a battle . If simulations complete quickly, many different

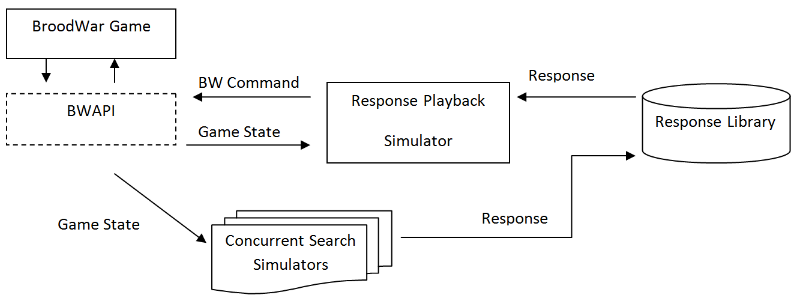

Figure 1. Search and recall agent process

possible outcomes can be calculated and used to update the solution available to the RPC before the time allotted to its execution within the game loop (5ms) expires. However, if it takes longer to simulate an outcome than the time Broodwar allows, then the result will not be available to the RPC during the current game loop. This results in the game agent taking longer to respond to a game state in real time, although simulation results do become available to the RPC over the next few game loop cycles as simulations complete.

When a simulation completes, the quality of the response is calculated as the total health percentage of the remaining allied units at the end of simulation. A quality of 0 is given for prediction in which all allied units are killed. This formula favours victories with lower casualties, and ranks all losses equally.

As in [6], the simulator is a mathematical model of the combat mechanics implemented in Broodwar, that allows simulations to be run without any frame rate derived speed limitations. As such, it is not an exact model of the combat mechanics implemented in Broodwar.

3.2.1 Simulators

Asynchronous execution of search allows the complexity and execution cost of the simulation engine used to be increased arbitrarily. In this work we explore the effect of increasing the complexity of the simulation engine used by evaluating the performance of two different simulators. These are:

1) Basic Simulator: This simulator handles unit health, shields, healing, attacking, and movement without collision or path finding. It can complete up to 2000 combat simulations per second per thread.
2) Complex Simulator: This simulator handles unit health, shields, healing, and attacking. However, the movement function detects collisions and finds paths around obstacles such as terrain and other units. Influence maps from [11] have also been integrated to support a 'kiting' behavior which has been added to the list of available behaviours. This simulator can complete only up to 200 combat simulations per thread per second.
   Kiting is a highly successful behaviour that fast moving ranged units can use against slower units. Kiting is the act of attacking an enemy unit and then moving away while reloading.

3.2.2 Response Divergence

A response grows stale the longer it is in effect. This is due to differences in mechanics between the Broodwar game and the simulator that even a very sophisticated model will find challenging to eliminate, in particular because there are random elements built into the Broodwar game engine. We call the differences between the simulated outcome and what actually happens in Broodwar as divergence. Divergence represents the cumulative error between the game states of the simulation and Broodwar as time passes.

Different game systems suffer differing amounts of divergence. While systems like health regeneration and attack damage are straight forward, other components such as attack cool downs are randomisied slightly, introducing small changes in combat outcomes. The precise mechanics of other systems such as path finding are unknown and this also increases the divergence of simulations from actual game encounters. Furthermore, an opponent model is not necessarily a precise model of the Broodwar AI, and this also leads to a large amount of divergence. Finally, the actual precise game state used to drive the search simulation that generated the response recorded in the database may differ from the precise current game state. If differences in precise unit location and health affect the outcome of the battle, divergence will occur.

3.2.3 Opponent Models

In order to combat the effects of divergence, solutions that generalise well are sought. The simulation outcome is heavily dependent on the strategy used by the opponent, so we attempt to find generalized solutions by taking the minimum of the solution quality score over a small set of opponent models. This favours the selection of robust strategies that are successful against a variety of opponent models for the response library (RL). In the current work this set of opponent models contains only 2 strategies; one using an 'Attack Weakest strategy, and the other using an 'Attack Closest' strategy.

3.2.4 Unit Behaviours and Grouping

A behaviour describes what action the unit should take in any given circumstance. A behaviour consists of a series of actions which a unit executes in sequence, moving on to the next action when the previous action is complete or appropriate conditions are met. For each behaviour we identify a primary action, and a secondary action which is applied if multiple targets are identified for the primary action. For example, if 'Attack Weakest' is the primary action, and all enemy units have the same health, then the secondary action 'Attack Closest', is applied. Behaviours are described in Table 1.

In order to allow the S&R agent some flexibility in terms of choosing and targeting particular units or types of enemy units, we provide the agent with the ability to separate the enemy into groups.

When setting up a simulation, not only are a random set of behaviours assigned to the agent's units, but the enemy is divided into 4 random groups. Actions are then made specific to groups. For example, the generic "Attack Closest" behaviour becomes "Attack Closest in Group 1". Grouping allows the agent to create plans that can focus fire individual or groups of units. This greatly increases the degree of freedom with which the agent can respond to situations.

3.3 Response Library Component (RLC)

The S&R agent receives its recall ability from the use of the response library. The response library is responsible for the storage and communication of the best recorded responses from the search simulations. The database is updated asynchronously by the SC, and queried from within the game loop by the RPC. It acts as a constantly growing and improving database of best seen responses to recorded tactical situations.

3.3.1 Game State and Response Descriptors

Preliminary testing identified that actual game state needed to be generalized for successful game state matching to occur. Furthermore, only a small number of game state attributes were required for the agent to adapt competently. Hence, the attributes used to identify game state within the RLC include only the number and type of each unit involved in the current scenario. Adding more detailed game state descriptors such as those describing unit health or position causes an explosion of possible states, this drastically shortens the time that a game remains in a particular state, and makes it difficult to match the current game state with a state recorded in the response library.

Describing game state by only the number and type of units involved results in relatively stable states that recur sufficiently frequently to make matching effective, and balances the frequency of response adaption. This approach effectively forces the chosen response to change only to when units are removed from or added to the game.

In addition to the game state information that is used as a key in the response library, each entry in the response database records the behaviour assigned to each unit, and the groupings assigned to the enemy units.

Response behaviours do not correspond with BWAPI commands: they need to be mapped into BWAPI commands by the simulator associated with the RPC.

4 Experimental Setup

The following experiments contain four tactical scenarios that an agent cannot resolve with a singular response. These are illustrated in Figure 2 and listed below:

A) 3 Zealots vs 3 Vultures (Attack Closest agent): This scenario pits 3 fast ranged units (Vultures) controlled by the agent against 3 slow close attack units (Zealots). This scenario favours the kiting strategy as it is extremely difficult to solve without it.

B) 6 Fast Zerglings vs 2 Dragoons (Attack Closest agent): This scenario pits 2 strong ranged units (Dragoons) controlled by the agent against 6 fast close attack units (Fast Zerglings). Once again a kiting solution is favoured, but far more precision is required to make this work.

C) 3 Zealots and 3 Dragoons vs 3 Zealots and 3 Dragoons (Default AI): This is a symmetrical scenario pitting ranged (Dragoons) and close attack (Zealots) units against each other. Precise control over unit attacks which enemy unit as well as unit placement is required to be successful.

D) 8 Dragoons vs 8 Dragoons (Default AI): Once again this is a symmetrical scenario that pits equal numbers of ranged units against each other. Control of attack strategy is important in this scenario, but unit placement is less important than in Scenario C.

The experimental setup is based on work by [5] although the experimental setups for scenarios A and B differ from Churchill's implementation. Due to problems encountered with the BroodWar AI's default behaviour it was replaced with a scripted agent designed to constantly attack the closest unit.

Each scenario is run against a particular configuration of the S&R agent for a total of 200 games at an acceleration of 5ms per frame. This is necessary since due to stochastic variation between games, the outcome of an actual game is not completely deterministic. The scores recorded in Table 2 are defined by the following function to the nearest percentage.

$$Score = (wins + draws/2)/200$$

Our experiment compares several different configurations of the SR agent. The performance of the basic and the complex simulator engine are compared in two modes: in pure search mode (ie without access to any stored responses), and in combined search and recall mode (with access to stored responses). This tests whether there is any advantage in retaining results from earlier simulations. For comparison purposes, the performance of two scripted agents was also evaluated: one based on an 'Attack Closest' strategy, and another which favours Kiting. Each configuration or agent is tested on the four scenarios listed above.

For the S&R agents, each configuration is initialised with a new empty response library at the beginning of the evaluations for all scenarios. All recorded responses are generated by simulations run during the actual games.

All S&R experiments utilise 4 threads within the SC for running simulations. Each search was limited to 2000 time steps although this number of steps was never reached. The results of the experiments are shown in Table 2.

5 Results and Discussion

The results of the experiments for the scripted agents show clearly that to do well in all four scenarios requires adaptive agent behaviour. The 'Attack Closest' scripted agent performs poorly in scenarios A and B, but is successful in scenarios C and D while the reverse is the case for the 'Kiting' scripted agent.

Results for the simple simulator, which does not have a kiting behaviour available are similar to the 'Attack Closest' scripted agent. This illustrates the importance of the simulator model containing a set of behaviours sufficient to cover what is required in a scenario.

On the other hand, results in scenarios A and B for the complex simulator show that agent clearly discovered and utilized the appropriate kiting behaviour. Results in Scenario A are stronger than in Scenario B, likely because the large speed difference between Vultures and Zerglings makes a wide range of successful kiting solutions relatively easy to find. In Scenario B, if the Dragoons performed a suboptimal action for even a small period they would lose to the larger numbers of Zerglings.

Table 1. Behaviour Descriptions

Behaviour	Primary Function	Secondary Function	Condition
G1, G2, G3 and G4	Attack unit of least health in group X	Attack closest unit in group X	No units in group X
Attack Closest	Attack closest unit	Attack unit of least health	N/A
Attack Wounded	Attack unit of least health	Attack closest unit	N/A
Kite	Attack unit of least health in range when ready to fire	Move away from all enemies and terrain	N/A

Table 2. Experiment 1 Results. S&R: Search and Recall. IM: Influence Map.

Setup	Churchill Search	Search	S&R	Search (IM)	S&R (IM)	Attack Closest	Kiter
A	0.81	0	0	0.96	1.00	0	1.00
B	0.65	0	0	0.65	0.92	0	1.00
C	0.95	0.95	0.80	0.76	0.94	0.77	0.26
D	0.96	1.00	1.00	1.00	1.00	0.97	0.14

The results for the complex simulator with recall enabled are better than for search alone, indicating that the recall capability provides a considerable advantage. The advantage conferred by the recall ability is much greater in Scenario B than in Scenario A. This suggests that the advantage of accumulating knowledge in the response database is greatest when solutions are relatively exact, and the exploration of the solution space is relatively slow.

Results for the simple simulator are equivalent or better than the complex simulator for scenarios C and D. This indicates that the range of behaviours available to the simple simulator are sufficient in these scenarios, and that the complexities introduced for the complex simulator have little impact in these scenarios. This result is not terribly surprising since the influence map affects only the kiting behaviour which is not necessary in these scenarios, and the close ranged combat and lack of terrain features in these scenarios reduces the impact of the path finding capability of the complex simulator. Given these considerations, it may be that the much greater number of simulations that the simple simulator can perform (2000 vs 200 per second) allows it to find better solutions than the complex simulator.

Results for scenario C yield are the most varied. The winning solutions for this scenario required more complex behaviours than in the other scenarios. Scenario C is similar in some respects to Scenario B with its rigorous success requirements.

Results for the complex simulator in Scenario C show a large difference between search only and combined search and recall. Once again it appears that that the recall capability becomes a significant advantage when solutions are hard to find and the exploration of solution space is slow.

Results degrade when recall is enabled for the simple simulator. It is likely that this is an example of the effects of divergence. The simulator has discovered an action set that is effective in simulation, but that does not translate well into the actual game. This indicates the importance of the simulator's combat model being a close match to the actual game's.

Results for Scenario D are both extremely strong and uniform across both the simple and complex simulators, both with and without recall enabled. This is probably a result of the scenario being relatively easy to solve, as indicated by the strong result also generated by the 'Attack Closest' scripted agent.

Over all scenarios, the strongest performance is shown by the complex simulator with recall enabled. This configuration of the S&R agent adapts strongly to all scenarios, even though its performance without recall enabled is relatively weak. The result is important, since it indicates that the build up of experience over many game cycles becomes greatly beneficial when solutions are hard to find, and simulation rates are slow. This is exactly the situation faced when attempting to apply accurate simulation models to complex commercial grade RTS AI problems.

Note that for all the search based configurations, results between zero and one are in some ways a measure of divergence, since the simulations return what they estimate as a winning solution or a loss. Solutions that win sometimes reflect differences between what the simulators calculate and what actually happens in Broodwar. This tends to impact weaker solutions to a greater extent, resulting in lower scores where search is less effective. Given this interpretation of each scenario score, it is an important result that the scores for the complex simulator with recall enabled are consistently high across all scenarios. This reflects relatively little divergence between what the complex simulator predicts and what happens in Broodwar, given a sufficient accumulation of simulations, and the capacity to retain the results.

Another important result is that the benefits of recall are delivered to the agent relatively quickly. There is a marked improvement for the complex simulator with recall enabled in the difficult scenarios even though the scenario is evolving in real time. This indicates that the advantage of receiving high quality solutions outweighs the disadvantage of them taking more than a game cycle to calculate.

In comparison with Churchill's results, the complex simulator with recall enabled dominates by a large margin in all but Scenario C, where it is only marginally weaker. Given the divergence interpretation of the evaluation scores, the results suggest that the complex simulator is a much closer approximation of the Broodwar combat mechanics, and that the predictions made by the complex simulator are much more accurate. The 'complex simulator with recall' approach is an approach worth pursuing.

6 Conclusions and Future Work

Overall the results of this preliminary study can be summed up as: high quality responses are worth remembering, when solutions are hard to find, the exploration rate of the solution space is low, and when the fidelity of the simulations is high.

The results strongly indicate that retention of results from search simulations is worthwhile, and that Search and Recall is a useful approach. This eliminates the need for a huge and uneven quality

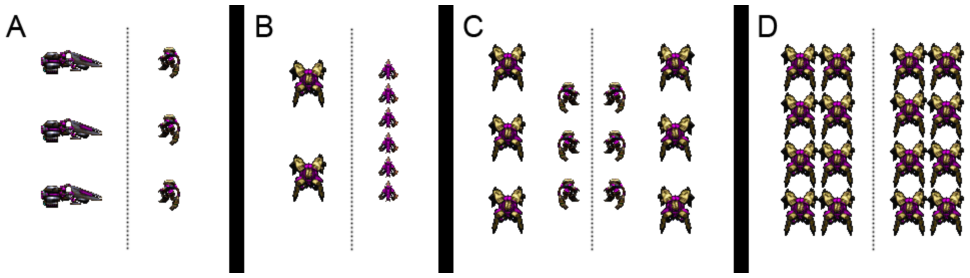

Figure 2. Experimental Setup

database of pre-played games on which to base CBR, and allows the situations a game AI can respond intelligently to grow over time. At the same time it guarantees fast decision making within the game loop.

An important implication of the proposed architecture is that because simulations are decoupled from the game loop, they become amenable to parallel, distributed, or offline processing. The exact actual game states sent to the SC, could instead be sent out over the network, or logged for later processing. Regardless of whether results arrive in time to advantage the S&R agent in the current game, the fact that the results are generated improves the response database over time, even when the game is not being played. Another implication is that simulation results from many separate instances of a game can be shared between games, allowing games to cooperate in improving the AI for all games.

A final implication is that simulations are not restricted to the CPU capacity of an ordinary gaming PC. Simulations could be conducted on server farms or supercomputers in the cloud, and the results used to update a global database available to all instances of a game.

Because the constraints on execution times and hence simulations complexity have been eased, future work could extend simulation models to scenarios of greater complexity such as working with terrain and larger unit encounters. It would also be interesting to explore the feasibility and utility of more detailed game state descriptors, and the associated much larger response databases required.

Once response databases become larger and more populated, game progression paths through state space and discovering general patterns of game progression could prove interesting. The sensitivity of results to the range of available behaviours also indicates that further work into more complex behaviour sets is also warranted.

S&R removes computational execution time restrictions on search but retains the ability of search based agents to adapt to new situations. The S&R agent model allows simulators used in searches to use much more complex models to deal with complex tactical situations. Simulators can include path finding, unit and terrain collision avoidance, and specialized behaviours. These complex simulators greatly improve the fidelity of the results produced, which reduces the divergence between predicted outcomes and those produced by the game. This makes the S&R method potentially useful in applying search techniques to commercial grade levels of combat scenario complexity.

REFERENCES

[1] David W. Aha, Matthew Molineaux, and Marc Ponsen, 'Learning to win: Case-based plan selection in a real-time strategy game', in *Case-Based Reasoning Research and Development*, volume 3620 of *Lecture Notes in Computer Science*, 5–20, Springer Berlin / Heidelberg, (2005).

[2] Klaus-Dieter Althoff, Ralph Bergmann, Mirjam Minor, Alexandre Hanft, Neha Sugandh, Santiago Ontan, and Ashwin Ram, 'Real-time plan adaptation for case-based planning in real-time strategy games', in *Advances in Case-Based Reasoning*, volume 5239 of *Lecture Notes in Computer Science*, 533–547, Springer Berlin / Heidelberg, (2008).

[3] Radha-Krishna Balla and Alan Fern, 'Uct for tactical assault planning in real-time strategy games', pp. 40–45. Morgan Kaufmann Publishers Inc., (2009).

[4] Michael Chung, Michael Buro, and Jonathan Schaeffer, 'Monte carlo planning in rts games', in *IEEE Symposium on Computational Intelligence And Games (Cig)*, (2005).

[5] David Churchill and Michael Buro, 'Incorporating search algorithms into rts game agents', in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, (2012).

[6] David Churchill, Abdallah Saffidine, and Michael Buro, 'Fast heuristic search for rts game combat scenarios', *AIIDE*, (2012).

[7] Leandro Soriano Marcolino and Hitoshi Matsubara, 'Multi-agent monte carlo go', pp. 21–28. International Foundation for Autonomous Agents and Multiagent Systems, (2011).

[8] Lorraine McGinty, David Wilson, Ben Weber, and Michael Mateas, 'Conceptual neighborhoods for retrieval in case-based reasoning', in *Case-Based Reasoning Research and Development*, volume 5650 of *Lecture Notes in Computer Science*, 343–357, Springer Berlin / Heidelberg, (2009).

[9] Manish Mehta and Ashwin Ram, 'Runtime behavior adaptation for real-time interactive games', *IEEE Transactions on Computational Intelligence and Ai in Games*, **1**(3), 187–199, (2009).

[10] Santiago Ontan, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Case-based planning and execution for real-time strategy games, 2007.

[11] Alberto Uriarte and Santiago Ontan, 'Kiting in rts games using influence maps'. AIIDE, (2012).

[12] Wang Zhe, Kien Quang Nguyen, Ruck Thawonmas, and Frank Rinaldo, 'Using monte-carlo planning for micro-management in starcraft', in *GAMEON Asia*, pp. 33–35, Japan, (2012).