Agents in a Multiplayer Snake Environment

Anonymous Authors

***Abstract*—Snake is a very common video game with many variations. First it has been popularized as a singleplayer game on Nokia mobile phones. More recently it has regained attention with the release of *slither.io*, a multiplayer browser game largely inspired by the original snake game. Each player controls a snake and aims to grow in size by consuming candies. Different types of strategies can be used to this end. The objective of this project is to design an artificial intelligence in an environment which reproduces the main characteristics of *slither.io*.**

I. Introduction

The browser game *slither.io* allows various types of strategies while having few parameters. Due to this wide range of strategies this game attracts the attention of the artificial intelligence community. Recently in its *Requests for Research 2.0*, *OpenAI* released a list of seven unsolved problems, including the design of an artificial intelligence for a clone of *slither.io*.

In this project we design an environment which looks like *slither.io*, while simplyfing some features to ease the training and the design of agents.

Regarding the environment a graphical interface is available. We reuse the interface of the project [1]. However we completely rebuild the backend to make it more efficient.

Then we design two types of artificial intelligence. On is based on reinforcement learning. The other one consists in tree search methods. Our cods is available here [2].

II. Background and Related Work

A. Reinforcement Learning

1) Learning: **Q-learning algorithm** [1] [2] [5] [3]

$$Q^{t+1}(s_t, a_t) = (1-\alpha)Q^t(s_t, a_t) + \alpha(r_t + \gamma \max_b Q^t(s_{t+1}, b))$$

SARSA algorithm [1] [2] [5] [4]

$$Q^{t+1}(s_t, a_t) = (1-\alpha)Q^t(s_t, a_t) + \alpha(r_t + \gamma Q^t(s_{t+1}, a_{t+1}))$$

2) Exploitation vs. Exploration: **ϵ-greedy exploration** [1] [2] [5] With probability ϵ, the agent chooses a random action to explore and learn the consequences.

Selection with softmax operator [5]

$$P(a_t = a) = \frac{\exp\left(Q^t(s_t, a)/\tau\right)}{\sum_b \exp\left(Q^t(s_t, b)\right)/\tau}$$

Elaborate (in your own words) the background material required to understand your work. It should cover a subset of the topics touched upon in the course. You are encouraged to cite topics in lectures, e.g., structured output prediction in , book chapters, e.g., Chapter 9 from , or articles from the literature, e.g., . Basically, you should prepare the reader to understand what you are about to present in the following sections. Eq. (1) shows a random equation.

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y} \in \{0,1\}} p(\mathbf{y}|\mathbf{x}) \tag{1}$$

III. The Environment

We designed a simplified *slither.io* environment while trying to keep the main characteristics of this environment. Several snakes move on a 2D grid. They grow each time they eat a candy, these candies randomly appear on the grid. When the head of a snake collides with a border or another snake, it dies and is tranformed into candies. The snakes aims at maximizing their score at the end of the party, which is equal to their relative size to the total size of all snakes alive.

A snake can move along 4 directions, but the head of the snake can not move backward. Identical to *slither.io* snakes can cross their own tail.

In *slither.io* when your snake is long enough, it is more efficient to kill snakes in order to collect highly concentrated candies than to collect randomly spawning candies. In contrast at the beginning of the game, the snake has a short range of vision, hence it is very vulnerable. And therefore the player is push to change its strategy while its snake is growing. The potential for the emergence of a complex and adaptative behaviour makes this environment really interesting.

IV. The Agent

We designed two different agents for this environment to compare different strategies. The *Reinforcement Learning* (*RL*) agent is model-free, whereas the *Minimax* agent is based on the exploration of the different possibilities.

A. The Reinforcement Learning Agent

This model-free agent uses either SARSA or Q-Learning to learn the model and the consequences of its actions. It also uses either ϵ-greedy exploration or the Softmax method to tackle the dilemma between exploitation and exploration.

To perceive its environment without having too many states to learn, we use 12 inputs. When the input is bounded, if the real value is greater than this or doesn't exist, it is assigned the greatest value.

1) 0 if the head of the snake touches the right wall, 1 otherwise.
2) 0 if the head of the snake touches the upper wall, 1 otherwise.
3) 0 if the head of the snake touches the left wall, 1 otherwise.

[1] http//github.com/bhairavmehta95/slitherin-gym

[2] Our code is available here: http://anonymouslinktoyourcode.zip

4) 0 if the head of the snake touches the lower wall, 1 otherwise.
5) The distance between the square located to the right of the head and the closest candy (between 0 and 7).
6) The distance between the square located to the top of the head and the closest candy (between 0 and 7).
7) The distance between the square located to the left of the head and the closest candy (between 0 and 7).
8) The distance between the square located to the bottom of the head and the closest candy (between 0 and 7).
9) The distance between the square located to the right of the head and the closest square of another snake (between 0 and 3).
10) The distance between the square located to the top of the head and the closest square of another snake (between 0 and 3).
11) The distance between the square located to the left of the head and the closest square of another snake (between 0 and 3).
12) The distance between the square located to the bottom of the head and the closest square of another snake (between 0 and 3).

These inputs allows the snake to have a local vision of the candies, a local but smaller vision of the other snakes, and feel when it touches a wall.

We also give rewards after each action done by the snake.

- 1 if the snake ate a candy.
- -10 if the snake died.
- 0 otherwise.

The agent will easily learn to look for candies and to avoid walls. It will also learn to avoid other snakes.

Finally, we use the following parameters:

- $\epsilon = 0.1$ in the case of ϵ-greedy exploration.
- $\tau = 0.1$ in the case of Softmax exploration.
- $\alpha = 0.2$. A learning rate so high is strong enough because the inputs and the best actions to take are highly correlated.
- $\gamma = 0.9$. We don't use 1 to make it clear that the best path is always the shortest, but it is near 1 since we care about the future almost as much as the present.

B. *The Minimax Agent*

V. Results and Discussion

To measure how well the two agents perform in our environment, we use several measurements. We introduce the *Random* agent, which chooses at every step a random direction. We simulate the games on a grid of size 30, with 10 candies on the map and the two adversarial agents we want to compare.

A. *Performance of your Agent in your Environment*

1) Performance of Learning: To analyze the learning time of the *RL* agent, we simulate it against the *Random* agent. At the beginning, the *RL* agent knows nothing. We run 5000 games, and we compute the frequency of victories for the *RL* agent for every chunk of 100 games. The results are presented on Figure 1. As we can see, with only a few hundreds games, this agent is far better than the *Random* one.

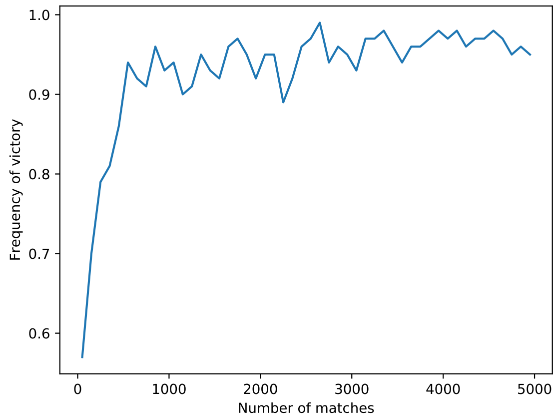

Fig. 1. The learning curve of the RL agent against a random one.

2) Performance of the agents: We simulated 1000 games between each pair of agents. The results are presented in Table I. When the sum of the scores isn't equal to 1000, it means that games ended because both agents died at the same time.

As can be seen, the *Minimax* agent is nearly unbeatable, no agent scored a point against it. However, it scored 933 against the *Random* agent, which means that it was killed 77 (while killing the other one). Even though the *RL* dies 4 times out of 1000 without killing the *Random* agent, it performs better with respect to its own score. In 991 out of 1000 games, it manages to stay alive while killing its opponent, which means it learned from experience to avoid going too close to its opponent, whereas the $emphMinimax$ agent thinks its opponent will take the best action. This hypothesis is clearly false for the *Random* agent.

TABLE I
Results of the games.

Game	Random	Minimax
RL	991 - 4	0 - 174
Minimax	933 - 0	

The game between the *RL* and *Minimax* agents ends before 1000 iterations 174 times out of 1000. For 50 of theses games, it ends in a tie, where both agents are killed at the same iteration. The other 776 games are interesting to look at. When a game reaches the maximum number of iterations, we stop it, and measure the sizes of the two agents. The distribution of the proportion of the size of the *RL* agent compared to the sum of both sizes is presented on Figure 2. The main difference between the *RL* agent and the *Minimax* one is the local vision of the former, which can only "see" candies and snakes up to a Manhattan distance of 8 and 4 squares. The average propotion size of 25 % for the *RL* agent is thus really strong.

B. *Performance of your Agent in the ALife Environment*

We only tried to deploy our *RL* agent in the ALife Environment since it is model-free. Deploying the *Minimax* agent would have require to create a totally new model, which would

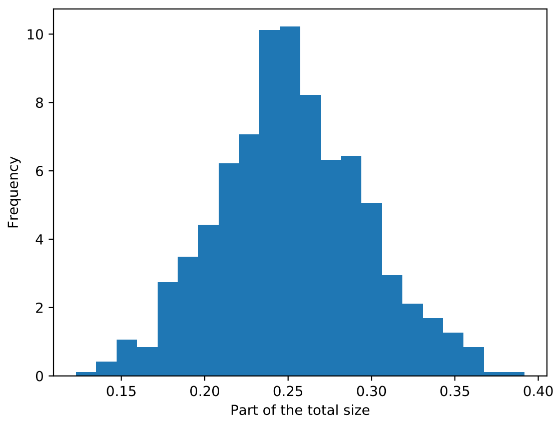

Fig. 2. Distribution of the proportion of the size of the *RL* agent for the 776 games ended after 1000 iterations.

have made it a different agent. When using the same technics (Q-Learning or SARSA, ϵ-greedy or Softmax exploration), the results aren't as satisfactory as in the Snake world. This environment is much more complicated, and the inputs and outputs are analogous.

We chose to tackle these issues by discretizing the inputs. We don't use the energy input. We multiply each other input (the color sensors) by 5, and round them to the lower integer. To keep the number of actions small, we only allow rotations of $-\frac{\pi}{2}$, 0, $\frac{\pi}{2}$ and pi.

After a few minutes of simulation, it seems that the herbivore have understood that they have to eat the plants. Unfortunately, this timeframe doesn't seem sufficient enough for the insects to learn other lessons.

VI. Conclusion and Future Work

This environment is interesting to use for multiple agents, because it has a few aspects. A snake has to eat candies to grow, but in the same time it also has to avoid collisions with walls and other snakes. The inputs defined in Subsection [**?**] allows an agent to have a local but precise vision of its environment. Such an agent learns quickly and is able to outperform a *Random* agent after a few hundreds games. When competing against a strong AI, the *Minimax* agent, whose only limitation is the depth it can explore when deciding which action to take, its results are quite satisfactory.

The main issue with the strategy we chose is the high number of states. For instance, the agent has to learn many times that going into a wall is bad, because there are a lot of different states where it is touching a wall. To continue this work, one should focus on Deep Reinforcement Learning, to give the snake a bigger vision of its environment, and make it easier and quicker for it to learn the best strategy.

References

[1] N. Tziortziotis. Lecture IV - Introduction to Reinforcement Learning. *INF581 Advanced Topics in Artificial Intelligence*, 2018.
[2] N. Tziortziotis. Lecture V, part II - Approximate and Bayesian Reinforcement Learning. *INF581 Advanced Topics in Artificial Intelligence*, 2018.
[3] Watkins, Christopher & Dayan, Peter. (1992). Technical Note: Q-Learning. Machine Learning. 8. 279-292. 10.1007/BF00992698.
[4] A. Rummery, G & Niranjan, Mahesan. (1994). On-Line Q-Learning Using Connectionist Systems. Technical Report CUED/F-INFENG/TR 166.
[5] Sutton, R. S. & Barto, A. G. 1998 Reinforcement learning: an introduction. Cambridge, MA: MIT Press.