Design and Evaluation of an Extended Learning Classifier-Based StarCraft Micro AI

Stefan Rudolph(✉), Sebastian von Mammen, Johannes Jungbluth, and Jörg Hähner

Organic Computing Group, Faculty of Applied Computer Science, University of Augsburg, Eichleitnerstr. 30, 86159 Augsburg, Germany
{stefan.rudolph,sebastian.von.mammen, joerg.haehner}@informatik.uni-augsburg.de

Abstract. Due to the manifold challenges that arise when developing an artificial intelligence that can compete with human players, the popular realtime-strategy game *Starcraft: Broodwar* (BW) has received attention from the computational intelligence research community. It is an ideal testbed for methods for *self-adaption at runtime* designed to work in complex technical systems. In this work, we utilize the broadlys-used Extended Classifier System (XCS) as a basis to develop different models of BW micro AIs: the Defender, the Attacker, the Explorer and the Strategist. We evaluate theses AIs with a focus on their adaptive and co-evolutionary behaviors. To this end, we stage and analyze the outcomes of a tournament among the proposed AIs and we also test them against a non-adaptive player to provide a proper baseline for comparison and learning evolution. Of the proposed AIs, we found the Explorer to be the best performing design, but, also that the Strategist shows an interesting behavioral evolution.

1 Introduction

Starcraft and its expansion pack *Starcraft: Broodwar*[1] (BW, sometimes also referred to as only *Starcraft* or *Broodwar*), combined, are one of the most famous instances of real-time strategy (RTS) games. They were released in 1998 for PCs and since then nearly 10 million copies have been sold. Founded on this number and on a huge number of players attracted to the game until today, it is seen as one of the most successful RTS game to date. RTS games can be characterized by three main tasks that the player has to fulfill: (i) collecting resources, (ii) creating buildings/units and (iii) controlling the units.

BW takes place in a science fiction setting, where three species compete for dominance in the galaxy. This are *Terrans*, a human-like species, *Protoss*, a species that is very advanced in technology and has psionic abilities, and *Zerg*, an insect swarm inspired species. The game has been extensively used for competitions, i.e., tournaments and leagues. These competitions usually consist of several 1-on-1 matches.

[1] *Starcraft* and *Starcraft: Broodwar* are trademarks of Blizzard Entertainment.

G. Squillero and P. Burelli (Eds.): EvoApplications 2016, Part I, LNCS 9597, pp. 669–681, 2016.
DOI: 10.1007/978-3-319-31204-0_43

BW represents exactly the kind of training ground needed for testing and honing online learning methods and their capacity to function in complex real-world scenarios. BW challenges the learner through its great complexity, the arising dynamics, and the fact that the fitness landscapes targeted by the learner are self-referential [1]. In BW, we face a set of entities (units and buildings) that interact with an environment (map and units of other players) in non-trivial ways. Furthermore, the environment is only partially observable and brings different types of uncertainty with it. Compared to other games that have been used as scientific testbeds, such as Chess, Go or Poker, it creates a much bigger challenge. Another reason to chose BW as an application to test and hone online learning methods fit for real-world scenarios is the availability of an easy to use C++ library[2] that provides an interface to the game and therefore allows the development of artificial players as well as automated test runs of them.

Learning classifier systems, in particular variants of the extended learning classifier system (XCS), have been successfully deployed in various online learning tasks in real-world scenarios. In this work, we present an XCS-based model design for the artificial intelligence assuming the role of a player in *Starcraft: Broodwar*. The remainder of this paper is structured as follows. In Sect. 2, we touch upon various related works in the context of RTS and corresponding machine learning approaches. We also introduce XCS as the learning system our approach is based on. In Sect. 3 we detail our model and the specific *Starcraft: Broodwar* scenario it was developed for. Section 4 presents and discusses the results of our co-evolutionary learning experiments. Afterward, we conclude with a short summary and an outlook on potential future work.

2 Related Work

In this section, we first touch upon the numerous approaches to development and deploying artificial intelligence techniques and machine learning in the Starcraft domain. Second, we present the Extended Classifier System (XCS) as the machine learning system used as the learning method for BW AIs in this work.

2.1 AI Approaches in Starcraft

A recent survey covering bot architectures, i.e. the algorithmic architectures for automated players, is given in [2]. It identifies *learning and adaptation* as an open question in RTS game AI, which is addressed in this work. Numerous works in the field target prediction and handling uncertainty. In [3], for instance, a method is introduced to predict openings in RTS games. As another example, [4] presents an approach for estimating game states. In contrast, this work focuses on learning, but, the presented methods could be combined with the approach given here. Another direction of research is the exploration of methods for the engineering of bots. To this end, [5] proposes to follow the paradigm of agent-oriented

[2] https://github.com/bwapi/bwapi.

programming, and [6] presents a method for automated testing of bots. Some works concentrate on providing data sets of BW games, e.g., [7,8]. There are also works about making and executing plans, such as [9] that proposes a method for the opening strategy optimization, or [10], where a method for the navigation of units is presented. Another category of works are the ones that innovate on mechanisms of strategy selection, e.g., [11], or of choosing tactical decisions [12]. Recently, in [13], a framework for the generation of a complete strategy from scratch using a evolutionary approach has been proposed. Finally, there are several works on the control of units, e.g., [14], where a Bayesian network is utilized for the unit control, or [15], where Reinforcement Learning methods are applied to learn *kiting*, a hit and run technique for a special unit type. In this work, we propose the use of Learning Classifier Systems for providing an AI that both evolves new behaviours through evolutionary computation and hones and refines established ones through reinforcement learning. We provide four according AI designs which exhibit different focusses of the learning system's deployment.

2.2 Extended Learning Classifier Systems

A *Learning Classifier System* (LCS) has originally been proposed in [16] by Holland. Later, he reworked the idea and proposes what today is considered a standard LCS in [17]. The most common extension of his work is the *Extended Classifier System* (XCS) of Wilson. It has been originally introduced in [18].

Since we adopted this variant for this work, the essence of the XCS is presented now. The basic architecture of an XCS is depicted in Fig. 1. It represents a very elaborate learning system tailored towards real-world applications. Accordingly, in Fig. 1, we see that the XCS gets a situation description of the environment through detectors. The situation is in the basic version of the XCS encoded as a bitstring. The population consists of classifiers, which hold several values:

- The **condition** is a string of 0s, 1s and *don't care*s (often represented by X). The purpose of the condition is to determine, if the classifier matches the situation given by the detector. A match is given, if for every 0 in the situation there is a 0 or an X at same position in the condition.
- The **action** is also encoded as a bit string. The set of available actions is typically provided by the designer of the system and depends on the application.
- The **prediction** is a value that approximates the expected reward, given the action of this classifier is executed in the situations described by the condition. It is constantly adapted by taking new observations into account.
- The **prediction error** is a value that reflects how much the prediction deviated from the actual reward.
- The **fitness** expresses the accuracy of the prediction of the classifier.

The match set holds all classifiers that match the current situation. If it appears that this set is empty a covering procedure is started that generates classifier that match the given condition and propose a random action. Afterwards, the

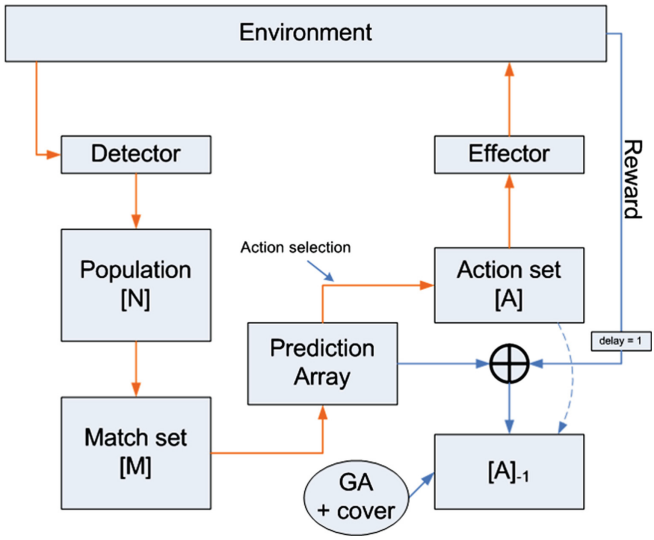


Fig. 1. The basic architecture of an extended learning classifier system or XCS.

XCS advances to the next step. Most often, the match set holds classifiers that suggest different actions. The purpose of the prediction array is to decide which action is applied. To this end, it uses a fitness-weighted average of the predictions for each action that is mentioned in the match set. The classifiers proposing the chosen action are transferred to the action set and the action is applied through the effector.

In the next step, a reward is provided by the environment that values the current state. The prediction, error and fitness values of all the classifiers in the previous action set are adjusted according to a given update rule, based on the reward. In addition, a genetic algorithm is applied to the action set in order to create more appropriate rules. It selects parent classifiers for generating new ones based on their fitness values and can apply different crossover and mutation operators, which is up to the designer of the system.

3 Approach

In this section, we first explain the general scenario the AIs we have developed had to train and prove themselves in. Based on this knowledge, it is easier to follow the motivation for their individual designs which follows next.

3.1 Competition Scenario

We let our AIs compete and train in so-called *micro* matches, which implies that each AI was only able to control one group of units. In the given scenario,

we did not consider the collection of resources and the production of units and buildings. A game is won, if all the enemy units or all the enemy buildings are destroyed. The match comes to a draw, if neither of the two competing AIs wins within a period of five minutes of simulated time. Each player starts out with the following heterogeneous set of predefined units which is a subset of the available zerg units.

Zergling. Each player has control over 64 Zerglings at the start of the match. They are light units dealing little damage and they can only suffer little damage before they are destroyed. Zerglings are melee units, which means they can only attack, if they are close to enemy units.

Hydralisk. Hydralisks can attack over distance but are not robust units, i.e., they should try to keep distance to the enemy units since they can be destroyed fast if they are attacked. Each player has control over 12 Hydralisks at the beginning of a match.

Ultralisk. Each player only has two Ultralisks at their disposition. They are heavy units that are very robust, i.e. they can sustain a high number of hitpoints. Like Zerglings, Ultralisks are melee-only units.

Scourge. Scourges are airborne units. Primarily, they attack other airborne units. At the loss of the Scourge unit itself, it can crash into other units to explode and damage the enemy. The player is provided with four Scourges at the beginning of a match.

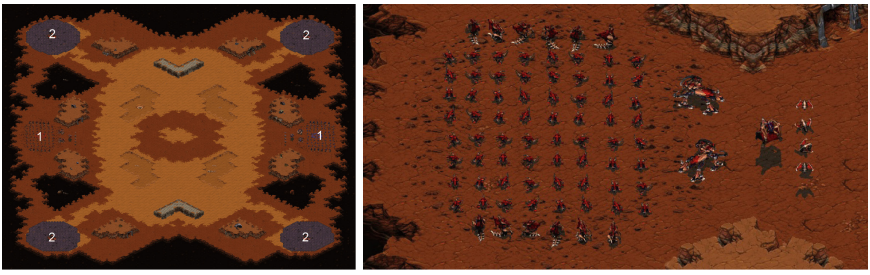
Zerg Queen. The Queen has no direct means of attack. Yet, it can slow down other units in a small quadratic area for 25 to 40 s, depending on the game speed. The enemies' movement velocity is halved, their rate of attack is reduced by between 10 to 33 %, depending on the affected unit type. In addition, the queen can hurl parasites at enemy units at a larger distances. The infested units' views extend directly add to the reconnaissance of the Queen's player.

The tournament map as well as the initial spatial arrangement of the given units is shown in Fig. 2. It has been established by the SCMAI[3] tournament. In Fig. 2a, we see the used map. The starting points of the players are marked with (1). At the positions marked with (2), there are buildings that can attack units that are within their range. Two of these buildings belong to each player. If a player destroys one of the opponent's buildings, additional Zerglings appear in the center of the map as a reinforcement. This is to encourage the players to engage and not just protect there own buildings.

3.2 AI Components

In analogy to the components of an extended learning classifier system (Sect. 2.2), we considered the following basic building blocks for creating an effective Starcraft AI. (1) Behavioral rules to classify and react to a given situation, (2) a reinforcement component to adjust the rules' attributes in order to

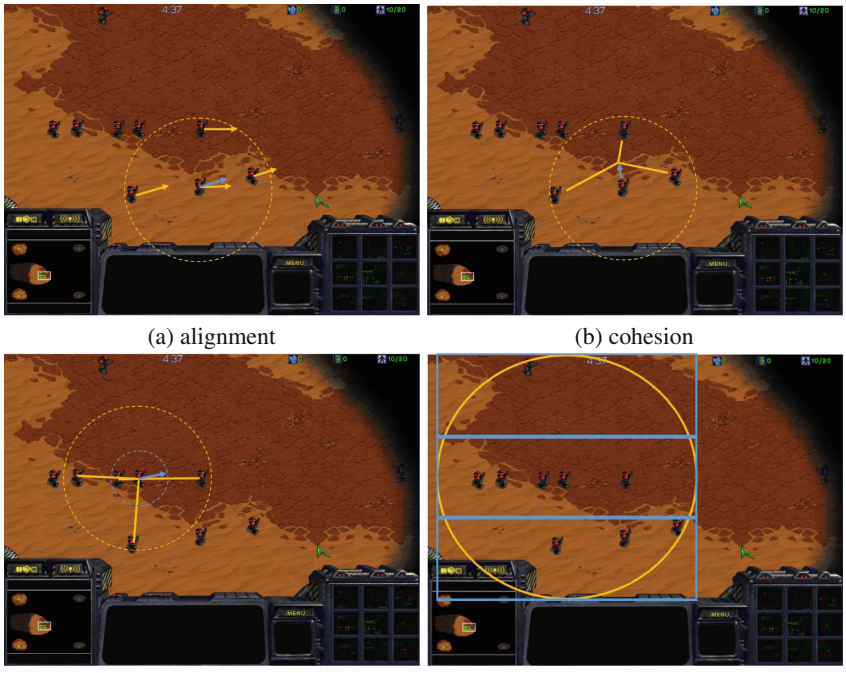
[3] Starcraft Micro AI Tournament.



(a) (b)

Fig. 2. (a) The map used for tournaments in our co-evolutionary experiment setup. (b) The initial spatial arrangement of the units made available to the AIs.

increase the achieved reward, (3) a covering mechanism to generate rules and fit newly encountered situations, and (4) a genetic algorithm for evolving the existing set of behavioral rules. In addition, we considered the ability to progress in battle formation based on individual *boid* steering urges [19], see Fig. 3. Different



(a) alignment (b) cohesion

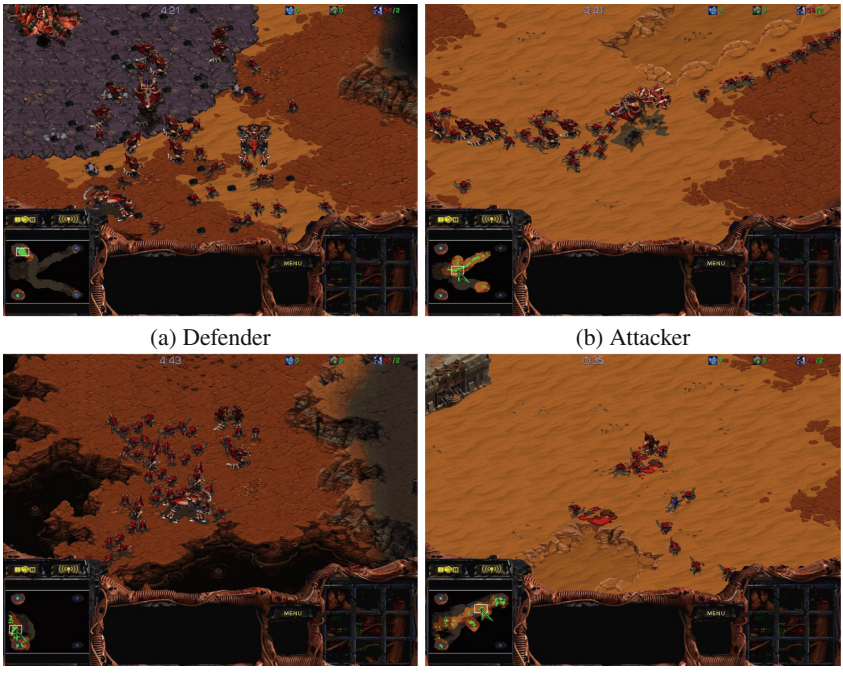
(c) separation (d) row formation

Fig. 3. The augmented screenshots (a) to (c) depict the steering urges as defined by Reynolds' flocking algorithm [19]. Based on these urges, formations emerge such as the one in (d).

from dynamically chosen but otherwise fixed formations, inferring the individual accelerations based on the units' neighborhoods results in emergent, adaptive formations [20]. In particular, the units sense their neighbors and (a) align their heading and speed with them, (b) tend towards their geometrical center, and (c) separate, if individual units get too close. As a result, battle formations such as the row formation in Fig. 3(d) emerge.

3.3 XCS-based AIs

We combined the AI components outlined above in four different ways, to trigger interesting competition scenarios and to trace and analyze the components' effectiveness. In particular, we implemented and co-evolved one defensive AI, one aggressive one, one that focuses on exploration and one where XCS takes global strategic decisions. Screen shots of their respective activities are seen in Fig. 4.



(a) Defender (b) Attacker

(c) Explorer (d) Strategist

Fig. 4. Screen shots of representative behaviors of the four implemented AIs. The Defender assembles his troops to defend the buildings. The Attacker storms towards the enemy's buildings to attack. The units of the Explorer swarm in different directions from the base. The Strategist decided on attacking enemy units.

The Defender. Right at the beginning of the match, all units move to the upper of two buildings on the map and stay there for its defense until the end. Using the full functionality of an XCS, the Hydralisks' as well as the Queen's behaviors are learned. For the Hydralisks, the condition part of the classifier rules considers the distance to the next visible enemy. Six actions are offered: (1) Approach and attack the closest ground or (2) airborne enemy, (3) move to a predefined point, (4) support a friendly unit, (5) protect the hatchery, or (6) burrow. For the Queen the proximity to the next enemy unit can trigger escape or *ensnare* airborne units or to hurl *parasites* at ground units. The XCS' reinforcement component positively rewards any attacks, whereas the other actions are only remunerated, if the player is attacked itself or if the units have built up a great distance to their buildings.

The Attacker. This is an offense-oriented AI. It will attack the next visible enemy. If none are in sight, the next enemy building is attacked. The units move in flocks, often they break into two clusters to attack both the enemy's buildings simultaneously. In perilous situations, however, an XCS may reinforce the attack or instigate retreat. Any kind of attack (the XCS also decides the Queen's mode of attack) is rewarded, suffering damage results in negative reinforcement.

The Explorer. The units are divided into two clusters that head into randomly chosen directions to explore the environment. When an enemy is sighted, an XCS determines to attack or to escape. Successful attacks directly translate into positive rewards, whereas loss of health points implies negative reinforcement. First strike is additionally greatly rewarded, whereas suffering a surprise attack results in an equally great loss—adding or subtracting 100 reward points, respectively. Similarly, winning and losing a match results in adding/deducting the comparatively small reinforcement value of 10.

The Strategist. Here, XCS is used to determine the overall strategy of the player. Based on the remaining time, the available and the opposing units, XCS determines whether to (1) attack enemy units or (2) buildings, whether to (3) defend one's buildings or (4) to idle. The respective strategies imply according convoy movements, if necessary. Independently of the strategy, the next sighted enemy is always attacked. The exhibited behavior is rewarded with the number of remaining units and buildings at the end of each match.

3.4 Learning Scenario

For the evaluation, we set up a learning scenario that addresses the issues of on-line learning, adaption and co-evolution. To allow this, we let the AIs compete with and learn from each other in several matches in a row. In particular, we first let the AIs train for 100 matches in a row with each of the other three AIs. In a second round, the previous adaptation is put to the test in the course of another 50 matches against each enemy AI. As all the AIs are designed to improve themselves by means of the combined reinforcement and evolutionary learning components of XCS, the tournament allows the AIs to co-evolve. Furthermore, for a better comparability of the approaches, we conducted additional

Table 1. Each of the four AIs trains with and competes against all the other ones. This table depicts the consecutive changes in the number of frames the simulation ran for, the health points successfully maintained by the AIs, and the frequency of winning situations.

Defender

enemy	*matches*	*frames*	*hp left*	*winner*
Explorer	100	+	+	0
Attacker	100	-	-	+
Strategist	100	+	-	-
Strategist	50	+	+	-
Explorer	50	+	+	+
Attacker	50	-	0	+

Strategist

enemy	*matches*	*frames*	*hp left*	*winner*
Attacker	100	-	+	-
Explorer	100	+	+	0
Defender	100	+	+	0
Explorer	50	-	0	-
Attacker	50	-	+	0
Defender	50	+	0	0

Explorer

enemy	*matches*	*frames*	*hp left*	*winner*
Defender	100	+	+	-
Strategist	100	+	-	-
Attacker	100	-	0	+
Attacker	50	-	0	-
Strategist	50	-	0	+
Defender	50	+	+	+

Attacker

enemy	*matches*	*frames*	*hp left*	*winner*
Strategist	100	-	+	+
Defender	100	-	-	+
Explorer	100	-	0	-
Explorer	50	+	+	-
Strategist	50	-	+	+
Defender	50	-	-	0

experiments that include a non-learning AI. Competing against a non-learning AI ensures that any observed improvements do not emerge from co-evolutionary dynamics but are the result of the individual learners themselves. In addition, the non-learning AI also provides a clear baseline against which all the other AIs can be measured against. The non-learning AI mainly defends its position by splitting the given units in two groups which defend the two buildings. It has no intention of winning the game by moving on to attack the enemy.

4 Evaluation and Discussion

Considering 450 matches, the Explorer with 269 won matches clearly represents the best designed AI. The Attacker is second best with 105 wins, followed by the Defender (30 wins) and the Strategist (7 wins). These performances are both the product of the AIs basic strategies but also of the sequence of their co-evolutionary learning experiences. Therefore, it is important to analyze the relative progress each of the AIs has achieved. An according, high-level summary is depicted in Table 1. It shows the consecutive changes regarding the number of frames a simulation runs, the number of health points left of a player and the number of experienced winning situations. A decrease in frames signals an increase in clarity regarding the winner, as draws become less likely and as quicker solutions take over. An increase of left-over health points of an AI may be considered an improvement. However, an AI may also learn to sacrifice more health points in order to win a match in the end. Therefore, an increase in wins statistically indicates an improved, i.e. learned, behavior.

However, as pointed out above, in an attempt to objectively compare the learning successes of each AI, we let all four of them train and compare against a simple, non-learning AI for another 200 matches. The results in terms of averaged fitness evolution as well as in terms of averaged prediction error can be

seen in Fig. 5. Although they do not seem to improve much, the Defender and the Attacker AIs have rather high average fitness values to begin with. Their averaged prediction error does not change over the course of the evolutionary experiment either. The average fitness of the Strategist AI, however, rises continuously and converges quickly, despite the fact that its average prediction error rises sporadically as well. The most likely explanation for this discrepancy is that the prediction errors rise so uniformly across the whole population of classifiers that a greater error value would not impact the selection and thereby the whole interaction process.

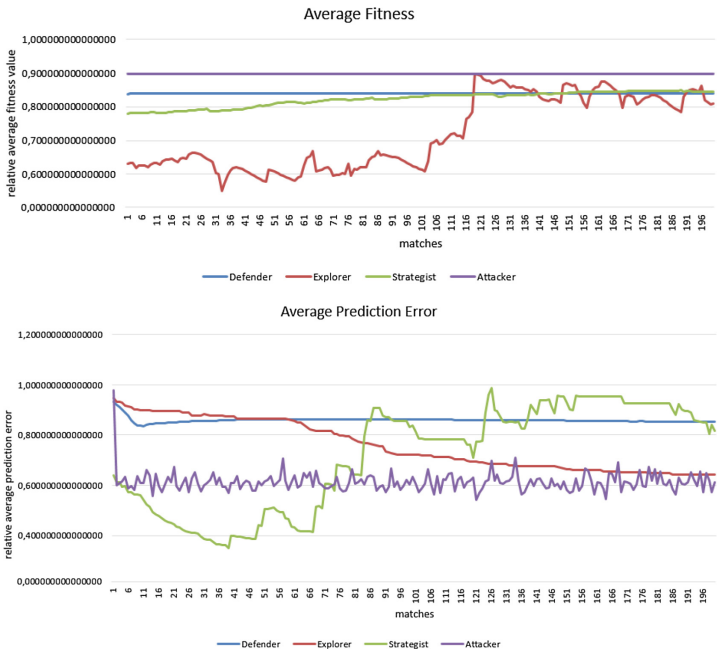


Fig. 5. The four XCS-based AIs' training progress when learning to compete against a simple non-learning AI.

4.1 Co-evolution Qualitatively

The Attacker AI reinforces its aggressive behavior, especially if it does not suffer from inflicted damages. As a consequence, it learns to ruthlessly exploit the Defender AI's feeble assaults by being even more fierce. When exposed to the other AIs, the Attacker AI quickly adapts to be slightly less aggressive. Similarly, the Strategist AI reinforces behaviors that minimize damage. As a result, when facing the Defender AI, the Strategist AI is not motivated to learn well-directedly, i.e., it does not tend to a more aggressive or defensive behavior. Instead, any behavior leads to success. When facing more aggressive opponents

such as the Attacker or the Explorer, the Strategist AI receives smaller rewards, almost independently of the ingenuity of a selected strategy. The Defender AI adapts its Hydralisks to shy away from enemies as they get destroyed too quickly, otherwise. Towards simple AIs, such as the simple non-learning AI mentioned above, the Defender AI increases aggressive behaviors. The Defender's Queen is mostly on the run, too, as otherwise the distance to the enemy's units becomes too small. The Explorer as the overall best designed AI presented in this work is discussed more in depth in the next section.

4.2 Decentralized XCS-Concept

After the description of the evaluation and results, we want to provide further details about the Explorer, the most successful AI in the tournament. It utilizes a decentralized XCS-concept, i.e., it adopts multiple XCS, that act in common. It uses one for each unit type (which are presented in Sect. 3.1). Each XCS decides whether the units of the respective type will engage or retreat when faced with an enemy. For the XCS the following configuration is used. Regarding the genetic algorithm, we empirically found the following parameters to be effective. Crossover is applied with a probability of 1 % and mutation with a probability of 1.5 % for each bit in a classifier. The parents are chosen by means of tournament selection with a tournament size of 5. The reinforcement component is configured with a learning rate of $\beta = 0.2$, i.e., the adaption of the prediction value is rather careful, and a discount factor of $\gamma = 0.71$ is used, i.e., future rewards are valued rather high. The reward is the difference between the damage the units have dealt and damage they have took, as it has been proposed in [15]. Additionally, there are some rewards that are considered in special situation. There is an extra reward of 10 if the game has been won and negative reward of -10 if the game is a draw or lost. The action selection is ϵ-greedy with $\epsilon = 0.02$, i.e., the action is selected randomly with a probability of 2 %, otherwise the best action is selected. Additionally, a battle formation procedure based on Reynolds' flocking algorithm is utilized, where the parameters in the algorithm are optimized by a genetic algorithm. Through the optimization, the player evolves a very tight formation, where the Queen is positioned in the center.

Based on this architecture, the Explorer exhibits the following behavior. If there are no enemy units in sight, it creates two separate swarms from the set of available units. These two groups go in an individual, randomly chosen direction in order to explore the map. Utilizing a formation for movement can lead to a tactical advantage, if the opponent's forces are met. The Strategist develops different behavior against the different opponents. If facing the Defender, it will attack immediately, since it appears that this enemy will not be harmful for the strategist. Against the other two AIs, the Strategist is more reluctant since these more offensive AIs tend to deal more damage then the defender.

5 Summary and Future Work

Concluding the work, a motivation for *Starcraft: Broodwar* as an ideal testbed for *self-adaption at runtime* is given in Sect. 1, followed by an overview over the state-of-the-art in the scientific developments in the BW domain and a short description of the XCS in Sect. 2. Next, in Sect. 3, four XCS-based AIs – the Defender, the Attacker, the Explorer and the Strategist – are presented. Furthermore, in Sect. 4, the evaluation scenario in a tournament scenario and against a non-learning AI with a special focus on the co-evolutionary behavior is presented and discussed.

Overall, we see two main results: The first is that the Explorer shows the best performance of all proposed XCS-based AIs. The second one is that, even though some players performed much worse than the Explorer, the *self-adaption at runtime* worked out for each player. This can be concluded since every player - after a period of adaption - shied away from the more aggressive AIs and, in turn, became more offensive against the more passive AIs.

For future work, we propose two directions. First, we want to refine the XCS-based approach for micro-management in BW. In particular, we want to provide all the units' degrees of freedom available to individual, decentralized XCS learners and also feed them with pre-processed data that indicates general trends in the evolution of the game by means of correlation factors (e.g. [21]) or ascertainment of structural emergence (e.g. [22]). Second, we want to develop an AI that considers a broader managerial scope including micro-management and strategic group activities. To this end, we deem a multi-layered AI architecture taking on different responsibilities through the consideration of different timescales and levels of abstraction a first important step [1].

References

1. Müller-Schloer, C., Schmeck, H.: Organic computing - quo vadis? In: Müller-Schloer, C., Schmeck, H., Ungerer, T. (eds.) Organic Computing - A Paradigm Shift for Complex Systems, pp. 615–625. Birkhäuser, Verlag (2011)
2. Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., Preuss, M.: A survey of real-time strategy game AI research and competition in StarCraft. IEEE Trans. Comput. Intell. AI Games **5**(4), 293–311 (2013)
3. Synnaeve, G., Bessière, P.: A Bayesian model for opening prediction in RTS games with application to StarCraft. In: Cho, S.B., Lucas, S.M., Hingston, P. (eds.) CIG, pp. 281–288. IEEE (2011)
4. Weber, B.G., Mateas, M., Jhala, A.: Applying goal-driven autonomy to StarCraft. In: Youngblood, G.M., Bulitko, V. (eds.) AIIDE. The AAAI Press (2010)
5. Shoham, Y.: Agent-oriented programming. Artif. Intell. **60**(1), 51–92 (1993)
6. Blackadar, M., Denzinger, J.: Behavior learning-based testing of StarCraft competition entries. In: Bulitko, V., Riedl, M.O. (eds.) AIIDE. The AAAI Press (2011)
7. Robertson, G., Watson, I.: An improved dataset and extraction process for StarCraft AI. In: The Twenty-Seventh International Flairs Conference (2014)
8. Weber, B.G., Ontañón, S.: Using automated replay annotation for case-based planning in games. In: ICCBR Workshop on CBR for Computer Games (ICCBR-Games) (2010)

9. Churchill, D., Buro, M.: Build order optimization in StarCraft. In: Bulitko, V., Riedl, M.O. (eds.) AIIDE. The AAAI Press (2011)
10. Hagelback, J.: Potential-field based navigation in StarCraft. In: IEEE Conference on Computational Intelligence and Games (CIG), Sep 2012, pp. 388–393 (2012)
11. Yi, S.: Adaptive strategy decision mechanism for StarCraft AI. In: Han, M.-W., Lee, J. (eds.) EKC 2010. SPPHY, vol. 138, pp. 47–57. Springer, Heidelberg (2011)
12. Synnaeve, G., Bessière, P.: A Bayesian tactician. In: Proceedings of the Computer Games Workshop at the European Conference of Artificial Intelligence 2012, pp. 114–125 (2012)
13. Garcia-Sanchez, P., Tonda, A., Mora, A., Squillero, G., Merelo, J.: Towards automatic StarCraft strategy generation using genetic programming. In: 2015 IEEE Conference on Computational Intelligence and Games (CIG), pp. 284–291, August 2015
14. Parra, R., Garrido, L.: Bayesian networks for micromanagement decision imitation in the RTS game StarCraft. In: Batyrshin, I., Mendoza, M.G. (eds.) MICAI 2012, Part II. LNCS, vol. 7630, pp. 433–443. Springer, Heidelberg (2013)
15. Wender, S., Watson, I.D.: Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar. In: CIG, pp. 402–408. IEEE (2012)
16. Holland, J.H.: Adaptation*. In: Rosen, R., Snell, F.M. (eds.) Progress in Theoretical Biology, pp. 263–293. Academic Press, New York (1976)
17. Holland, J.H., Reitman, J.S.: Cognitive systems based on adaptive algorithms. SIGART Bull. **63**, 49–49 (1977)
18. Wilson, S.W.: Classifier fitness based on accuracy. Evol. Comput. **3**(2), 149–175 (1995)
19. Reynolds, C.W.: Flocks, herds, and schools: a distributed behavioral model. Comput. Graph. **21**(4), 25–34 (1987)
20. Lin, C.S., Ting, C.K.: Emergent tactical formation using genetic algorithm in real-time strategy games. In: Proceedings of the 2011 International Conference on Technologies and Applications of Artificial Intelligence, TAAI 2011, Computer Society, pp. 325–330. IEEE, Washington (2011)
21. Rudolph, S., Tomforde, S., Sick, B., Hähner, J.: A mutual influence detection algorithm for systems with local performance measurement. In: Proceedings of the 9th IEEE International Conference on Self-adapting and Self-organising Systems (SASO15), held September 21st to September 25th in Boston, USA, pp. 144–150 (2015)
22. Fisch, D., Jänicke, M., Sick, B., Müller-Schloer, C.: Quantitative emergence - a refined approach based on divergence measures. In: 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Sep 2010, pp. 94–103 (2010)