Adaptive Agents in 1v1 Snake Game with Dynamic Environment

1st Hampus Fink Gärdström
The Mærsk Mc-Kinney Møller Institute
University of Southern Denmark
Odense, Denmark
hamp@mmmi.sdu.dk

2nd Henrik Schwarz
The Mærsk Mc-Kinney Møller Institute
University of Southern Denmark
Odense, Denmark
hschw17@student.sdu.dk

3rd Marco Scirea
SDU Metaverse Lab
University of Southern Denmark
Odense, Denmark
msc@mmmi.sdu.dk

***Abstract*—This paper delves into the adaptability of Proximal Policy Optimization (PPO)-trained agents within dynamic environments. Typically, an agent is trained within a specific environment, learning to maximise reward acquisition and to navigate it effectively. However, alterations to this environment can lead to performance deficiencies. Existing research does not fully elucidate how the training of agents influences their adaptability in different environments and which parameters significantly impact this. This study aims to fill this gap, contributing to the creation of more versatile intelligent agents.**

The objective of this study is to explore how training agents in various environments affects their adaptability when introduced to unfamiliar environments. To this end, 36 models were trained using 36 different configurations to play a one-versus-one (1v1) Snake game. These models were subsequently compared against each configuration to measure their adaptability.

The results reveal that map size substantially affect the adaptability of agents in different environments. Interestingly, the results showed that the most adaptive agents were not those trained on the most expansive and complex environment, but rather the simplest.

***Index Terms*—snake game, multi-agent, reinforcement learning, adaptive agents, proximal policy optimization**

I. Introduction

The intersection of Reinforcement Learning (RL) and Multi-Agent Systems (MAS) in gaming presents a compelling research area. This field primarily explores the training of agents through self-play, relying solely on the game state and potential actions. Notable achievements include Google DeepMind's AlphaGO Zero algorithm [1] which mastered the game of Go within 24 hours, and AlphaStar which plays Starcraft 2 at grandmaster level [2].

Recently, algorithms such as Q-learning and Proximal Policy Optimization (PPO) have gained popularity. While existing research examines the performance of reinforcement-trained agents in static environments using Q-learning algorithms, this paper shifts the focus to the PPO learning algorithm [3]. Specifically, we investigate the adaptability of trained models in dynamic environments.

This study employs a 1v1 Snake game as a sandbox for this investigation inspired by and built upon the international BattleSnake[1] competition [4], where AI agents compete in a game of four-players Snake. The traditional Snake game, a single-player game developed by Taneli Armanto and launched as pre-installed software on the Nokia 6110 mobile phone in 1998, involves the player guiding a snake to gather apples in an attempt to achieve the highest possible score. The introduction of multiple snakes adds complexity to the domain. For this initial study, we limit the game to 1v1, although the framework supports more players.

The objective of this study is to examine the effects of training agents in a specific environment and compare their performance in a different environment.

Our research seeks to contribute to the broader understanding of agent generalised learning. This knowledge could potentially aid the development of more robust and versatile intelligent agents capable of transitioning between different environments and tasks, thereby enhancing their applicability and use.

[1]https://battlesnake.com

979-8-3503-5067-8/24/$31.00 ©2024 IEEE

II. Method

In this section the implementation of the game, the feature design and the reward structure of the agent is detailed. Additionally, the details on how and which models were trained are described along with how the adaptability experiment was performed.

A. Snake implementation

The game was implemented as a multi-agent PettingZoo [5] environment for 2 agents. In algorithm 1 pseudo-code for the step function of the game is shown. The function is run at each update of the game. It takes the actions of each snake as input, and for each snake derives the next position based on their respective action. Depending on whether certain conditions are met the snake is rewarded or terminated. If the snake collides with an obstacle or goes out of the map bounds it is terminated and negatively rewarded. If it discovers food it is rewarded, if not it is instead negatively rewarded and loses its tail. Depending on whether the snake got closer to food it is also rewarded.

Finally, the next position is set as the new head of the snake. Once each snake is processed, walls and food are added based on a 20% chance and given that the max limit of each item has not been reached inside the map. The placement is random. In

case the max limit of walls has been reached then a random wall is removed and a new wall is placed in a random location.

Terminated agents do not receive further actions and are therefore no longer updated. The game lasts until both agents have been terminated or a set number of max game iterations has been reached. In order to limit the runtime of the game and prevent the model running each episode too long, a limit of 5.000 step iterations was set for each game run, and if it was exceeded the game will terminate all agents thereby ending the game.

Algorithm 1 Step Function

```
function STEP(actions)
    if not actions then
        return {}, {}, {}
    end if
    rewards ← {agent | agent ∈ agents : 0}
    observations ← {agent | agent ∈ agents : getObservation(agent)}
    terminations ← {agent | agent ∈ agents : False}
    for each agent ∈ agents do
        snake_head ← getHead(agent)
        next_pos ← move(snake_head, actions[agent])
        if posInvalidOrBlocked(next_pos) then
            killAgent(agent)
            rewards[agent] ← −29
            terminations[agent] ← True
            continue
        end if
        if posIsFood(next_pos) then
            rewards[agent] ← 27
        else
            removeTail(agent)
            rewards[agent] ← −0.3
        end if
        if agentGotCloserToFood(next_pos, agent) then
            rewards[agent] ← 0.3
        end if
        addHead(next_pos, agent)
    end for
    addFoodRandomly()
    addWallsRandomly()
    return rewards, observations, terminations
end function
```

B. Features Design

The feature design was inspired by the approach taken by Yuhang *et al.* [6]. The state input for the learning algorithm is the same where the 16 directions were chosen and the same five observation parameters were chosen due to their demonstrated adequacy, however they were adjusted slightly after testing to perform better in this implementation of snake.

The first two parameters visibility of the snake body and food are boolean values (1 and 0) that describes whether a part of the snake's body or a piece of fruit can be found in the square. This provides the algorithm with immediate information of the surroundings of the snake.

The next three parameters provide the snake with a representation of the global state of the game without having to provide the whole board. The distance to body uses the Manhattan Distance[2] to calculate the distance to the nearest body part and should help the algorithm with understanding confinement and optimizing for having available state. Similarly, distance to the nearest food and distance to the nearest wall, are both also defined using Manhattan distance. These parameters are designed to optimize finding food and avoiding collision with walls.

The action space for the game is defined as an integer value from 0 to 3 that are defined as up(0), down(1), left(2) and right(3).

C. Reward Structure

Rewards were based on conditions described by Yuhan [6] where the initial values were the same and subsequently adjusted based on preliminary testing. The preliminary testing adjustments resulted in setting the reward for obtaining food at 27, a penalty of -0.3 for in-action i.e., not eating food, -29 for a death and 0.3 reward for moving closer to food.

D. Model training

A total of 36 game scenarios, or game parameter combinations, were devised from a range of parameters listed in table I, where all permutations that change the environment were included. The game parameters were selected as they were envisioned to significantly impact the behavior and need of the agent and were therefore suitable for the experiment on adaptability. The implementation of PPO from Stable Baselines3 [7] using the MlpPolicy was used to train the models. The hyperparameters underwent brief testing and changes were made to the step size and batch size to speed up training time, the used hyperparameters are listed in table II; the entries in bold are the ones that differ from the default configuration. All models were trained to 25 million total timesteps using a CPU device. The total amount of timesteps was selected, as during preliminary testing, it appeared to provide adequate results and it made it possible to train all 36 models within a reasonable time frame.

TABLE I
GAME SETTINGS

Parameter	Variables
Map Size	5, 11, 19
Food Total Max	2, 10, 15
Walls Enabled	False, True
Walls Max	2, 10, 15

[2]Manhattan Distance Wikipedia

TABLE II
HYPERPARAMETERS

Hyperparameter	Value	Hyperparameter	Value
Policy	***'MlpPolicy'***	***Batch Size***	***8000***
Learning Rate	0.0003	GAE Lambda	0.95
n_steps	***32000***	Clip Range	0.2
n_epochs	10	Gamma	0.99
Normalize Advantage	True	Entropy Coefficient	0.0
VF Coefficient	0.5	Max Grad Norm	0.5
Use SDE	False	SDE Sample Freq	-1
Rollout Buffer Class	None	Rollout Buffer Kwargs	None
Target KL	None	Stats Window Size	100
Tensorboard Log	None	Policy Kwargs	None
Verbose	0	Seed	None
Device	**'cpu'**	_init_setup_model	True

E. Experiment

The experiment was executed by running each model against every combination of game parameter configurations listed in table III. This totalled 1296 (36 models $\times$ 36 configurations) different model combination pairings, and each pairing was evaluated for 500 episodes running through an entire game, from which average and max metrics were collected. These values were collected into a CSV-file for processing.

The metrics collected were the average of the total reward, snake size, food eaten and moves taken for every agent across all. Moreover, the maximum of these values across all episodes of an evaluated model $\times$ configuration pairing was also recorded.

III. RESULTS

The results of the experiment produced multiple graphs for analysis.

Figure 1 displays the average performance of each model across all game configurations. Every metric for the models has been averaged over all game parameter configurations it has been evaluated against. The data shows that certain models perform consistently worse or better than others across all game parameter configurations, specifically models trained on combinations with larger maps sizes compared to smaller. Models 9 to 13, 21 to 24 and 33 to 36, all seem to perform under the other models that have a smaller map size. Model 11 with a max wall size of 10 also performs consistently better better than model 10 and 12 with max walls parameter 5 and 15. When comparing other models that also have a max wall parameter of 10 and their neighboring models with parameters of 5 and 10 the same cannot be found.

The average value charts seem to be more consistent with less variance where the with max value have a bigger variance in value.

IV. CONCLUSION

This study offers initial insights into the impact of training environments on the adaptability of agents, particularly those environments that may impede learning.

TABLE III
PARAMETER CONFIGURATIONS

Id	Food Max	Map Size	Walls Enabled	Max Walls
1	2	5	False	0
2	2	5	True	2
3	2	5	True	10
4	2	5	True	15
5	2	11	False	0
6	2	11	True	2
7	2	11	True	10
8	2	11	True	15
9	2	19	False	0
10	2	19	True	2
11	2	19	True	10
12	2	19	True	15
13	10	5	False	0
14	10	5	True	2
15	10	5	True	10
16	10	5	True	15
17	10	11	False	0
18	10	11	True	2
19	10	11	True	10
20	10	11	True	15
21	10	19	False	0
22	10	19	True	2
23	10	19	True	10
24	10	19	True	15
25	15	5	False	0
26	15	5	True	2
27	15	5	True	10
28	15	5	True	15
29	15	11	False	0
30	15	11	True	2
31	15	11	True	10
32	15	11	True	15
33	15	19	False	0
34	15	19	True	2
35	15	19	True	10
36	15	19	True	15

To thoroughly investigate the impact of varying game parameters on adaptability, we devised 36 different configurations for the 1v1 Snake game and trained a model on each of these environments. These models were subsequently evaluated across different game parameter configurations, yielding metrics that demonstrate each model's performance and adaptability.

We observe that map size was the most significant factor affecting both performance and adaptability of agents. Larger map sizes resulted in less proficient agents and posed greater adaptation challenges for agents not trained in such environments. This seems aligned with the results from Soemers *et al.* [8], and might be explained by how smaller game take less time to complete, so the model gets more training steps. The maximum amount of food and the presence of walls did not significantly impact training or adaptability. However, the existence of walls negatively affected both adaptability and training of agents.

The best performing agents were those trained in a small map size with no walls present. However, the most adaptable agents were the ones trained in small or medium map sizes with walls enabled.

In conclusion, this paper presents an initial contribution by

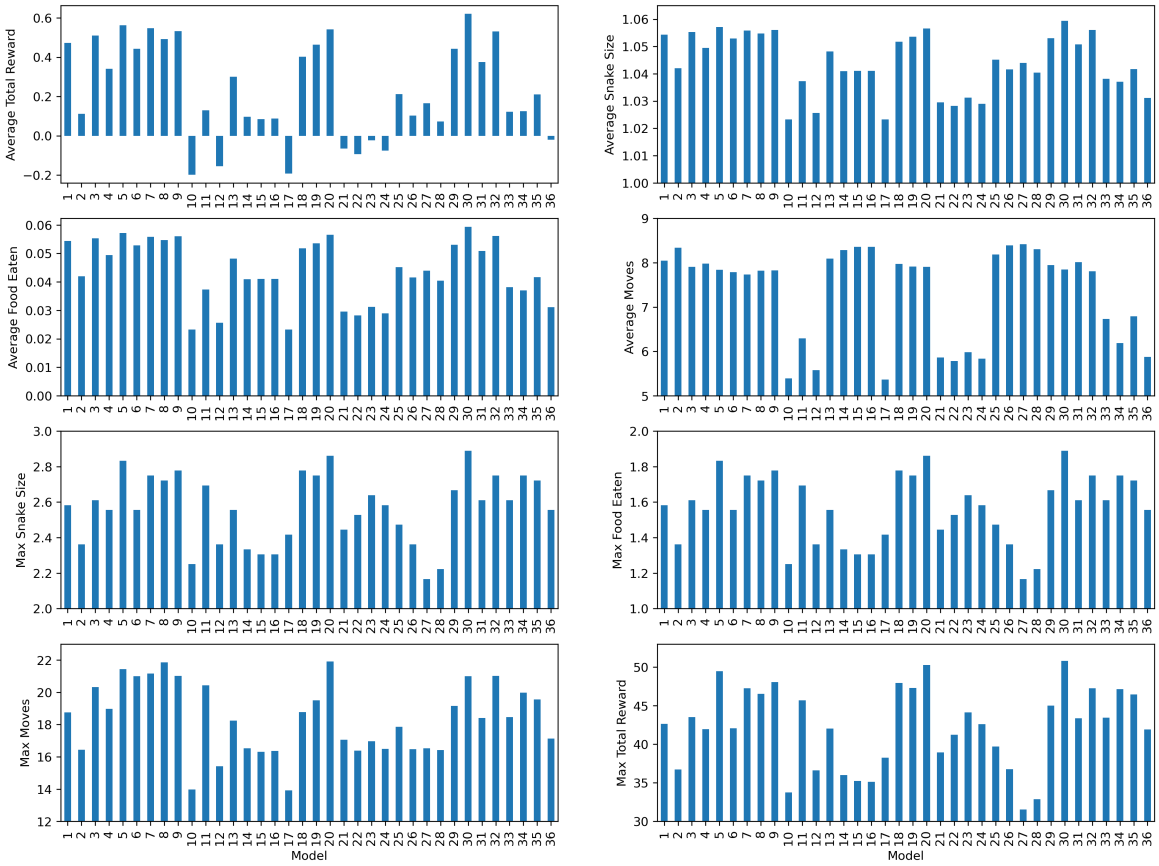

Fig. 1. Bar charts of all the measured metrics with the average value over all runs across all game parameter configurations for each model.

providing a comprehensive analysis of agent adaptability in dynamic environments.

REFERENCES

[1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[2] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[4] J. Chung, A. Luo, X. Raffin, and S. Perry, "Battlesnake challenge: A multi-agent reinforcement learning playground with human-in-the-loop," *arXiv preprint arXiv:2007.10504*, 2020.

[5] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente, *et al.*, "Pettingzoo: Gym for multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15032–15043, 2021.

[6] P. Yuhang, S. Yiqi, M. Qianli, G. Bowen, D. Junyi, and T. Zijun, "Playing the Snake Game with Reinforcement Learning," *Cambridge Explorations in Arts and Sciences*, vol. 1, July 2023.

[7] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[8] D. J. Soemers, V. Mella, É. Piette, M. Stephenson, C. Browne, and O. Teytaud, "Towards a general transfer approach for policy-value networks," *Transactions on Machine Learning Research*, 2023.