Deep Reinforcement Learning For Competitive Multi-Agent Systems
First year Master's project

Lucien Drescigh

UMONS, Faculty of Engineering (FPMs)

***Abstract*—The Agar.io online game is popular due to its intuitive gameplay and easy access on the internet. The game has a continuous action space and allows players to apply various complex strategies to compete with and defeat their opponents. The aim of this project is to use a reinforcement learning algorithm to train multiple agents to compete one against each other in the Agar.io game. Several reward systems and state representations are studied to find the best fit with the application of a Deep Q-learning algorithm. A form of curriculum learning was used to optimise twice faster the convolutional neural network that returns the Q-values according to the current state. Whether the state is represented in grayscale or RGB values of pixels, it offers the same performance at the end. A reward system related to the loss or gain in mass of the agents was found to be ineffective and gave the agents a strategy of escaping. A more traditional reward system provided proper results and showed that the agents can learn how to survive in a multi-player environment. When confronted with bots that move randomly, the agents eat around 5 times more opponents than the bots. In the same way, the agents get eaten a bit less than 15 times less than the bots.**

I. Introduction and motivations

Reinforcement learning (RL) is an area of machine learning which consists of an autonomous agent learning the actions to be taken in an environment, based on experiments, in order to optimise a cumulative reward over time. The agent is immersed in an environment and makes decisions based on its current state. In return, the environment provides the agent with a reward which can be positive or negative. The agent then seeks through iterated experiments an optimal decision-making behaviour in order to maximise the sum of rewards over time.

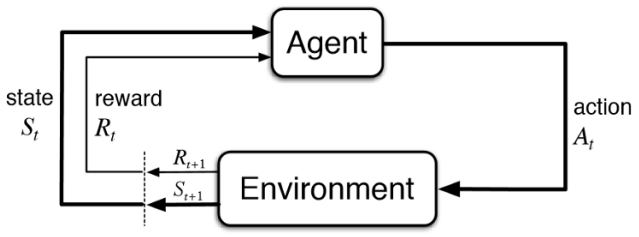

Figure 1. Reinforcement learning illustration [10]

The environment used in this project is based on the game of Agar.io, in which the player controls a circular cell in a 2-dimensional grid. The player's cell can eat small pellets that are spread out in the environment, or smaller cells from other players to increase its size. In the same way, the player must avoid being eaten by larger cells. The Agar.io environment is therefore quite interesting for research on RL since it mostly depends on the behaviour of other players and never stops evolving.

Deep neural networks have also shown a great efficiency at learning representations of complex environments. Convolutional neural networks (CNNs) have reached human-level performance by learning only from pixel values in Doom [6] and a wide range of Atari games [14].

This project studies how to use RL to build intelligent agents so that they show a competitive gameplay for the game Agar.io using a Deep Q-learning (DQN) algorithm.

The Q-learning algorithm [3] is a successful RL technique which does not require an initial model of the environment. The letter 'Q' stands for the function that measures the quality of an action performed in a given state of the system. It has already been used on the previously mentioned Doom [6], Atari games [14] and defeated world-class Go players [12].

Section II outlines an overview of related works. In section III, the fundamental principles of DQN are explained. Section IV describes the game Agar.io and the state representation. The experimental parameters and the neural network's structure are described in section V. The results are then discussed in section VI.

II. Related work

Mnih et al. [8] propose a DQN model to play various two-player Atari games. They showed that in certain games, DQN is able to find out long-term strategies. Their work demonstrates that their single architecture can successfully learn control policies in different environments with minimal prior knowledge. The model takes only pixels and the scores as input, and using the same DQN and hyperparameters for all games, considers the same inputs a human being would have, i.e RGB image. Their model's success is quite dependant on the incorporation of a replay algorithm, which tries to simulate the mammalian brain by storing and replaying

previous experiences.

Nil Stolt Anso et al. [9] also provided a Double DQN algorithm to train an agent to eat pellets. One aim of their paper is to compare the performance of the agent considering different representations.

One state representation is the usual player's view, i.e RGB image. Another representation uses the grayscale values of the pixels. The third state is a semantic state representation consisting of a vision grid laid out on the player's view. Each area unit of the grid returns a value based on the number of pellets it contains. This representation is therefore understandable to humans.

In this research, the game consists of only one agent. The two first representations seemed comparable to each other, and the semantic state representation seemed to outperform them. After 10 simulations of 300,000 training steps, for a resolution of 84x84 pixels and 2 convolutional layers, the vision grid shows a mean mass value of 494, where the Grayscale and RGB respectively obtain 439 and 427. However, it must be noted that the pixel value representations had not fully converged yet after this number of training steps and thus might match the performance of the vision grid if given more training steps.

Another aim was to compare the model's performance according to the resolution. They compared 63x63 and 84x84 pixel images, and concluded that the higher the resolution, the better the performance.

The aim of this project is to take a step further and study the performances of a simple DQN algorithm when several smart agents compete in the same grid.

III. REINFORCEMENT LEARNING

In the case of RL, the environment is usually described as a Markov decision process [11] since many RL algorithms use dynamic programming techniques in this context. In this type of process, an agent can perform an action in one state to reach a new state, for which it is given a scalar reward. The transitions between states have the Markov property. In other words, the stochastic transition probabilities between two states depend exclusively on the current state and the chosen action.

To model this RL problem as a Markov decision process, the state must be defined. The state is composed of the properties of the environment and the way the agent perceives the relevant available information. The game engine, in this case Unity, manages the transition from one state to another when an action is taken.

A. The State-Value Function

The value function V_π(s) is defined as the expected return when starting with the state s and successively pursuing the policy π. Basically, the value function judges "how good" it is to be in a given state.

$$V_\pi(s) = E[R] = E\left[\sum_{t=0}^{\infty} r_t \cdot \gamma^t | s_0 = s\right] \tag{1}$$

In RL, a state transition to a reward is mapped by a reward function. The agent must maximise its total expected reward obtained in the long run from the following reward function [9].

$$R = \sum_{t=0}^{\infty} r_t \cdot \gamma^t \tag{2}$$

Where R is the sum of future discounted rewards, r_t represents the reward given to the agent at a step t and γ is the discount factor [1]. Its value ranges between 0 and 1 so that events in the nearer future provide higher weights to the rewards. It is a mathematical trick used to bound an infinite sum.

B. Deep Q-learning

The Q-learning algorithm [3] predicts the quality (Q-value) of every action that can be taken regarding a specific state. The higher the value, the greater the reward the agent is likely to be offered according to a Q-table. The algorithm obviously assigns the action with the highest Q-value to the agent. The Q-values reflect the long-term reward that the agent can hope to obtain by carrying a certain action *a* in a certain state *s*.

Since Q-learning works by attributing Q-values to all the possible actions that can be taken in a state, the action space cannot be continuous. For this reason, in this project, the action space is constituted of 4 discrete actions. The agent can move towards 4 different directions: North, South, East and West.

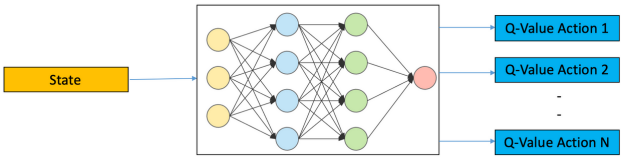

Figure 2. Deep Q-learning illustration [4]

In Deep Q-learning, A Convolutional Neural Network replaces the Q-table to predict the Q-values for each action. The network's architecture takes the current state as an input and returns the Q-values for the 4 possible actions. The action-value function is estimated using the Bellman equation [2], where *r* is the reward, γ is the discount factor, *s* is the current state, *a* is the current action, *s'* is the next state and *a'* is the next action.

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \cdot \max_{\text{a'}} Q^*(s',a')\right] \tag{3}$$

1) Experience Replay: To improve the performance of the neural network, the experience replay method [7] can be used. This method was proven to work nicely for Deep Q-learning [14]. In this method, each transition tuple is stored in a replay buffer. The memory stores the state, the reward, the action, the next state and whether the action has completed the episode or not. When the buffer is full, the oldest transitions get replaced by the new ones. At each training step, a batch of random experiences is sampled from the replay buffer to train the network. This method has for advantage that the samples used to train the batches are identically distributed and independant from each other. It helps break correlations and prevents action values from oscillating or diverging too much.

2) Exploration vs Exploitation: In the case of Deep Q-learning, an ϵ-greedy algorithm [11] was chosen for its simplicity. ϵ is the proportion of random actions to actions that are informed by the existing 'knowledge' that the agent accumulates during the episode. Before playing the game, the agent has no experience so it is common to set it to high values and then gradually decrease it. In the course of this project, the initial ϵ value is set to 1 (pure exploration), and decreases exponentially towards a value close to 0 (pure exploitation) throughout the training. The exploration of the action space helps to avoid the agent getting stuck in a local optima.

C. Curriculum Learning

Curriculum learning [5] aims at optimizing the order in which the agent gains experience, in such a way that the speed of learning or the global performances are improved. Thanks to the knowledge that is acquired on simple tasks, the ratio between exploration and exploitation can be reduced when training on more complex tasks. In the case of this project, a simple task might be for a single agent to eat pellets. This pre-trained model could then be reused when training four agents to add the competitiveness to a model that can already eat pellets.

IV. GAME AND STATE REPRESENTATION

Agar.io is an online multiplayer game in which the player controls a circular cell in a 2-dimensional square grid. The goal is to grow as big as possible by eating pellets or smaller players while avoiding to get eaten by bigger cells. The game never ends and when a player joins a game, he spawns as a single cell at a random location with a determined initial mass. If the player's cell gets eaten, he loses the game and is offered the possibility to respawn with the initial mass.

All the cells that have a mass greater than the initial mass lose a small percentage of it at every time step, so that the larger cells get punished in case of inaction. The position of the mouse cursor in the game provides the direction towards which the cell moves. Two actions can also be taken. If the player chooses to 'split', every one of his cells divide themselves in two cells, each one sharing half of their parent's mass. This action increases the player's speed during a short amount of time. The player also has the possibility to 'eject' a small blob made of his own mass to be eaten by other players.

The more a player grows in mass, the slower he moves. Also, the player's camera offers a wider view as he grows. There is a mass threshold beyond which eating pellets is no longer sufficient to grow. The big players must then try strategies such as splitting up to eat the cell of an opponent to eat him, but this would make him vulnerable to the bigger entities.

In the context of this project, the game was simplified so that the agent could only move in the 4 cardinal directions. The 'split' and 'eject' actions were disabled. The size of the grid was reduced, depending of the number of agents, to increase the probability that two agents run into each other.

The agent's objective is to maximise its total mass to grow as big as possible in a short amount of time, which leads to the first reward system that was used for the training. If there is a change in mass between two steps, the reward will have the value of the difference between the new and the previous mass. This can be rewarding or punitive. If the agent eats pellets or a smaller opponent, the reward will be the pellets' mass or the eaten opponent's mass. On the contrary, if nothing is eaten by the agent, or gets the agent gets eaten, the rewards will be negative and equivalent to its loss of mass. The second reward system is more classical and described in section V.

The state was also defined in 2 ways. The first state is a RGB camera view of the agent and its surroundings, as shown on figure 3. The second state is the same but in grayscale. The states have a 84 by 84 resolution since Nil Stolt Anso et al. [9] showed that a higher resolution increases the performances of the DQN algorithm.

The aim of this project is to study and find the limits of Deep Q-learning in the case of a multi-agent system.

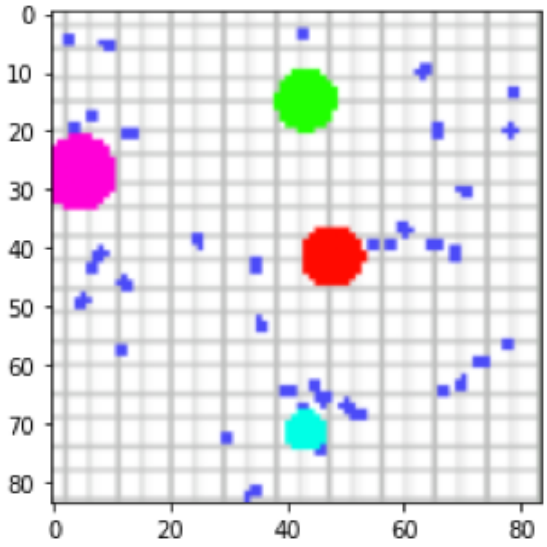

Figure 3. RGB input of the neural network. The pellets are the navy blue pixels.

V. EXPERIMENTAL SETUP

This project uses a Unity environment from the ML-AgarISIA [1] repository. A low-level Python API [13] is used to interact with the Unity environment.

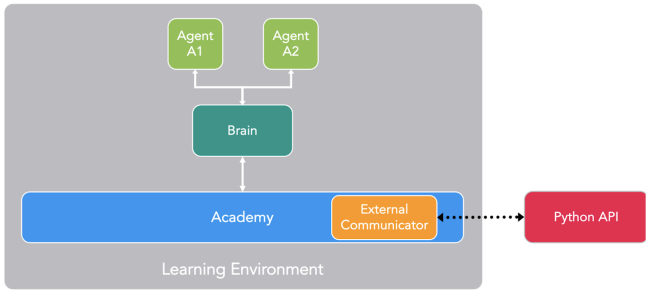

Figure 4. Python API illustration [13]. The academy is the element which orchestrates agents and their decision-making process. It makes sure the agents collect observations, select an action according to our policy, take the action and reset the environment if asked.

First, the Deep Q-learning algorithm was tested for a single agent in a 350x350 pixels grid, containing a maximum of 150 pellets. These trainings were meant to determine the influence of some hyperparameters and optimise their values, while performing fast trainings. The agent was meant to learn to eat pellets. Therefore, the agent gained 1 point each time it ate a pellet, and -0.01 if it didn't eat anything. The small penalty was meant to encourage the agent to move around and explore the grid. A lack of penalty for inaction may lead the agent to a strategy in which it doesn't move at all, as it understands that it will not be detrimental. The player's mass was reset at the beginning of each episode.

The DQN algorithm was then tested for 4 agents competing in the same grid. In this case, the mass of each player was redefined to a random value at the beginning of each episode, to better teach small agents to run away from big ones, and vice versa. Several configurations were tested. First, a reward system related to the variation in mass was tested. Then, a second reward system was imagined, which goes as follows:

1) +1 if the agent eats an opponent;
2) +0.4 if the agent eats a pellet;
3) -0.5 if the agent is eaten by an opponent;
4) -0.01 else.

This reward system was designed in a way that the penalty for dying was not too hard to encourage the agents to take risks to eat pellets. This system was tested with a RGB state and a grayscale state.

The best trained model will then be tested in an environment where 2 agents will compete against 2 bots to see if there is a difference of levels.

VI. EXPERIMENTAL RESULTS

Regarding the model in which a single agent learns to eat pellets, the trend observed on the score graph 5 shows an

[1]The Github link can be found at https://github.com/bastienvanderplaetse/ML-AgarISIA

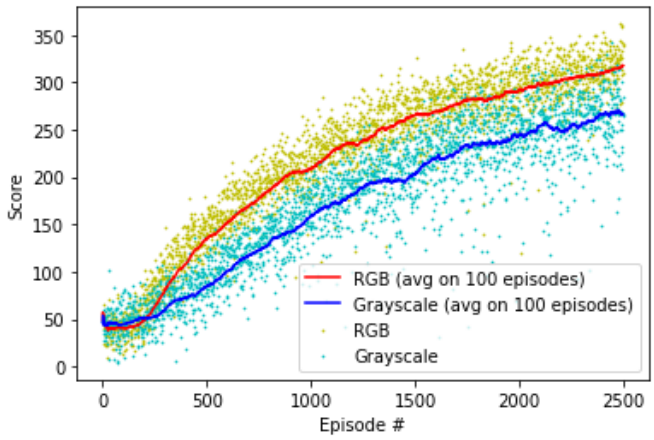

Figure 5. Evolution of the scores of a single agent trained to eat pellets

increase in the agent's scores whether the state is in RGB or grayscale. The agent has correctly learned to eat pellets. The model that uses the RGB state seems to learn faster than with the grayscale state, although both models offer great and comparable behaviors when the trained agents are observed in the grid. It is also important to note that both models have not converged yet after 2500 episodes, and might reach the same scores if given enough time.

It is also interesting to note that both models, when applied in an environment of 4 agents, present different behaviors. With the RGB model, the red agent eats pellets properly, but the three other agents, of different colors, just stick to the wall. This means that the model learned that the *red* agent must eat the *blue* pellets, and therefore has no notion of the other colors and panicks when it sees an unknown one. A random variation of the agent's color at the beginning of each episode can help the model to release the focus on the colors.

On the other hand, the grayscale model makes the agents eat pellets regardless of their colour. This means that it learned that the *big* agent must eat the *small* pellets.

Regardless the model, they only learned to eat pellets and therefore don't know how to behave in a multi-agent environment. They therefore run into each other quite often.

Surprisingly, the reward system based on the difference of masses between two states doesn't perform well. Trained agents spend their time sticking to the wall. This avoidance strategy is preferred to the strategy of growth because the penalty of being eaten is too high. The graph 6 shows a huge decrease in the global scores as exploitation takes the lead over exploration. However, a slight increase in the scores can be observed towards the last episodes. This is mostly due to the fact that the agents learn to stick to the wall, and run into an opponent if they see one. Therefore one opponent will gain mass, and the loser will respawn at a random place in the grid. From that place it will rush back towards the wall and eat the

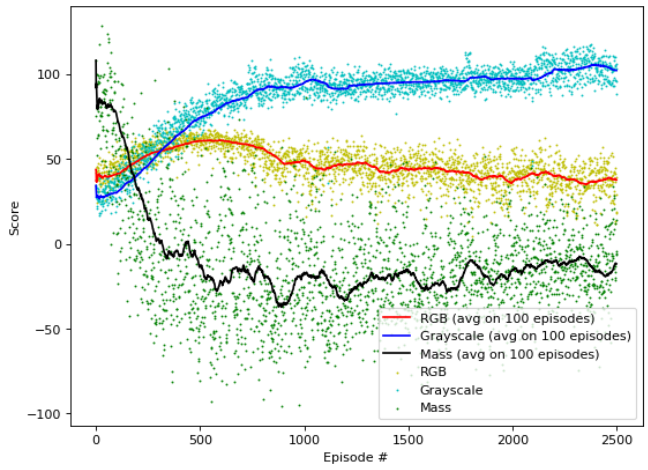

Figure 6. Evolution of the average score of 4 agents with different states and reward systems

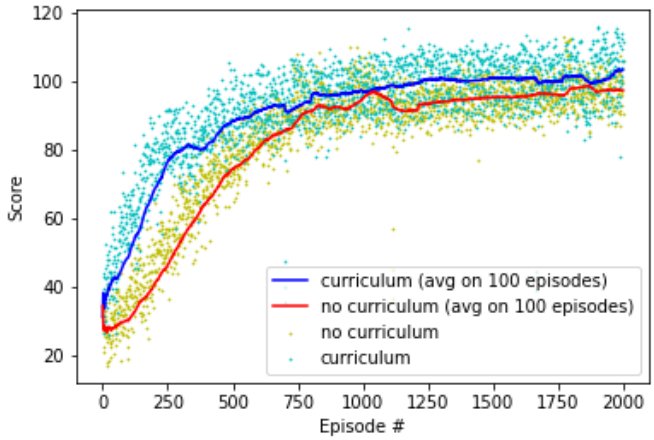

Figure 7. Evolution of the scores with and without a pre-trained model able to eat pellets

few pellets that stand in its way, involuntarily.

The custom reward system shows an increase in the global agents scores when the state is set in grayscale values. A plateau is reached fairly quickly, although scores continue to rise. However, this increase is small and it is therefore not relevant to continue training for too long. When the trained agents are let free in the grid, they show competitive behaviors and chases can be observed from time to time. The RGB-value state oversees an increase in the scores but then the model begins overfitting over a half escape, half competitive policy. This might be due to the fact that there are too many colors for the neural network to work with. The large number of parameters the network is required to have might brings issues with the amount of training time before convergence.

The model that learned to eat pellets was reused as the basis for a training session when 4 players are on the field. This pre-trained model was compared to another model that started from scratch on figure 7.

The use of a form of curriculum learning provides a model with the same performance as the basic one, but those performance are achieved almost twice as fast, as illustrated on figure 7. This result is logical given that the pre-trained model only had to learn how to interact with its opponents, whereas the model that started from scratch still had everything to learn about its environment.

As the pre-trained model already knew how to eat pellets, the starting value of ϵ was reduced to 0.5 in the training, to allow more exploitation to be done in this new environment. As a comparison, ϵ is set to 1 for the trainings that start from scratch.

The performance of the model was also tested in an environment where two intelligent post-training agents cohabit with two bots that perform random actions. The number of times each agent was eaten and reversely the number of times each agent ate an opponent were counted and cumulated on a period of 150 episodes and plotted over time on figures 8 and 9. The agents clearly surpass the bots on both points of view. Table I compares the scores of the agents with the random bots. The agents represent in 83.49% of the cases when a cell ate an opponent. At the same time, they were only eaten 6.34% of the total number of times a cell was eaten. The agents eat around 5 times more opponents than the bots and get eaten a bit less than 15 times less.

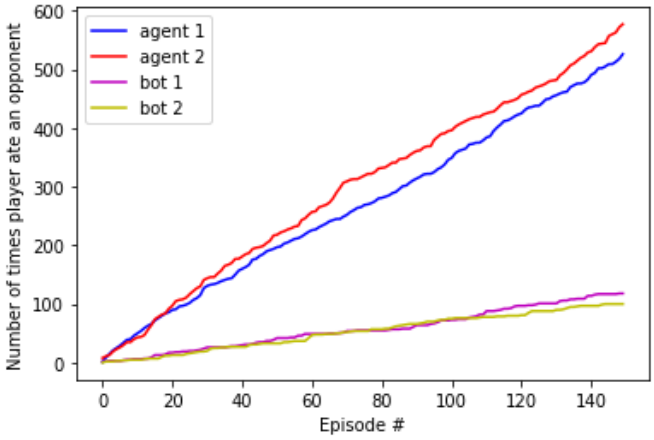

Figure 8. Number of times each agent and bot ate an opponent

As far as face-to-face interaction between agents and bots is concerned, the result is clear. The agents play much better than the bots. The model did learn how to interact in the Agar.io environment. The lack of aggressiveness between two smart agents is due to the fact that the small cells properly learned to run away from bigger entities. The addition of the 'split' action could remedy this lack of violence by allowing larger entities to trap smaller ones. Since the bots move in random directions, they obviously run into each other from time to time, which explains why they get to eat around 100 times an opponent on a period of 150 episodes, even though

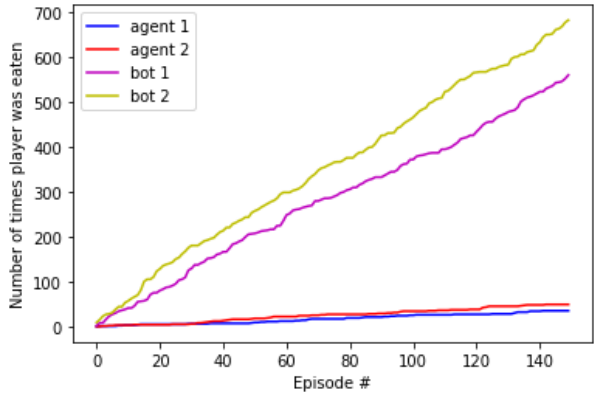

Figure 9. Number of times each agent and bot was eaten by an opponent

they get eaten mostly by the smart agents.

Table I
POST-TRAINING PERFORMANCE COMPARISON BETWEEN TRAINED MODEL AND BOTS THAT MOVE RANDOMLY AFTER 150 EPISODES

Number of times	**agent 1**	**agent 2**	**bot 1**	**bot 2**
was eaten	35	49	559	681
ate an opponent	526	577	118	100

An environment with only one agent can be considered stationary. If the agent is in a state *s* and performs an action *a*, it will always reach the new state *s'*. However, if two players are in the same environment, there is not the same stationarity. If an agent performs an action *a*, it will not necessarily end up in the state *s'*. It will also depend on the action performed by the second agent. The environment therefore varies according to the decisions made by the two agents. This notion of stationarity makes training more difficult when more agents are added to the environment.

It is important to note that the trainings were performed using a GTX 1050. With this processor, a training of 2,500,000 steps took just under 24 hours when 4 agents were in the environment. The limited computing power was therefore a constraint for these trainings.

VII. CONCLUSION

Deep Q-learning has proven to be effective in optimising the behaviour of agents from pixel values only as input. The aim of this project was to demonstrate the effectiveness of this algorithm in a competitive multi-agent context within the Agar.io environment. Different reward systems were tested and compared, along with two different state representations.

The results show that a reward system linked to the variation of the player's mass was ineffective because the penalties they face are too high. Also, the state in RGB-values, although more faithful to that seen by a human player, doesn't necessarily provide better performance than the state in grayscale values, which in addition consumes less processor memory as there are less parameters. The curriculum learning method allows to reach the maximum performance level much faster than a model that starts from scratch.

In future work, this project could be taken further by adding, for example, 'split' and 'eject' actions, and moving to a continuous environment to get the full functionality of the real game. Virus cells, which split the agents that eat them into a multitude of small entities, could also be included in the environment. In the case of a continuous environment, other algorithms should be used, such as the Proximal Policy Optimisation (PPO) or the Soft Actor Critic (SAC) algorithms.

ACKNOWLEDGMENTS

I would like to sincerely thank Luca La Fisca and Bastien Vanderplaetse for their supervision and precious advice all along this project, without whom it could not have been completed in such good conditions.

REFERENCES

[1] Christian Baukhage. *Lecture on game AI*. URL: https://www.researchgate.net/project/lectures-on-game-AI/update/5b31ebe5b53d2f8928979923.

[2] Richard Ernest Bellman. "The Theory of Dynamic Programming". 1954.

[3] Watkins C. J. C. H. "Learning from Delayed Rewards". PhD thesis. King's College, Cambridge, 1989.

[4] Ankit Choudhary. *A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python*. URL: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.

[5] Guy Hacohen and Daphna Weinshall. *On The Power of Curriculum Learning in Training Deep Networks*. School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel. URL: https://arxiv.org/abs/1904.03626.

[6] Guillaume Lample and Devendra Singh Chaplot. *Playing FPS Games with Deep Reinforcement Learning*. School of Computer Science, Carnegie Mellon University. URL: https://arxiv.org/abs/1609.05521.

[7] Lin LJ. *Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach Learn 8, 293–321*. 1992. URL: https://doi.org/10.1007/BF00992699.

[8] V. Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* (518 2015), pp. 529–533. URL: https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf.

[9] Stolt Anso Nil et al. *Deep Reinforcement Learning for Pellet Eating in Agar.io*. Bernoulli Institute, Department of Artificial Intelligence, University of Groningen and ITLearns.Online, Utrecht, Netherlands, 2019. URL: https://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/RL_AGARIO.pdf.

[10] Shanika Perera. *An introduction to Reinforcement Learning*. URL: https://towardsdatascience.com/an-introduction-to-reinforcement-learning-1e7825c60bbe.

[11] Sutton R. and Barto A. G. "Reinforcement Learning: An Introduction." In: *MIT Press* (2017).

[12] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* (529(7587)), pp. 484–489.

[13] Unity-Technologies. *Unity ML-Agents Toolkit*. 2021. URL: https://github.com/Unity-Technologies/ml-agents.

[14] Mnih V. et al. "Playing Atari with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1312.5602* (2013). URL: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf.

APPENDIX

Hyperparameter	Value
Number of episodes	2500
Number of steps	1000
ϵ start	1
ϵ end	0.1
ϵ decay	0.999
Buffer size	30,000 (1 player) / 15,000 (4 players)
Batch size	64
γ	0.99
τ	10^{-3}
Learning rate	$2.5 \cdot 10^{-4}$
Update every	4
Optimizer	RMSprop