Training a Reinforcement Learning Agent based on XCS in a Competitive Snake Environment

Johannes Büttner
Games Engineering
University of Würzburg
Würzburg, Germany
johannes.buettner@uni-wuerzburg.de

Sebastian von Mammen
Games Engineering
University of Würzburg
Würzburg, Germany
sebastian.von.mammen@uni-wuerzburg.de

***Abstract*—In contrast to neural networks, learning classifier systems are no "black box" algorithm. They provide rule-based artificial intelligence, which can easily be analysed, interpreted and even adapted by humans. We constructed an agent based on such a learning classifier system and trained it to play in a competitive snake environment by utilizing reinforcement learning and self-play methods. Our preliminary experiments show promising results that we plan to extend on in the future.**

***Index Terms*—Extended Classifier System, Game-playing AI, Reinforcement Learning, Snake**

I. Introduction

In recent years, game-playing artificial intelligence (AI) has in many games achieved super-human levels of competency, that were previously impossible. Deepmind's AlphaZero [1] uses a deep neural network to learn a policy used in Monte Carlo tree search and was able to beat a world-champion program for the games of chess, shogi, and Go each. To achieve this, it was trained solely by reinforcement learning (RL) with self-play.

While the feats achieved by artificial neural networks used in deep reinforcement learning have been impressive, there are multiple problems still to be solved to improve our understanding and the efficiency of AI. An inherent problem of artificial neural networks lies in them being a "black box", as even their designers cannot explain why the AI has come to a specific decision. The complex nature of deep neural networks has even led to a whole research field about the question why an AI made a specific decision, called "explainable AI" [2]. Other machine learning methods do not have these issues but are designed as transparent and interpretable algorithms. Alas, the research regarding them has been miniscule compared to deep neural networks. Another problem of the current state-of-the-art RL algorithms lies in the time consuming process. Even though the final results are exceptional, the initial version of the AI has no knowledge about the rules of a given game and takes very long to even get to an amateur's competency. For example, OpenAI Five was trained for 180 days, where each day consists of 180 years worth of games, before competing against and beating human players [3], [4]. By providing AI with knowledge about the game and basic correct behavior, the learning process could be kickstarted and the time to reach exceptional levels drastically reduced. While the name might be a rather confusing term nowadays, a "learning classifier system" is a rule-based machine learning algorithm, that has been originally proposed in 1978 [5]. The rules used by a learning classifier system are of a simple $IF \rightarrow THEN$ form and can easily be analysed by humans. Furthermore, these rules cannot only be read but also be written by humans. This way, learning classifier systems can be given rules that were crafted by domain experts, even before the training process started. Thus, the initial part of learning basic rules and strategies could possibly be reduced significantly. Therefore, our research aims to improve and we focus on learning classifier systems, which provide a rule-based, human-understandable machine learning framework that can be used for both supervised and reinforcement learning.

In this paper we present our approach on using reinforcement learning with self-play on an agent utilizing a learning classifier system for a competitive snake game.

The remainder of this paper is structured as follows. In Section II we provide relevant works regarding typical approaches to AI in snake games, the extended classifier system (XCS) and an improved variant of it, and previous usages of the XCS in games. In Section III we describe the learning environment and the reinforcement learning approaches we used. In Section IV we present experiments we conducted to evaluate different configurations of our system. We present the corresponding results in Section V. Finally, we conclude with a summary of the provided work and possible future work in Section VI.

II. Related Work

In this section we first describe the XCS framework. Second, a variant of the XCS using inductive reasoning is presented. Finally, we present relevant works using an XCS-based AI in games.

A. Extended Classifier System

The XCS [6] is built to receive input about an environment's state through detectors and return an action to change the environment's state (see Fig. 1). This action is then rewarded, which updates the inner mechanisms of the XCS. Original implementations of the XCS used binary strings as input. As real-world problems often need to be represented by real

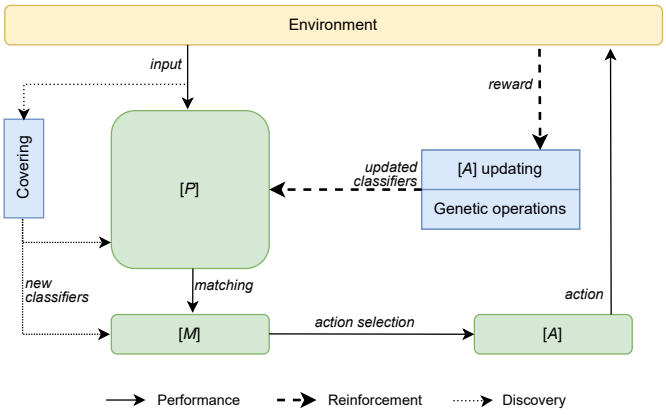

Fig. 1. Architecture of an XCS, adapted from [10].

values, there has since been a lot of research to enable such a representation in the XCS [7]–[9].

The XCS holds a population [*P*], that consists of classifiers. These classifiers are made of a condition and an action and save multiple variables, which are updated periodically. These values include:

- the **prediction**, which represents the expected reward,
- the **prediction error**, that holds information about the difference between the prediction and the actual reward,
- and the **fitness**, which represents the accuracy of the prediction.

After an input is given to the XCS, the match set [*M*] is constructed from all matching classifiers. If [*M*] is empty, a covering mechanism generates a classifier for this situation proposing a random action. By comparing the average of the classifiers' predictions weighted by their fitness, a best action is found. Each classifier proposing this action is added to the action set [*A*] and the action is returned to the environment. By receiving an according reward from the environment, all classifiers in the action set are then reinforced. Additionally, a genetic algorithm selects classifiers based on their fitness as parents and uses them to create more fitting classifiers.

B. XCS with Rule Combining

Traditional XCS approaches use a darwinian genetic algorithm to produce new and improved rules. In contrast, the XCS with rule combining (XCS-RC) uses inductive reasoning instead (see Fig. 2), which does not include random processes. Thus, the XCS-RC also introduces new hyperparameters:

- the **combining period** (Tcomb) describes after how many learning cycles the rule combining mechanism will be applied,
- the **prediction tolerance** (predTol) defines the maximum difference with which two classifiers can be combined
- and the **prediction error tolerance** (predErrTol) indicates inappropriate combining which can lead to the deletion of a rule.

Experiments have shown that the XCS-RC can learn correct behavior faster and with a smaller rule population than the traditional XCS. For more information about the differences between XCS and XCS-RC, and the aforementioned experiments, see [10].

C. XCS in Game Environments

There have been a few attempts to utilize a learning classifier system for a game-playing AI. Reference [11] details an approach to an AI playing Starcraft by learning with an XCS. Therein, multiple agents were trained and evaluated against each other. The agent with the highest win rate is constructed of multiple XCS instances, one for each unit type. Reference [12] proposes an AI agent that learns to play Othello using an XCS. The XCS agent was evaluated against and outperformed an agent based on the minimax algorithm and human players. Reference [13] proposes an algorithm that combines the XCS with opponent modeling for concurrent reinforcement learners in zero-sum Markov Games. Compared to neural network based algorithms, the XCS has a similar learning performance and the advantage of easily interpretable action selection rules.

While relatively little research has been done in this direction, the referenced publications show promising results on which further research can be built.

III. Method

In this section we first describe the game environment. Second, we detail the used methods and parameters for training the XCS agent.

A. Description of the Environment

The content of this paper is based on the "AI Snakes 2021" competition [14]. The competition's environment is that of a classic game of Snake. The player or agent controls a snake in a two-dimensional world, trying to eat as many apples as possible. Only one apple is available on the map at any time and after eating one apple, the next one appears on a random free field. By eating an apple, the snake increases its length by 1, starting with a length of 3. The game ends when the snake crashes either into the edge of the world or itself. In this competitive version of the game, a second snake is added to the environment. One snake is declared winner of a match

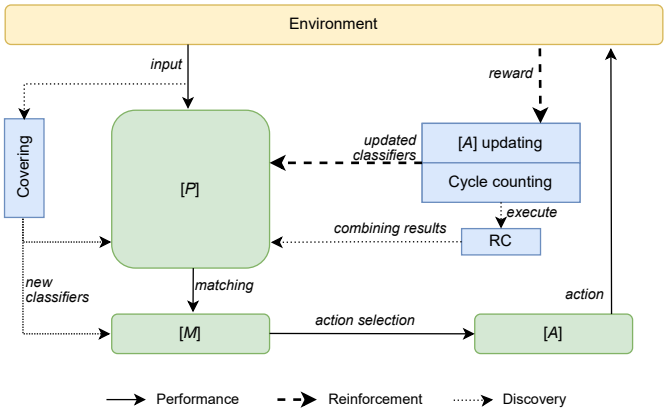

Fig. 2. Architecture of an XCS-RC, adapted from [10].

either if the opposing snake crashed into any snake or the edge of the world, or after 3 minutes, if it ate more apples than the opposing snake. While training our agent we used a map size of 14x14 fields. An example of the game's interface can be seen in Fig. 3. While the environment provides four actions an agent can take – one each for moving up, down, left and right – we simplified them due to the fact that moving backwards always results in an immediate loss. Therefore, the three actions our agent can take are moving forward, turning left and turning right. The environment provides information about the current position of the agent itself, the opposing snake, the size of the map and the current coordinates of the apple. Using these values we calculate a series of 14 booleans that is then given to the agent as input. These values describe whether the agent can move freely forward, left or right; whether an apple is directly in front, left or right of the snake's head; whether the apple is generally in front, behind, left or right of the snake's head; and whether the opponent's head is generally in front, behind, left or right of the snake's head. Since all of the input values are relative to the snake's position and direction only and no information about the map is given, the resulting behavior can be used on any map, regardless of size or shape.

B. Reinforcement Learning with Self-play

In reinforcement learning, an agent uses the information about the current state provided by the environment to decide which action to take. Depending on the outcome the agent receives a positive or negative reward and uses it to reinforce its behavior. Specifically in the field of learning to play games, RL is used extensively and specific methods like self-play have emerged. This method, where an agent learns by playing against itself, has proven very successful [15]. Therefore we used a general reward of 100 for winning a match and -100 otherwise. Additionally we used small rewards to improve the initial learning rate and reduce the time to learn basic game rules:

- +1 if the agent moves closer to the apple's position, -1 otherwise and
- +10 for eating an apple.

All rewards are propagated backwards through the previous actions with a discount factor of $\gamma = 0.71$ to enable learning of sequences of multiple steps. This rather unusual value for γ was adopted from the original proposal of the XCS [6].

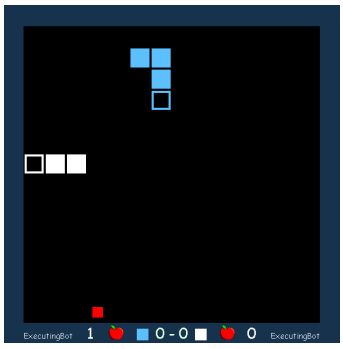

Fig. 3. The game's interface.

IV. Experiment

To find a good set of hyperparameters, we trained and evaluated different configurations of the XCS-RC using a combination of these values:

- maximum population size (maxPop): 200, 800, 2000
- Tcomb: 0, 10
- predTol: 10, 20
- predErrTol: 5, 100

Each configuration was trained by self-play for 5 iterations of 2500 learning games and 100 test games, where one agent was learning and the other used the current best set of rules. After winning 55% of the test games, the current best set of rules was replaced by the newly learned ruleset.

After the training completed, we tested all agents against each other to evaluate their strength. The results of the tournament can be seen in Table I. The set of hyperparameters to achieve the highest mean win rate used these values:

- maxPop: 2000
- Tcomb: 10
- predTol: 20
- predErrTol: 100

Further training of an agent with this configuration revealed the tendency to walk in circles. This way the agent would not die and beat their opponents most of the time, since running into a wall instantly loses the game. We theorize that after training against such an agent for a long enough time, exploration would lead to enough positive rewards after eating apples and not running into any obstacles and thus

TABLE I
Results of agent tournament. Best agent marked in grey.

Agent Index	Mean Win Rate in %
0	16,1
1	13,0
2	17,8
3	18,7
4	48,7
5	37,8
6	50,0
7	10,9
8	36,5
9	37,8
10	46,5
11	38,7
12	58,7
13	63,5
14	66,1
15	66,5
16	69,1
17	72,6
18	63,9
19	67,4
20	76,1
21	72,6
22	70,4
23	80,4

winning the game, to improve the agent's AI and change its behavior to a more advanced one. However, to reduce the time required, we adjusted the reward function. Since one wrong action can instantly lead to a loss, we changed the reward of a loss to only be -10, which lets the AI explore its actions more freely. After these results, we used the previously acquired overview of sensible hyperparameters to manually test promising configurations and found these values to work well:

- maxPop: 2000
- Tcomb: 0
- predTol: 20
- predErrTol: 100

This configuration was then further trained in self-play.

V. Results

The resulting AI could reliably find the shortest way to the apple. It learned to not run into walls, itself or the other snake. However, in very rare cases this can still happen. The agent has no possibility to simulate the game. Thus, it can easily get into situations where the next action does not directly lead to a loss but to a situation where no further actions can lead to a win, e.g. by being backed into a corner by the opponent. For the same reason, both snakes often run heads on into each other, resulting in a draw.

As can be seen in Fig. 4, the average prediction error over all rules is decreased by training. The average fitness over all rules (see Fig. 4) shows a general increasing tendency, which indicates an improving prediction accuracy. Interestingly, a great decrease in the average fitness and a great decrease in the average prediction error happened at the same time. These two facts seem to contradict each other at first, but could be explained by the fact that a new rule set was created that could make better predictions than previous rule sets. The initial fitness value of these rules had not yet been adjusted, but steadily increased afterwards.

Overall, it can be seen that using reinforcement learning on an AI using the XCS-RC improves its capabilities of playing a competitive version of snake. However, the AI converges at an average prediction error between 60 and 65. Thus, possible advancements of the implementation are discussed in the next section.

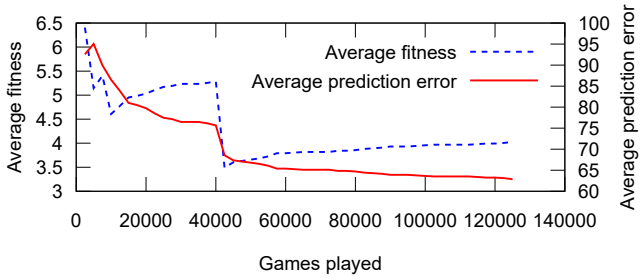

Fig. 4. Average fitness and prediction error of each rule

VI. Conclusion and Future Work

In this paper, we have proposed the usage of learning classifier systems as reinforcement learning components for game-playing AI and shown their general suitability through a series of experiments. The resulting behavior lead to efficient movement towards the apple while mostly dodging obstacles. Problems included the inability to plan ahead, since all information given regarded the current game state and the agent had no ability to simulate further steps.

The presented work represents the first step towards using the XCS in game-playing AI and shows that the XCS is able to learn how to play a turn-based strategy game. At this point, the AI we developed only uses the current state of the game and does not utilize any search mechanisms, thus the resulting AI is far from perfect. Next steps could include implementing a simulation mechanism such as Monte Carlo tree search with an XCS learning the used policy to further improve our results. Thus, the agent would be able to plan multiple steps ahead and significantly improve its performance. Additionally, one of the described qualities of learning classifier systems is the possibility to analyse their rules and insert additional rules crafted by domain experts. This factor has not been used to its full extent yet and the possibilities to improve the AI and its training process by rule inspection need to be researched further in this context.

References

[1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[2] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (xai)," *IEEE Access*, vol. 6, pp. 52138–52160, 2018.

[3] OpenAI, "Openai five." https://blog.openai.com/openai-five/, 2018.

[4] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. W. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," *ArXiv*, vol. abs/1912.06680, 2019.

[5] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," in *Pattern-directed inference systems*, pp. 313–329, Elsevier, 1978.

[6] S. W. Wilson, "Classifier fitness based on accuracy," *Evol. Comput.*, vol. 3, p. 149–175, June 1995.

[7] S. W. Wilson, "Get real! xcs with continuous-valued inputs," in *Learning Classifier Systems* (P. L. Lanzi, W. Stolzmann, and S. W. Wilson, eds.), (Berlin, Heidelberg), pp. 209–219, Springer Berlin Heidelberg, 2000.

[8] C. Stone and L. Bull, "For real! xcs with continuous-valued inputs," *Evol. Comput.*, vol. 11, p. 299–336, Sept. 2003.

[9] N. Fredivianus, K. Kara, and H. Schmeck, "Stay real! xcs with rule combining for real values," in *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, (New York, NY, USA), p. 1493–1494, Association for Computing Machinery, 2012.

[10] N. Fredivianus, *Heuristic-based Genetic Operation in Classifier Systems*. PhD thesis, University of Karlsruhe, 2015.

[11] S. Rudolph, S. V. Mammen, J. Jungbluth, and J. Hähner, "Design and evaluation of an extended learning classifier-based starcraft micro ai," in *EvoApplications*, 2016.

[12] S. Jain, S. Verma, S. Kumar, and S. Aggarwal, "An evolutionary learning approach to play othello using xcs," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 2018.

[13] H. Chen, C. Wang, J. Huang, J. Kong, and H. Deng, "Xcs with opponent modelling for concurrent reinforcement learners," *Neurocomputing*, vol. 399, pp. 449–466, 2020.
[14] L. J. P. de Araujo, J. A. Brown, and A. Grichshenko, "AI Snakes 2021." https://agrishchenko.wixsite.com/aisnakes2021, 2021.
[15] A. Plaat, *Learning to Play: Reinforcement Learning and Games*. Springer Nature, 2020.