Evolving the Behavior of Autonomous Agents in Strategic Combat Scenarios via SARSA Reinforcement Learning

Clauirton A Siebra Gutenberg P Botelho Neto

Federal University of Paraiba, Dept. of Informatics, Brazil

Figure 1: Snapshots of experiments, using the BWAPI for Starcraft Brood War, where behaviors of autonomous agents are evolved via Reinforcement Learning techniques.

Abstract

Computational agents of commercial Real Time Strategic (RTS) games mostly have their behaviors designed via simple ad hoc and static techniques, which require manual definition of actions. Thus, such agents are not able to adapt themselves to diverse situations and their behavior becomes predictable along the game, enabling human players to eventually discover the strategies used by them. This work proposes a modeling approach for the use of SARSA reinforcement learning technique applied to combat situations in RTS games. This technique enables that computational agents evolve their combat behavior according to actions of opponents. The performance of this technique was evaluated using a Starcraft based simulator. The experiments showed that agents were able to improve their behavior, developing knowledge to decide about the best actions during different game states and using this knowledge in an efficient way to obtain better results in later battles.

Keywords: combat, reinforcement learning, SARSA, strategic games

Authors' contact:
{clauirton,gutenberg}@di.ufpb.br

1. Introduction

Several games, especially Real Time Strategic (RTS) games, use to apply a simple set of ad hoc rules to define the behavior of computational agents. Thus, such agents do not usually present more complex and unpredictable behaviors, mainly when we consider such behaviors along the time. Furthermore, a common strategy to increase the game challenge is to enable that computational agents have advantages that are not available to human players [Dimitriadis 2009; Sailer et al. 2007; Sandberg 2011; Walther 2006]. Common examples of this strategy are: use of extra resources such as more combat units and structures, bonus that turn the actions of computational agents more efficient, perception of additional information that cannot be perceived by human players such as to see throughout fog of war and so on [Micic et al. 2011; Pérez 2011; Rormark 2009; Walther 2006].

In this context, several recent researches in IA applied to RTS games are considering machine learning techniques to increase the intelligence and adaptability of computational agents considering that they use the same information than human players. That means, without additional advantages. Thus, computational agents should be able to adapt themselves to opponents' actions and test new strategies along the game [Aha et al. 2005; Andersen et al. 2009; Balla and Fern 2009; Dimitriadis 2009; Jaidde et al. 2011; Kok 2008; Ponsen 2004; Ponsen and Spronck 2004; Rormark 2009].

Even considering the importance of combat in modern RTS games, we see a lack in works that explore the opportunities of this area [Neto and Siebra 2012]. The majority of the IA researches for RTS games are focused on more generic techniques or problems that do not consider conflicting actions, such as high level planning and resources gathering.

This work analyses the efficiency of applying reinforcement leaning approaches, in special the SARSA(State-Action-Reward-State-Action) technique, in strategic combat situations. In order, combat is an interesting field for IA research once this situation presents several complex features such as the need of interaction and cooperation between joint units and the presence of opponent units that have conflicting goals.

The remainder of this work is organized as follows: Section 2 discusses important related works that have similar aims than our proposal. Section 3 details our approach, presenting the SARSA technique and how we have modelled the learning module in accordance with the RTS combat scenario. Section 4 describes the experimental scenarios, which have used a Starcraft based simulator, and the obtained results. Finally, section 5 concludes this work with the main remarks and research directions.

2. Related Work

According to [Ponsen 2004; Ponsen and Spronnck 2004] about 80% of commercial RTS games use deterministic techniques, such as scripts and finite state machines, which require a manual definition of behavior aspects of computational agents. In this scenario, game developers should think about all situations and possible game states where agents could act. Based on this limitation, Ponsen and Spronnck [2004] proposed the use of an offline evolutionary learning algorithm to evolve the use of dynamic scripts. In such approach, a new script that control the behavior of agents is created based on a set of pre-defined rules. The probability of choosing rules is based on weight values and such weights are modified according to the results of their use during the game. This technique of dynamic scripts was used in several combat situations in the Wargus environment and it obtained good results. After that, some of the resultant behaviors were included in the set of rules, once its application was offline.

A different approach is presented in Ponsen et al. [2005]. The authors use the case-based principles to save past situations of previous games as cases. These cases contain variables that represent the situation of each game and such cases are evaluated according to the scores of players in different moments of the game. This learning strategy was also evaluated in the Wargus environment and the results show that the agents that used this strategy defeated their opponents in about 80% of the games, even when they play against distinct opponents. However, the results of this work are not clear once they do not show the exact efficiency of agents when several games are sequentially played against different opponents [Kok 2008].

The work of Kok [2008] used a BDI architecture to create intelligent agents. The idea was to use a set of pre-defined rules, or static scripts, and provide agents with the ability of selecting the best rules to be applied in each moment. This selection ability was implemented via different techniques, such as the Monte Carlo algorithm. The experiments were executed in the Bos Wars RTS game and this approach had good results when used against different agents controlled by static scripts. However the same good results were not repeated when opponents were controlled by dynamic scripts.

Another work that proposes a learning framework for RTS games was presented in [Andersen et al. 2009]. This framework was specified in multiple layers, where the first layer accounts for identifying the opponent profiler, the second layer accounts for high level strategies, where rewards are received based on the profile previously identified, and several low level actions layers. As learning technique, this framework used the SARSA and Q-Learning algorithms to control agents against static agents. In the majority of the experiments, agents implemented according to this framework had better performances than their static opponents and Q-Learning obtained better results than SARSA in such situations.

Laursen and Nielsen [2005] investigated the use of game trees during RTS combat situations using the Wargus environment. The proposal aimed at calculating the possible future alternatives of the game, considering the current state, future actions of opponents and future actions of the own agent. Thus, this agent could decide for the best theoretic action to perform. This approach is very common in turn-based games, such as Chess. To be used in real time games, the tree formation was modified via the insertion of timestamps that store the number of the game cycle associated with each state of the tree. Then, the children of a specific node always have a timestamp higher than its parent and this timestamp indirectly represents the time required to complete the action related to this node. In this way, if the current state has a timestamp of 100 and its action spends 15 time units, then its child will have a timestamp of 115. Several experiments were carried out in the Wargus environment in different combat situations and also using different sequence of rules. The results showed the complexity in creating good strategies to combat situations using trees. The authors did not find sequence of rules or strategies that could be optimal in all situations, once the features of each experiment (type of opponents, environment, etc.) had a considerable influence on the results. Despite the problems, the game tree technique was able to defeat the embedded IA of Wargus.

Dimitriadis (2009) explored the use of reinforcement learning in RTS games using the Glest game as environment. The state representation in this work was carried out via four discrete variables: number of units killed by the player, number of units produced by the player, resources recovered by the player and a game cycle counter. A reward function was based on the score of each player. Two techniques were used during the experiments: SARSA (online) and LSPI (offline). These techniques were used in two Glest scenarios: Duel (medium difficulty level) and Tough Duel (high difficulty level). Both SARSA and LPSI obtained good results in simple scenarios, but they were not able to discover the winner strategy in

more complex scenarios. According to the authors, the main reasons for this negative result were the "subversive" advantages given to Glest agents by the game. Authors also commented that the simple representation of states may be the reason for the lack in convergence to a stable policy.

Balla and Fern [2009] proposed an intelligent planning based approach to control the computational agents in a RTS game during combat situations. The proposal is based on concepts of unit groups. For each decision moment, the game considers the alive units and an online planning is carried out to define abstract actions to groups. This planning is executed until the next decision moment. Two abstract actions were defined. Join(G), where G represents a set of groups, is an action that forces the groups G to move to the groups centroid location. In this way, the idea of this function is to form a bigger and compact group. The Attack(f,e) function, where *f* is an joint group and *e* is an opponent group, indicates that *f* must move against *e* to attack it. The authors do not define individual behavior to each group unit, so that these behaviors are controlled by the IA of the game. This strategy was evaluated in the Wargus environment, considering 12 different scenarios. The results were positive and the planning techniques were able to find winner strategies in all scenarios.

The work of Sandberg [2011] proposes the use of Multi-Agent Potential Fields (MAPF) to control the formation of units during combats in RTS games. Genetic algorithms were used to select fields' parameters that ensure the best quality of the results. Using this strategy, agents are able to observe the current state and move themselves to a location that returns the higher positive value for them. The proposal was validated in the BWAPI-Starcraft environment. The author used this technique to control the *Terran* group, using eight different potential fields defined via equations whose parameters are defined via genetic algorithms. The experiments were executed in diverse combat scenarios and the agents were able to defeat the embedded IA of Starcraft in both situation: when the opposite team was composed of same units and when it was composed of different units.

Jaidee, Muñoz-Avila and Aha [2011] proposed the use of reinforcement learning via the use of case-based learning to Goal-Driv4en Autonomy (GDA) agents. The idea of a GDA agent is to execute a specific action in the environment and calculate the difference between the real result and the expected result. This information is stored as cases and the learning algorithm in GDA must be able to learn the best goals during the game via the analysis of these cases. Authors defined five possible goals to be performed by agents in two different maps against two other agents that were respectively implemented using the Q-Learning and the traditional GDA approaches. Positive results were obtained against the Q-Learning agent. However, this performance was not repeated against the GDA agent. However, the authors say that the performance of their proposal becomes better along the time when more cycles of the game are carried out.

Micic et al. [2011] analyzed the application of SARSA reinforcement learning in combat situations in the Starcraft: Brood War game. Two situations were considered: Dragoon vs. Zealot and 4 Draggons vs. 4 Zealots. The idea was to verify if the use of reinforcement learning is able to improve the behavior of Dragoon units so that they defeat their opponents (Zealot units). Compared to zealots, dragoon units fire slower and has an attack optimized against only large-sized units. Thus, we can consider the Zealot units as the favorite in such battle. However, the victory of Dragoon units is possible if they use the *kiting* strategy. This means, they must maintain a long distance of their opponents so that they can attack while avoid counter-attacks. In this case, the aim of the learning technique is to learn this strategy. In the single agent experiment, the rate of victory was 60%. However, in the multiagent experiment, the rate of victory was only 32%. Table 1 stresses some of the main features of our analysis.

Table 1. Summary of the comparative analysis among AI approaches for computational agents control in RTS games.

Ref	Learning	Validation in combat	Environment	Technique
1	Yes, offline	Yes	Wargus	Evolutionary algorithms
2	Yes, offline	No	Wargus	Case based reasoning
3	Yes, offline	No	Bos War	Monte Carlo applied to dynamic scripts
4	Yes, online	No	Tank General	Q-Learning and SARSA
5	Yes, along game state evaluation	Yes	Wagus	Game tree with timestamps
6	Yes, offline and online	No	Glest	LPSI and SARSA
7	No	Yes	Wagus	Planning
8	Yes, but to only adjust field values	Yes	BWAPI	Potential fields
9	Yes, online	Yes	Wagus	Case based learning to Goal-Driven autonomy
10	Yes, online	Yes	BWAPI	SARSA with variations for each scenario

This table shows that several AI techniques have been applied to provide learning abilities to computational agents of RTS games. However, few of them were validated in combat situations. Another important point is related to the use of SARSA approach. We have found that three works have used

this learning technique. In the first work [Andersen et al. 2009], SARSA was only used in an initial test and, after that, the authors started to use Q-Learning based on the better results of this latter approach. In the second work, the problem modelling was very abstract so that SARSA did not present good results in several game situations [Dimitriadis 2009]. The unique work that evaluated SARSA in specific combat situations was [Micic et al. 2011]. However, its authors decided to modify the SARSA modelling to each test scenario, so that the results cannot be generalized to be used in a commercial game. Furthermore, only two scenarios were used during the tests and the technique only presented good results in the simpler scenario. Based on the analysis of these works, we argue that SARSA was not extensively and correctly explored, so that we focused our approach on this technique.

3. SARSA Based Combat Modelling

Our proposal has five fundamental aspects: the environment used during the implementation and validation, the state representation, the possible actions of agents and the leaning technique. Each of these aspects is detailed along this section.

3.1 Environment

The environment used in this work was the BWAPI, which presents several advantages, mainly related to the support to definition of IA modules that directly runs inside the *Starcraft: Brood War*, one of the most popular RTS game. The validation in this environment is also interesting because we can test our ideas against the IA of a real commercial game.

3.2 State Representation

The state representation is fundamental to the success of any AI modelling, once agents use this representation as a form to understand its current game situation and, consequently, select the appropriate actions at each moment. However, this representation cannot contain so many details because the search space tends to exponentially increase, creating problems to the learning processes.

The combat situation does not have a pattern to represent states in RTS games. Thus, we have analyzed the domain and identified possible situations for combat units in the environment. The idea was to select a set of variables that could provide enough knowledge to agents, at the same time that the use of these variables in the learning process does not negatively affect the performance of the agent, considering the real time feature of the system.

The representation contains six variables and each of them has three possible values. Thus, the total number of states is 729 (3^6). The variables and their possible values are:

- Life: represents the current life points of the agent. Possible values are low (less than 1/3), medium (between 1/3 and 2/3) and high (more than 2/3);
- Enemies that can be attacked: represents the amount of enemies that can be attacked by the agent from its current position. This means, the attack does not require additional moving actions. Possible values are 0, 1 and 2 or more;
- Enemies that can attack: represents the amount of close enemies that can attack the agent in its current position. Possible values are 0, 1 and 2 or more;
- Enemies attacking: represents the amount of enemies that are currently attacking the agent. Possible values are 0, 1 and 2 or more;
- Close joint units: represents the amount of joint units close to the agent. The close joint units are those that are in a specific neighborhood area of the agent. This area is defined by the game and depends on the maximum attack distance of each unit weapon. Possible values are 0, 1 and 2 or more;
- Medium life of close enemies: represents the medium life of enemies that are close to the agent. Possible values are low (less than 1/3), medium (between 1/3 and 2/3) and high (more than 2/3).

We argue that this representation is appropriate once it enables that an agent has a general sense of the battle. For example, the agent is able to know if it is trapped around enemies, if enemies are in good or bad situations, if there are joint units that could assist it, if it is in risk to die soon and so on.

3.3 Agents Actions

The amount of possible actions that can be carried out for an agent also affects its search space. Thus, similarly to the state representation, we need to abstract these actions to decrease the amount of possible actions.

In RTS, for example, the amount of possible actions that could be carried for an agent was estimated in 1500 [Aha et al. 2005]. This estimation considers actions related to battle, resource gathering and construction of buildings. However, even considering just battle actions, this number is still intractable. Just to set a comparison, a Chess player has about 30 possible actions at each turn.

The approach used in our work was to consider actions in a high level of abstraction, which define general behaviors. Thus, low level actions are not

considered in the learning process. The next actions were defined to agents, together with the low level actions that compose them:

- Attack: the agent must engage in an offensive attitude, looking for the maximum damage to opponents. To that end, the following priorities were defined in order of importance:
  - Attack closer enemies with less life;
  - Attack enemies that are attacking the agent;
  - Attack enemies that are already being attacked by joint units;
  - Attack closer enemies;
  - If no enemies are closer to the current position, the agent tries to move to some attack position.
- Support: the gent is more worried in assisting its joint units. To engage in this attitude, the next priorities are defined:
  - Search for closer joint units with less life. If the agent has some assistive action (e.g. cure), this action is executed. Otherwise, the agent attack the closer enemy that has the joint unit as target;
  - Try to move in formation. This means, move to the centroid of the closer joint units to increase the cohesion;
  - If no joint units are close, then try to move to a position that allows an attack action;
- Retreat: the agent engages in a retreating attitude to preserve its survival. To that end, it leaves the offensive attitude and tries to move to a secure local. This means, the agent looks for a position that it can reach inside a specific time interval and where there are fewer enemies that could attack it.

These actions represent just a small percentage of all possible actions that an agent could execute during the game. However, when combined, they present a good variety of behaviors to each agent.

3.4 SARSA Reinforcement Learning

The learning technique used in our implementation was SARSA, a variation of the Q-Learning reinforcement learning strategy. As Q-Learning, SARSA ensures the convergence to an optimal policy. This means, it always ensures the choice for the best action in each state. SARSA uses state-action pairs $Q(s,a)$ to represent the expected result of the action a at the state s. This set of pairs is updated according to the next function:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

In this function, α ($0 \leq \alpha \leq 1$) represents the learning rate of the system, γ ($0 \leq \gamma \leq 1$) represents the discount factor that means the importance given to future actions, r represents the received reward and s' represents the new state of the environment when the action a' is executed in the next iteration. Thus, the value Q(s',a'), related to the next action, is directly used to update the Q table. The SARSA algorithm is specified as:

1. Start $Q(s,a) = 0$ to all state actions pairs (s,a);
2. At the state s_0, select a possible action a_0;
3. To each t = 0, 1, 2…;
   3.1. Execute action a_t;
   3.2. Receive reward r;
   3.3. At the state s_{t+1}, select a possible action a_{t+1};
   3.4. Update $Q(s_t, a_t)$ using the update function;

Note that the environment of a RTS game is multiagent. However, this proposal does not consider the use of multiagent reinforcement learning (MARL). The reason is the exponential increase of the search space of MARL in practical applications, according to the number of agents. In addition, the current MARL approaches do not ensure the convergence to optimal policies for joint behaviors. On the other hand, our approach considers a light notion of cooperation between agents. In order, this notion of cooperation is fundamental among agents of a team, once if agents act completely independent, there is a high chance that they execute conflicting actions, decreasing the efficiency of the team.

A strategy to consider the teamwork aspect is via the reward function. Our proposal implements this strategy via the specification of the next function:

$$(\sum_{i \in I} h_{i,t} - \sum_{i \in I} h_{i,t+1}) + (100 * n_{\Delta t}) - [(v_t - v_{t+1}) + (100 * m_{\Delta t})]$$

In this function, I represents the set of close enemies at the moment t, $h_{i,t+1}$ represents the life of the unit i at the moment t+1, $h_{i,t}$ represents the life of the unit i at the moment t, $n_{\Delta t}$ represents the amount of died enemies between the moments t and t+1, v_{t+1} represents the life of the agent at the moment t+1, v_t represents the life of that the agent has at moment t, $m_{\Delta t}$ is 1 if the agent died between the times t and t+1, otherwise $m_{\Delta t}$ is 0. This function considers all the life points that are lost for all close enemies of the agent, counting the damage caused by joint units and given bonus for each died enemy. Thus, if the agent decides for an action that enables that its joint units can be more effective, this action will be rewarded. The penalty occurs if the agent dies after deciding for an action.

Our work uses the same Q table to all units of the same type (e.g. *Terran Marines* or *Protoss Zealots*). This is useful because enables that each agent uses the knowledge acquired by other similar agents, accelerating the learning process. Another advantage is

that, as each player normally uses in each attack several units of different types, the need of memory will be decreased once we do not need to individually store a different Q table to each unit during the game.

BWAPI enables the control of the game frame to frame. The use of just a frame is not enough to evaluate the result of an action. Thus, our work also proposes the relation of a time unit *t* to a pre-defined amount of frames. This amount must be bigger enough to enable the evaluation of an agent action, but it cannot be so longer because the agent will maintain the execution of the same action and will not be able to react to new perceptions of the game. The interval of the time used in our work was 20 frames. Thus, if the moment *t* occurs at the frame 80, the moment *t*+1 will occur at the frame 100.

The choice of an action by the agent is given via *ε-greedy* [Watkins 1989], with value equals to 0,1. Our SARSA version defines the discount factor γ as 0,9 and the learning rate α as 0,1. Using these values, we tried a balance between the need of exploring the environment and taking advantage of the knowledge acquired from previous executions.

4. Experiments

The experiments were carried out using the Starcraf: Brood War game, via battles of native AI agents against agents using the SARSA based proposal. These battles used different maps, which are specified just to battle situations. This means, the maps did not contain buildings and resources to be collected by agents. The scenarios also presented distinct initial configurations regarding formation of both teams.

The main aim of the evaluations was to show that this proposal is able to evolve the computational agents behavior, which initially do not have specific knowledge of the game, during combat situations so that along the time they can be able to defeat their opponents in distinct situations. We also tried to use results of similar evaluations presented in the related work, so that we could have a basis for comparison. In all tests, 5 set of 100 battles were carried out. This means, the agent learning occurred along 100 battles and, at the end, the knowledge tables and the whole process were restarted.

4.1 Evaluation 1 - Protoss Dragoon x Protoss Zealot

This test is similar to the presented in Micic et al. [2011], where its agent controls a Dragoon unit of the race Protoss, while the native AI controls the Zealot unit. The Zealot unit is a stronger unit once it has more attack capacity than the Dragoon. However, Zealot can only attack close opponents.

This test requires that the agent learns to execute a *kitting* strategy once it is impossible that the Dragoon wins a "face-to-face" battle. Then, the Dragoon needs to learn to keep distance and use long distance shots, avoiding that the Zealot decreases this distance.

Table 1 presents the percentage of wins obtained by the agent after 20, 50 and 100 battles. It shows that in this simple situation the agent is able to quickly learn the best strategy to defeat its opponent, even just after 20 disputed battles. In all test sets, the agent showed that its efficiency is improved along the remainder 100 battles and this result shows that it is in a continuous process of learning. In all results, the agent also presented best results than the results of Micic et al. [2001], where the best result was 72% of victories at the same situation.

Table 2: Results of evaluation 1

Set	Wins after 20 battles	Wins after 50 battles	Wins after 100 battles
1	90%	94%	96%
2	90%	94%	96%
3	85%	94%	97%
4	85%	92%	96%
5	75%	86%	93%

Figure 2 shows the amount of total wins obtained by the agents considering the 5 tests sets. This means, if the agent losses the first battle in all 5 evaluation sets, the Cartesian value will be (0,1). If the agent wins three times the first battle in all 5 evaluation sets, then the Cartesian position will be (3,1). This idea is repeated to all 100 battles of each evaluation set, generating the line in the graph below. This graph also shows a straight line that represents the trend of wins. Thus, for this scenario, this trend line shows the improvement of the agent efficiency along the time and the usefulness of the acquired knowledge.

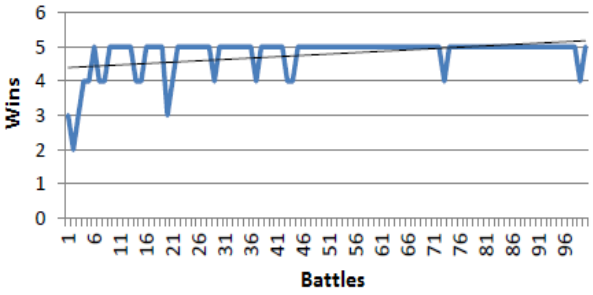

Figure 2: Graphic representation of wins in the evaluation 1

4.2 Evaluation 2 – 4 Protoss Dragoons x 4 Protoss Zealots

This evaluation was also carried out in [Micic et al. 2011] and it is similar to the previous scenario. However, the agents now need to consider the need of cooperation between them, rather than just engaging in the behavior discussed in the previous scenario (*kitting*

strategy). This means each agent of a team needs to assist their joint units during the battle. Thus, this multiagent situation is more complex and the aim of this evaluation is to check if our learning proposal maintains the efficiency showed in the single agent scenario. Furthermore, this multiagent scenario is more common in real game situations.

Table 3 presents the results obtained in this scenario. The numbers of this table shows that the increase of the situation complexity also increases the difficulty in finding the appropriate strategy to its behavior. Thus, the learning process is longer and positive results require more time to appear. Apart this fact, the agent had a relatively rapid learning, obtained more than 50% of wins in four evaluation sets after 20 games. In all cases, we had a better efficiency with the increase of the number of battles and, after 100 battles, the agent always obtained victories. The agent also presented better results than the work of Micic et al. [2011] that modified its modelling to consider this situation. Ever after this modification, the results showed only 32% of wins.

Table 3: Results of evaluation 2

Set	Wins after 20 battles	Wins after 50 battles	Wins after 100 battles
1	35%	62%	76%
2	90%	92%	93%
3	65%	78%	88%
4	60%	68%	77%
5	60%	74%	84%

The next graph (Figure 3) also shows that there is a trend in the number of wins in accordance with the number of battles, as in the previous case.

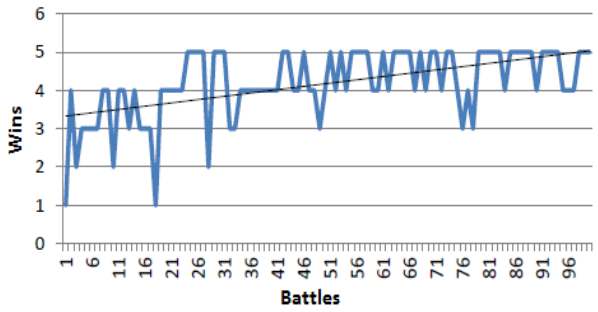

Figure 3: Graphic representation of wins in the evaluation 2

4.3 Evaluation 3 - 3 Zerg Hydralisks x 3 Terran Vultures

This evaluation represents a balanced combat situation. The *Hydralisks* from the *Zerg* race are units very similar to the *Vultures* from the race *Terran*. Both units are able to long distance attacks and the attack range of Vultures is a bit longer than the Hydralisks range. Both units also have the same amount of life points and similar DPS (damage per second). In order, the Hydralisk attack causes half of the damage of a Vulture attack, although the Hydralisk executes the double of attacks than a Vulture in the same time. To win a battle, it is necessary to intelligently coordinate the moving of joint agents and the choice of their targets.

In this scenario, where agents have very similar attack conditions, the SARSA agents were able to learn an appropriate strategy, so that, at the end of 100 battles, they won the majority of these battles against their opponents in all evaluation sets (Table 4).

Table 4: Results of evaluation 3

Set	Wins after 20 battles	Wins after 50 battles	Wins after 100 battles
1	40%	46%	54%
2	50%	68%	71%
3	65%	70%	75%
4	45%	58%	76%
5	30%	40%	56%

Following the results of the previous experiments, this scenario also presented a positive learning trend (Figure 4), so that the number of wins was proportional to the number of battles. At the end of 100 battles, the agent obtained a positive average of 66,4% of wins.

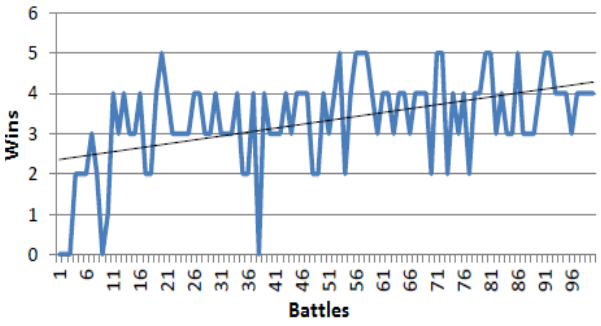

Figure 4: Graphic representation of wins in the evaluation 3

4.4 Evaluation 4 – 1 Protoss Draggon + 1 Protoss Zealot x 3 Terran Marines + 1 Terran Medic

This situation places two strong units of the *Protoss* race, one *Dragoon* and one *Zealot*, against one Medic and three *Marines* that are the basic combat units of the *Terran* race. The *Dragoon* and *Zealot* are able to easily defeat the three *Marines*. However, in this situation, the presence of a *Medic* requires a change in the strategy. The reason is the ability of the *Medic* in curing its joint units. Thus, the Dragoon and Zealot must learn that, to have some probability of winning, they must focus their attack on the *Medic* unit, rather than the *Marines*, ever if they become easy targets during some time interval. Then, after eliminating the Medic, they can attack other units (*Marines*). Table 5 shows that the agents were able to learning this strategy and improve their performance in this scenario.

Table 5: Results of evaluation 4

Set	Wins after 20 battles	Wins after 50 battles	Wins after 100 battles
1	20%	28%	46%
2	50%	52%	58%
3	35%	58%	60%
4	40%	62%	69%
5	15%	38%	48%

Although the agent finished the 100 battles with a slight negative retrospect in two evaluation sets, the average number of wins, considering the five evaluation sets, was 56,2%. The main reason was the complexity of this scenario, where the agent required a higher number of battles to be able to develop a winner strategy.

Even considering the higher complexity of this scenario, the agents were able to present a continuous learning behavior (this means, to attack the *Medic* unit) as shows the learning trend line in the graph below (Figure 5).

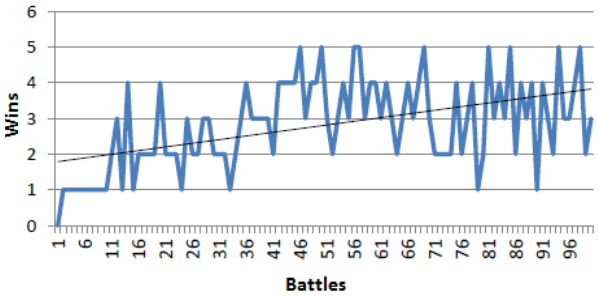

Figure 5: Graphic representation of wins in the evaluation 4

4.5 Evaluation 5 – 2 Terran Vultures + 1 Terran Marine x 2 Zerg Hydralisks + 2 Zerg Zerglings

This evaluation shows that the modelling proposal also enables the learning in situations with different types of agents. As discussed during the Evaluation 3, the *Vultures* are basically equivalent to the *Hydralisks*. The Marine unit is an unit stronger than a Zergling, however it is considered inferior against two Zerglings. Thus, the Terran units have an initial disadvantage and, to win the battle, they must efficiently coordinate their attacks to overcome the higher offensive power of the opponents. Note that this case does not present an obvious strategy to be used. Table 6 shows the results for this scenario.

Table 6: Results of evaluation 5

Set	Wins after 20 battles	Wins after 50 battles	Wins after 100 battles
1	30%	40%	48%
2	35%	42%	48%
3	30%	38%	46%
4	25%	40%	52%
5	25%	36%	45%

The results of this table show that in fact this is the most complex learning situation to the agents. However, we can still note an evolution of the agents along the ongoing battles. This fact is interesting because SARSA agents compose the weaker team, but they are able to acquire almost 50% of the victories after 100 battles, overpassing this value in one of the evaluation set. At the end, the agents obtained an average of 47,8% of victories. However the trend learning line (Figure 6) shows that this behavior could be better than 50% if we had more battles in our experiment.

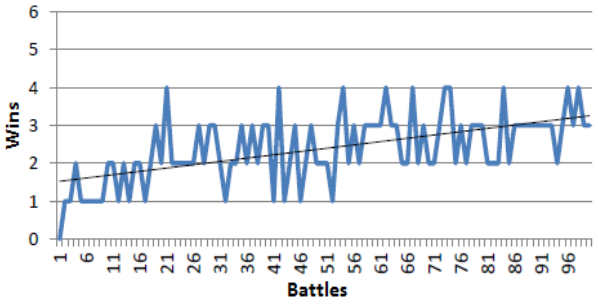

Figure 6: Graphic representation of wins in the evaluation 5

4.6 Summary

Table 7 presents the summary of the results obtained in all testes (evaluations 1 to 5), showing the average of the victories obtained after 20, 50 and 100 battles. This table shows that, in all tests, the agents were able to improve their performance along the time, always obtaining a proportional efficiency to the number of battles. This result stresses that in fact the agents are in an ongoing process of learning.

Table 7: Summary of results, considering the evaluation1 to evaluation 5

Evaluation	Average wins after 20 battles	Average wins after 50 battles	Average wins after 100 battles
1	85%	92%	95,6%
2	62%	74,8%	83,6%
3	46%	56,4%	66,4%
4	32%	47,6%	56,2%
5	29%	39,2%	47,8%

5. Conclusion and Future Works

During the evaluations, we tried to represent several different game situations, where our SARSA modelling was used to evolve several races of the Starcraft: Brood War game. These evaluations were important to show that the SARSA modelling is eventually able to create agents more efficient than the default Starcraft agents. However, we cannot generalize this result to all game situations. Independently of this generalization, the use of adaptive/learning agents certainly brings an additional charming to RTS games once they provides

opponents with some sense of intelligence and, consequently, more challenging situations to human players.

To be in fact used in commercial games, this learning approach should be evaluated in more complex situations, which may require modifications in the modelling. Thus, for future works, a proposal is to extend the approach to other RTS game situations, which are naturally different from the combat scenarios discussed in this work. In this way, the agents could play a complete game against human players and adapt their actions regarding other situations such as resources gathering, technological research, general strategies and so on. This research direction presents a huge challenge in terms of game modelling, so that this modelling must enable that agents understand their environment and take intelligent decisions at the same time that the search space be maintained in a tractable size and still supports the learning process.

Acknowledgements

The authors would like to thank the National Counsel of Technological and Scientific Development for the financial support to this project.

References

AHA, D. AND MOLINEAUX, M. 2004. Integrating Learning in Interactive Gaming Simulators". *Challenges in Game AI:: Papers of the AAAI'04 Workshop* (Technical Report WS-04-04), 2004.

MICIC, A., ARNARSSON, D. AND JÓNSSON, V. 2011. Developing Game AI for the Real-Time Strategy Game Starcraft. *Technical Report, Reykjavik University*.

ANDERSEN, K., ZENG, Y., CHRISTENSEN, D. AND TRAN, D. 2009. Experiments with Online Reinforcement Learning in Real-Time Strategy Games. *Applied Artificial Intelligence*, Vol. 23, pp. 855-871.

BALLA, R. AND FERN, A. 2009. UCT for Tactical Assault Planning in Real-Time Strategy Games. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 40-45.

DIMITRIADIS, V. 2009. *Reinforcement Learning in Real Time Strategy Games Case Study on the Free Software Game Glest*. Technical University of Crete, Greece, 2009.

JAIDEE, U., MUÑOZ-AVILA, H. AND AHA, D. 2011. Case-Based Learning in Goal-Driven Autonomy Agents for Real-Time Strategy Combat Tasks". *ICCBR Workshop on Case-Based Reasoning in Computer Games*.

KOK, E. 2008. Adaptive Reinforcement Learning Agents in RTS Games. *M.Sc. Dissertation*. University Utrecht, Netherlands.

LAURSEN, R., AND NIELSEN, D. 2005. Investigating Small Scale Combat Situations in Real Time Strategy Computer Games. *M.Sc. Dissertation*. University of Aarhus, Denmark.

MICIC, A., ARNARSSON, D. AND JÓNSSON, V. 2011. Developing Game AI for the Real-Time Strategy Game Starcraft. *Technical Report*, Reykjavik University.

NETO, G. AND SIEBRA, C. 2012. Uma Proposta de Modelagem para uso do Aprendizado por Reforço na Definição de Táticas de Combate em Jogos de Estratégia em Tempo Real. *Proceedings of SBGames 2012*, Brasilia, Brazil.

PÉREZ, A. 2011. Multi-Reactive Planning for Real-Time Strategy Games. M.Sc. Report. Universit Autònoma de Barcelona, Spain.

PONSEN, M. 2004. Improving Adaptive Game AI with Evolutionary Learning". *M.Sc. Dissertation*. Delft University of Technology, Netherlands.

PONSEN, M., SPRONCK, P. 2004. Improving Adaptive Game AI with Evolutionary Learning. *Computer Games: Artificial Intelligence, Design and Education*, pp. 389-396.

PONSEN, M., URBAN-LEE, S., MUÑOZ-AVILA, H., AHA, D. AND MOLINEAUX, M. 2005 Stratagus: An Open-Source Game Engine for Research in Real-Time Strategy Games. *IJCAI Workshop on Reasoning Representation and Learning in Computer Games*.

RORMARK, R. 2009. Thanatos – A Learning Game AI". *M.Sc. Dissertation*. University of Oslo, Norway.

SAILER, F., BURO, M., AND LANCTOT, M. 2007. Adversarial Planning Through Strategy Simulation. *IEEE Symposium on Computational Intelligence and Games*, pp. 80-87.

SANDBERG, T. 2011. Evolutionary Multi-Agent Potential Field Based AI Approach for SSC Scenarios in RTS Games. *M.Sc. Dissertation*. IT University, Denmark.

WALTHER, A. 2006. AI for Real-Time Strategy Games. *M.Sc. Dissertation*. IT-University of Copenhagen, Denmark.

WATKINS, C. 1989. Learning from Delayed Rewards. *PhD Thesis*, University of Cambridge, Cambridge United Kingdom.