Reinforcement Learning for Spatial Reasoning in Strategy Games

Michael Leece and **Arnav Jhala**

CCS, Dept. of Computer Science
UC Santa Cruz
Santa Cruz, CA, 95064, USA

{mleece, jhala}@soe.ucsc.edu

Keywords: Reinforcement learning, Q-learning, RTS games, spatial reasoning

Abstract

One of the major weaknesses of current real-time strategy (RTS) game agents is handling spatial reasoning at a high level. One challenge in developing spatial reasoning modules for RTS agents is to evaluate the ability of a given agent for this competency due to the inevitable confounding factors created by the complexity of these agents. We propose a simplified game that mimics spatial reasoning aspects of more complex games, while removing other complexities. Within this framework, we analyze the effectiveness of classical reinforcement learning for spatial management in order to build a detailed evaluative standard across a broad set of opponent strategies. We show that against a suite of opponents with fixed strategies, basic Q-learning is able to learn strategies to beat each. In addition, we demonstrate that performance against unseen strategies improves with prior training from other distinct strategies. We also test a modification of the basic algorithm to include multiple actors, to speed learning and increase scalability. Finally, we discuss the potential for knowledge transfer to more complex games with similar components.

Introduction

Spatial control is a critical aspect of real-time strategy games, and reasoning about spatial control is important for creating integrated agents that play real-time strategy (RTS) games. Recently, there have been several RTS domains in which integrated agents have been developed to play complete games. In addition, there have been competitions for agents playing RTS, the two prominent examples being Blizzard's StarCraft: Brood War and Wargus. However, these integrated agents have shown weaknesses in the area of reasoning about spatial control. Most work in this area has focused on real-time control of multiple heterogeneous units, strategies for maintaining technology trees, and opponent state estimation. This paper investigates performance of reinforcement learning algorithms against a variety of opponents in a game called Planet Wars, which was featured as one of the Google AI challenges. We look at whether playing against a variety of opponents improves performance against unseen opponents for our agent.

The spatial control problem in Planet Wars can also be represented as a sub-problem in a larger game like StarCraft: Brood War. Following from this work, we would like to explore whether results from learning against opponents in Planet Wars transfers to these more complex domains.

Related Work

Reinforcement learning has seen successful implementations for strategy games, but the general bent of the work has focused on aspects of these games other than spatial reasoning.

Jaidee and Muñoz-Avila have implemented a Q-learning agent for the real-time strategy game Wargus that has shown promise in defeating fixed-strategy opponents (Jaidee and Muñoz-Avila 2012). However, their state and action space is devoid of positional information, instead using resource and troop levels to define state, and using simpler position-ignorant actions that are variations on the basic attack and defend actions. Rörmark has shown a similar result, with an agent that primarily enforces resource management strategies (Rörmark 2009). Again, the action space is mostly to determine the order in which to construct buildings with limited resources, with simple attack and defend orders that ignore location on the map.

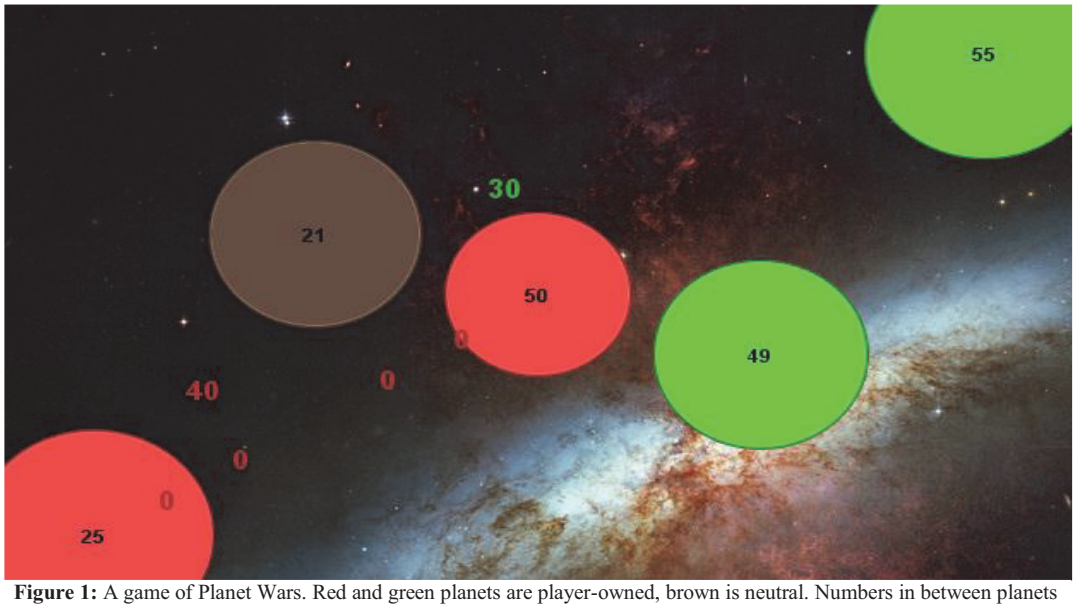

Figure 1: A game of Planet Wars. Red and green planets are player-owned, brown is neutral. Numbers in between planets represent fleets in transit.

In contrast, our environment eliminates these aspects, and focuses exclusively on troop locations.

Amato and Shani have implemented a number of reinforcement learning algorithms in Civilization IV, a turn-based strategy game, also with some success (Amato and Shani 2010). However, they were focused on choosing high-level strategy, and their agent chooses between a small number of hand-coded strategies based on current game state. We wish to avoid hand-coding the agent if possible.

A number of projects have investigated reinforcement learning for low-level unit control in games. Jaidee et al. have implemented a Wargus unit control agent for multiple units in a static environment, where the initial troops are the only actors throughout a test, although it ignores location to focus more on target selection (Jaidee, Muñoz-Avila and Aha 2011). Wender and Watson demonstrated an agent that was able to learn simple unit control tactics, and Ponsen et al. successfully navigated space while avoiding a pursuer (Wender and Watson 2012; Ponsen, Spronck and Tuyls 2006). However, both of these only demonstrated single agent directing a single unit. Molineaux et. al. have proposed an integration of reinforcement learning and case-based reasoning for continuous action spaces (Molineaux, Aha and Moore 2008). This last set is more closely related to our work, as they take unit location as input for their decision-making process. However, the environments in which they operate are neutral in terms of unit creation, which removes much of the spatial control aspect of the problem.

Fernández-Ares et al. have also done some work within the Planet Wars domain (Fernández-Ares et al. 2012). They use a genetic algorithm to learn parameters for a hand-coded agent, and have extended it to include pre-match map analysis to choose between a suite of learned parameters. While they have taken a different approach to the problem, there may be some interesting crossover between their work on map analysis and our system.

Q-Learning in Planet Wars

To test the ability of reinforcement learning in spatial reasoning and high-level resource management, we implemented a Q-learning agent to play Planet Wars, a two-person strategy game. Planet Wars is an adversarial, perfect information, deterministic, turn-based environment. The game involves taking and holding planets, which generate troops for further conquests. Planets are laid out in two-dimensional space with straight Euclidean distance as the travel time, and generate troops for their owner each turn according to their size. Neutral planets begin with a fixed army that must be defeated before it will begin producing for a player, and enemy planets can be conquered by bringing a greater army to bear on the planet than it has defending.

The mechanics of the game allow for some interesting gameplay tradeoffs. If an opponent is being hyper-aggressive, it is better to react to their moves: take planets where they have overextended their forces to defeat the neutral defense and left a weak garrison of his own. If they are playing reactively, it is best to play safe and only expand to defensible planets. If they are playing safely and expanding slowly, it is good to be hyper-aggressive and grab up as many planets as possible. Any single non-adaptive strategy is insufficient, and we wish to demonstrate that a reinforcement learning agent has the capability to learn and exploit these tradeoffs.

In addition, Planet Wars has the desirable quality of resembling an abstract version of more complex real-time strategy games, like Starcraft: Brood War and Wargus. If

individual unit management and technology choices are removed from these games, they become near duplicates of Planet Wars, revolving entirely around spatial control and resource management. In light of this, discoveries in the Planet Wars domain should translate fairly directly into more complex real-time environments.

QPlayer – A Single-Agent System

Our first agent is a basic implementation of the classical Q-learning algorithm, as described by Russell and Norvig in (Russell and Norvig 2003). In order to do this, we must define a game state, an action function to generate possible actions from any given state, and a reward model.

A Planet Wars state consists of three things: planet ownerships, planet troop levels, and fleets in flight. However, if we provide no abstraction, this leaves us with a state-space of

$$3^P * T^P * T^{F_l}$$

where

- P is the number of planets on the map
- T is the maximum number of troops possible on the map
- F_l is the number of possible locations for a fleet on the map

Even on a smaller map with just 5 planets, this number is approximately 10^{170}, clearly infeasible.

In response to this, we added three adjustable granularity levels, to control how fine-grained of a state the agent sees. The first is fleet distance. We divide fleets into buckets based on how far they are from their target, and merge each bucket into a single fleet for the state representation. The higher the granularity, the more buckets there are, and the less merging occurs. The second and third discretize planet and fleet troop sizes into chunks. For instance, with a planet granularity of 50, every planet's troop level is divided by 50 and truncated in the state representation. In this situation, the agent would view a planet with 60 troops as identical to one with 75.

The action function is simply the set of possibly troop dispatches from owned planets to all other planets, regardless of ownership. Troop amounts are also discretized in a similar manner as the state representation in order to allow for full exploration of the state-action space.

Finally, a reward model must be defined to let the agent evaluate how successful or unsuccessful its chosen actions have proven. For this, we simply took the difference between total troop numbers across the map for player 1 and player 2. This is the most sensible reward, as a game that goes to the turn limit will have the tie broken by troop levels, and within the game it is an accurate measure of which player is leading.

Planets as Actors

The previously described system is the most pure in terms of authored knowledge. Namely, if the granularities are brought down to one, it is a full representation of the game state, with no expert knowledge included. This is ideal for testing, as there is no question of how much of the agent's intelligence is derived from the authors, and how much it is truly learning.

However, as with any representation that involves the full game state, this has scalability issues. If trained on anything bigger than a small map, the size of the state-action table quickly becomes unmanageable. In light of this, we have also implemented a system in which each planet is an actor, and they share a single Q table. This approach is similar to the agent implemented by Jaidee (Jaidee and Muñoz-Avila 2012).

In addition to the in-game features to define a planet's state, we add a high-level "threat level" feature, which allows a planet to indicate to its neighbors whether assistance is required or not. This is to deal with the loss of information in moving from a global state to a local one, in that planet X does not include information on whether nearby planet Y is under attack when making its decisions, while it may need to know this in order to send reinforcements. A planet's state consists of its troop level, the troop levels and owners of its neighbors (nearest n planets), the size and distance of incoming fleets, and the threat levels of its neighbors. An action consists of dispatching some number of troops to a neighboring planet and setting your own threat level. The reward model is unchanged from the original system.

Experiments

The primary goal of our experiments was to determine the intelligence of QPlayer for this environment. A secondary objective was to examine scalability, and whether our modifications succeeded on full sized maps where the original system failed.

Opponents

We collected 8 fixed-strategy opponents of varying complexity for use in testing. A short description of each follows.

Simple Opponents

- Random: Chooses a random legal action each turn and executes it.
- Prospector: Expands slowly, only attacks the opponent after all neutrals on the map have been taken.
- Rage: Only attacks opponent's planets, never neutrals. Picks one planet and sends troops until it falls, then picks a new one.
- Bully: Focuses on attacking opponent planets, but sometimes takes neutrals.
- Dual: Attempts to get ahead of opponent in planets—more specifically in unit production, then only attacks opponent.

Medium Opponents

- WithDefense: Similar in high-level strategy to Prospector, it expands as aggressively as possible.

	Random	GeneralMedOpt	Bully	Dual	Rage	GeneralHiOpt	WithDefense	Prospector
Random	-	0.0%	0.0%	0.0%	10.1%	0.0%	0.4%	0.0%
GeneralMedOpt	100.0%	-	100.0%	58.5%	84.9%	7.5%	39.6%	100.0%
Bully	100.0%	0.0%	-	0.0%	50.9%	0.0%	0.0%	18.9%
Dual	100.0%	32.1%	100.0%	-	58.5%	3.8%	9.4%	98.1%
Rage	89.7%	17.0%	54.7%	37.7%	-	32.1%	45.3%	45.3%
GeneralHiOpt	100.0%	86.8%	100.0%	100.0%	69.8%	-	100.0%	100.0%
WithDefense	100.0%	66.0%	100.0%	96.2%	54.7%	0.0%	-	98.1%
Prospector	100.0%	3.8%	94.3%	5.7%	60.4%	0.0%	1.9%	-

Table 1: Win rates for fixed-strategy opponents (row wins against column represented)

The key difference is that it includes reinforcement and defense of its own planets as a highest priority, while Prospector ignores attacks.

- GeneralMedOpt: Weighs expanding aggressively and attacking opponent, also defends own planets. However, the defense code for both of these agents is fairly simplistic, only looking to the immediate attack, and not subsequent ones.

Complex Opponent

- GeneralHiOpt: Balances different high-level strategies, similar to GeneralMedOpt. The difference is that planet strengths are projected into the future to allow for more optimized defending and attacking. In particular, this agent considers all fleets and their distances affecting a particular planet when deciding whether to reinforce or attack, and from where those troops should come.

While we did not perform a thorough test comparing this suite to human players, our experience places human play at or slightly below the level of the medium opponents. To determine a rough ranking of these opponents' difficulty, we played them against each other on 51 different maps. Each map was played 10 times, although since every agent but the Random agent were deterministic, this repetition proved redundant (but helpful for validating the system). The results of these tests are shown in Table 1.

QPlayer – Learning Times

Initial testing showed that QPlayer was infeasible on large maps (>10 planets). Therefore, our first rigorous test used a smaller map with 5 planets, symmetrically distributed. The goal for this test set was to show that QPlayer could learn to defeat fixed opponents, and also to provide a baseline learning level for comparison with learning after training. Testing involved setting a fresh QPlayer against one of the fixed strategy opponents, and running either until the QPlayer converged to repeating a winning strategy, which we define as playing the exact same game 300 games in a row, or we reached 10,000 games, whichever came first. We then took the last loss of QPlayer as the learning time for that test. Note that our definition of convergence does not presume that the agent has a complete policy, but rather that it has exhausted the exploration side of the classic explore-exploit tradeoff and is only exploiting learned states. As a result, convergence for QPlayer occurs once it has a strategy that is relatively robust to small variations in the game.

Using the granularity variables in QPlayer, we tested 18 different configurations, each of which had a different fineness of state-action options.

In addition, we performed a similar suite of tests using the modified multi-agent system with planets as actors on 3 full-sized maps (20-30 planets). The intention of these tests was to determine whether the modifications would be able to handle the increased complexity and state-space size.

QPlayer – Prior Training

Our second suite of tests aimed to determine whether QPlayer was actually learning anything of use during these state-space explorations, or whether knowledge learned against one opponent was completely invalid against another. To do this, we trained QPlayer using leave-one-out training against seven of the opponents for 10,000 total games, with opponents being rotated each game. We then set it against the eighth opponent and measured the learning time again. We used the same map, and tested each of the same 18 granularity configurations.

For the results of both of these tests, we have removed the Random opponent from the tables. While QPlayer dominates Random after learning (>90% win ratio), it would take an extraordinary number of games to reach convergence, since the Random player will explore states with equal probability. As it is, the Random player is only able to win when it randomly sends the game into a state the agent has not seen before, from which it will respond randomly as well.

Results and Evaluation

Tables 2 and 3 show the results of the QPlayer tests. There are a number of interesting conclusions, which we will highlight here.

	Ps100Fs50Fd0	Ps100Fs50Fd1	Ps100Fs50Fd3	Ps100Fs20Fd0	Ps100Fs20Fd1	Ps100Fs20Fd3	Ps100Fs10Fd0	Ps100Fs10Fd1	Ps100Fs10Fd3	Ps50Fs50Fd0	Ps50Fs50Fd1	Ps50Fs50Fd3	Ps50Fs20Fd0	Ps50Fs20Fd1	Ps50Fs20Fd3	Ps50Fs10Fd0	Ps50Fs10Fd1	Ps50Fs10Fd3
WithDefense	417	299	718	189	201	5235	413	531	2541	120	49	179	408	1062	2061	310	1239	3379
GeneralMedOpt	1618	4521	4942	1064	2412	5320	1463	4563	5752	2340	1186	3399	2393	2423	6168	412	1579	5067
Bully	89	58	228	136	1294	1909	381	1658	1518	207	1271	80	259	2867	3047	1537	5873	2395
Dual	100	102	83	327	743	1872	671	4669	2448	636	2808	2820	408	1683	348	1664	3354	2896
GeneralHiOpt	2777	4936	4235	6050	6788	8948	3746	9003	8241	1909	3129	6137	5465	7754	10001	3650	8617	10001
Prospector	116	95	513	416	250	324	220	773	1398	128	63	316	331	1541	369	1531	1566	2532
Rage	10001	176	269	8906	326	1043	10001	6850	9442	10000	2002	3190	10001	3629	9938	9277	10001	10001

Table 2: Average number of games before convergence for QPlayer for various granularity configurations
(Ps = Planet size granularity, Fs = Fleet size, Fd = Fleet distance divisions)

State-Space Granularity

The first observation that we make is regarding the difference that the granularity makes in the success of the agent. If we consider Table 2, we can clearly see that, as a general rule, the more finely grained the state, the longer it takes to reach convergence. There are two factors at play here. First, the more coarse the states and actions, the more difficult it is for the agent to effectively play the game. It may use more troops than necessary for some task, leaving it short in other areas, purely because it only can send troops in large increments. On the other hand, the state-space of a coarser configuration is much smaller, and so the agent can explore it more fully, getting more accurate value assignments for each state-action pair.

Figures 2 and 3 show the state-space exploration for a coarse- and fine-grained test, respectively. It is clear that the coarser agent was able to explore a majority of the state, while the finer agent only a small fraction. In combination with the observation that the more coarse-grained versions tend to converge to a winning strategy faster, these results show that, in this domain, state-space coverage is more crucial than representation precision. We postulate that this is due to the adversarial nature of the environment, where accurate values for actions can be difficult to determine, as they depend heavily on opponents' responses. However, to verify this hypothesis we would need to implement this approach in other domains, including non-adversarial, and compare the relationship between state-space exploration and representation precision in each of those.

In short, what these figures show is that there may be more winning strategies available in the fine-grained representation, as we can find one with a much smaller percentage of the space explored, but that it is faster for QPlayer to find a winning strategy among the fewer options that are presented by a coarser state.

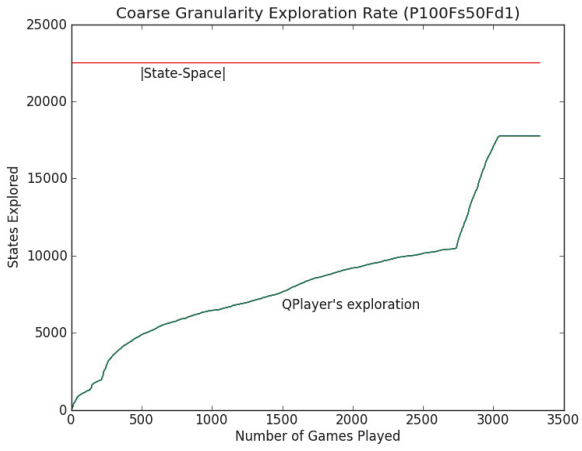

Figure 2: State-space exploration for QPlayer in a coarse configuration

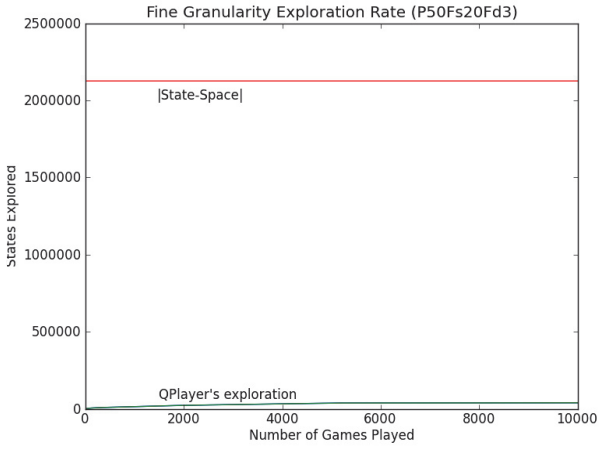

Figure 3: State-space exploration for QPlayer in a fine configuration

Learning Capabilities

While this granularity conclusion is nontrivial, it is still secondary to the main question of our work, whether Q-learning is able to learn strategies in a spatial management game. To answer this question, we consider Table 2, where entries of 10,000 indicate a failure to learn a winning strategy, and anything less indicates the mean number of games required to learn a winning strategy. We can see clearly that, while the convergence time depends on the granularity and the optimization level of the opponent, QPlayer does in fact converge to a winning strategy in the strong majority of cases. In particular, with correct state granularities, it will converge to defeat every fixed-strategy opponent in our suite.

	Ps100Fs50Fd0	Ps100Fs50Fd1	Ps100Fs50Fd3	Ps100Fs20Fd0	Ps100Fs20Fd1	Ps100Fs20Fd3	Ps100Fs10Fd0	Ps100Fs10Fd1	Ps100Fs10Fd3	Ps50Fs50Fd0	Ps50Fs50Fd1	Ps50Fs50Fd3	Ps50Fs20Fd0	Ps50Fs20Fd1	Ps50Fs20Fd3	Ps50Fs10Fd0	Ps50Fs10Fd1	Ps50Fs10Fd3
WithDefense	720	1567	75	37	2307	262	128	775	151	2163	**1**	210	3335	3353	149	1246	75	**233**
GeneralMedOpt	1200	158	3396	766	7122	10000	2456	8240	5361	5279	569	7286	3487	3415	3720	41	3970	189
Bully	0	**0**	83	25	111	43	19	0	95	0	0	0	11	0	0	942	3606	0
Dual	2508	32	83	53	15	45	**0**	66	5000	90	277	1728	2357	101	199	1822	3130	6698
GeneralHiOpt	2607	992	205	5581	6719	6717	6678	4435	5156	3460	7434	3404	205	5404	**3398**	1960	10000	10000
Prospector	183	0	5	0	33	0	313	209	5067	91	**0**	0	110	1255	0	3981	112	908
Rage	10000	176	**164**	3414	143	390	10000	6667	10000	10000	453	7358	10000	1914	**9292**	5836	**6667**	10000

Table 3: Average number of games to convergence for QPlayer following 10,000 games of leave-one-out training. Green cells indicate an improvement from Table 2, bolded values are statistically significant (two-tailed t test, $p < 0.05$)

In addition, by comparing Tables 2 and 3, we can see that when QPlayer has the opportunity to train against the remaining opponents prior to testing, its learning time improves. This tells us that QPlayer is truly learning strategies for spatial control and resource management, rather than simply randomly exploring until it hits on a winning strategy and then exploiting it. This is our key result, as it indicates that Q-learning is a legitimate approach for reasoning about spatial control in fully integrated agents.

Scaling Up

Unfortunately, our attempts to scale Q-learning up to larger maps were unsuccessful. While the agents were able to compete on the large maps without running out of memory, and even defeat the opponent occasionally, they never were able to win consistently.

There are two potential reasons for this. The first is that it has simply not had enough training games, and is still attempting to learn accurate values for each state-action pair. Continued testing, with greater than 10,000 games, can determine if this is the case. However, while we are performing these tests for edification purposes, there is a point at which the amount of time required to train an agent becomes infeasible. The second potential flaw is that, in removing state information from each planet's individual view, we eliminated information that is necessary to correctly assess the value of a state. Fixing this issue would require more fine-tuning of our expert knowledge, which is of less interest to us than the agent learning itself.

Future Work

There are two major directions for future work on this project. The first is continuing to explore reinforcement learning within Planet Wars itself. While our implementation was successful on smaller maps, we still struggle to maintain the learning ability when scaling up to full sized maps. We would like to implement a hierarchical reinforcement learning system in an attempt to solve this problem. This system could tie in to the work done by Fernández-Ares et al. involving map characterization (Fernández-Ares et al. 2012), such that the hierarchical structure could be map-dependent. It is also important to gain a more complete understanding of the role that state granularity can play. In light of this, we are continuing to run granularity tests on a wider range of configurations.

Secondly, now that we have shown that reinforcement learning is a viable tool for spatial management strategies, we plan to utilize this in more complex games. Current real-time strategy game agents are often compositions of multiple managers, each handling a different aspect of the game. In addition, while complications in creating a reduction from these more complex games do exist (such as differing unit movement speed or terrain effects), we believe creating a reduction from an instance of these more complex games to an instance of Planet Wars should be possible. A reinforcement learning agent trained in Planet Wars on these reductions with an appropriate knowledge transfer algorithm, should be able to provide the base for a spatial control manager in a higher-level agent.

Conclusion

In this paper we demonstrated the capability of reinforcement learning in handling spatial reasoning and resource management in strategy games. We showed this by implementing a Q-learning agent, QPlayer, in the game Planet Wars, and testing it against a suite of fixed-strategy opponents. Going forward, these results provide an additional method for handling an area of weakness for current strategy agents.

Acknowledgments

This work is supported by the National Science Foundation under Grant Number IIS-1018954. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Amato, C., & Shani, G. (2010, May). High-level reinforcement learning in strategy games. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1 (pp. 75-82). International Foundation for Autonomous Agents and Multiagent Systems.

Fernández-Ares, A. J., et al. "Adaptive bots for real-time strategy games via map characterization." Computational Intelligence and Games (CIG). IEEE, 2012.

Jaidee, U., & Muñoz-Avila, H. (2012, July). CLASSQ-L: A Q-Learning Algorithm for Adversarial Real-Time Strategy Games. In Eighth Artificial Intelligence and Interactive Digital Entertainment Conference.

Jaidee, U., Muñoz-Avila, H., & Aha, D. W. (2011). Case-Based Learning in Goal-Driven Autonomy Agents for Real-Time Strategy Combat Tasks. In Proceedings of the ICCBR Workshop on Computer Games (pp. 43-52).

Molineaux, M., Aha, D. W., & Moore, P. (2008). Learning continuous action models in a real-time strategy environment. In Proceedings of the Twenty-First Annual Conference of the Florida Artificial Intelligence Research Society (pp. 257-262).

Ponsen, M., Spronck, P., & Tuyls, K. (2006). Hierarchical reinforcement learning in computer games. ALAMAS'06 Adaptive Learning Agents and Multi-Agent Systems, 49-60.

Rörmark, Richard. 2009. Thanatos – A learning RTS Game AI. Masters Thesis. Department of Informatics, University of Oslo. Oslo, Norway.

Russell, S., & Norvig, P. (2003). Artificial intelligence: A modern approach (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Wender, S., & Watson, I. Applying Reinforcement Learning to Small Scale Combat in the Real-Time Strategy Game StarCraft: Broodwar.