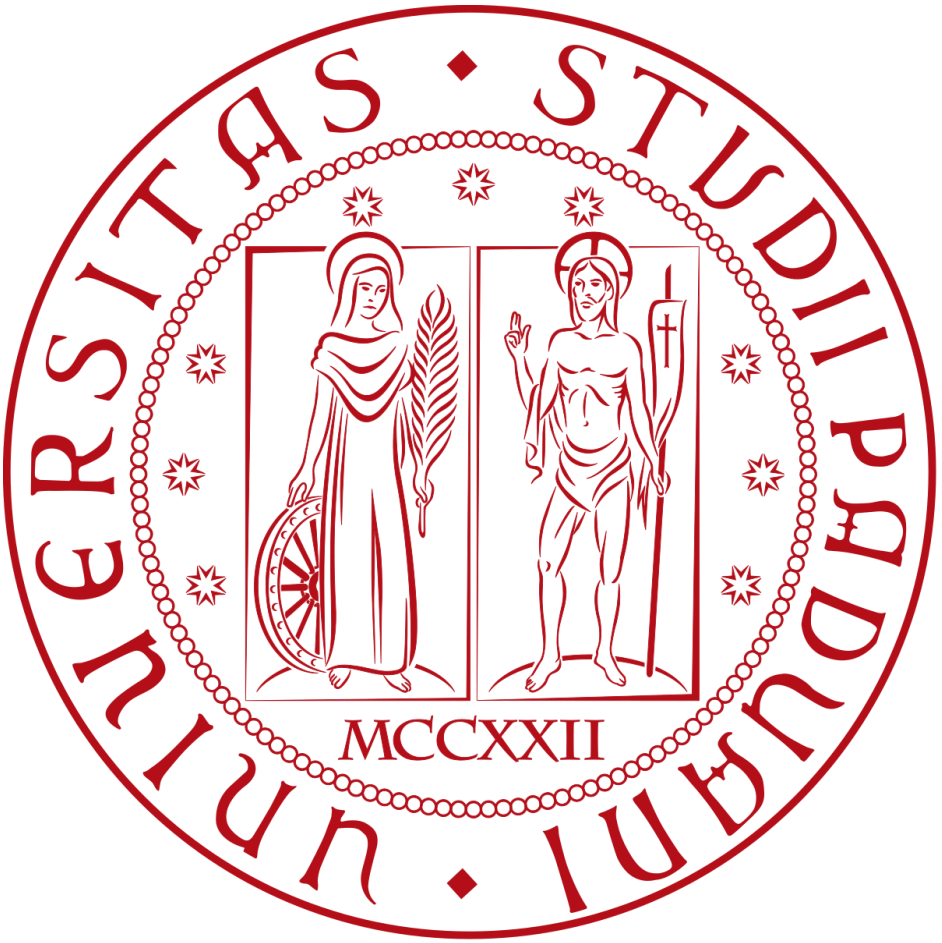
REINFORCEMENT LEARNING PROJECT

Snake Game



Pietro Girotto 2088245

Pd, 26/08/2024

INDEX

The Project

Introduction

The problem presented wants to create a Reinforcement Learning Agent capable of playing on a square board that closely resembles the standard rules of Snake with a few modifications:

1. Colliding with a wall will just give a negative reward, without actually resetting the board.
2. If the snake eats himself all pieces from the eaten-block to the tail are erased, a negative reward is given and the game continues.
3. Filling up the board gives a strong positive reward and resets the board from scratch.
4. It is a legal move to stay in the current position for the snake.

The goal metrics specified in the reports were the following:

1. “maximize the amount of "fruits" over a time-frame”
2. “limiting the number of times it hits the walls”

Brief presentation of the work done

The main idea was to compare the following agents on the problem:

- ➔ A random action policy as a reference
- ➔ A rule-based agent crafted from scratch
- ➔ A deep Q-learning network for comparison
- ➔ A MoE approach for an hybrid agent that based on specific features uses either the rule-based or the Deep Q-learning agent

I started with the baseline policy and the DQ network. It was clear that the early game of the baseline was quite good whereas the DQN was handling way better in more advanced scenarios. As such I created the hybrid agent combining the rule-based algorithm in the early game with DQN for the late game exceeding previous results.
By inspecting the rewards over time I could also analyze why the DQN strategy was better than the rule-based one and interestingly enough it narrows down to the fact that making the snake eat itself will clear a great portion of the board in late game allowing for faster collection of fruits and overall better cumulative rewards.

All code for training/plotting/evaluation can be found in the github repo. You can either see the already computed results in the **main.ipynb** or compute them by yourself by either running the notebook or the **training.py** and **evaluation.py** files.

The Agents

Random Agent

As straightforward as it can get, the random agent (RND) chooses a random action from the pool UP, RIGHT, DOWN, LEFT, NONE where NONE means the snake does not move.
Performances of this agent were, as expected, poor and it served solely as an insightful benchmark for the other agents.

Baseline Agent

The baseline agent (BASE) is the rule-based heuristic. It follows a sequence of steps to find the final answer:

1. Compute all new possible positions (4 total, one per direction).
2. Filter out the illegal ones (hitting a wall or eating itself).
3. Out of the remaining positions, find the one that is the closest to the fruit and move to it.
4. If no position is legal, move to a position that makes the snake eat itself. If more than one such position is available, choose one at random.

The idea is to move the snake closer to the fruit as possible and, when it is not possible to move further, just make it eat itself to allow for new fruits to spawn and the board to clear.
This strategy worked quite well in the early game, but failed to scale when the body length started to be noticeable as more planning was needed.

Deep Q-Learning Agent

The deep q-learning approach (DQL) was chosen because in literature it was proved effective with simple games like the Atari and the environment representation was easily translated to a discrete input-space suitable for neural networks.
I thought that it was a great showcase of how ML can deduct sensible complex policies in the late game whereas the BASE agent failed.

Hybrid Agent

During evaluation, it was quite clear that the baseline was faster at getting through the early game, achieving better rewards overall. As such an hybrid agent (HYB) was used that operated on the following structure:

$$\pi(s) = \begin{cases} \text{BASE agent}(s) & \text{if } \text{len}(s, \text{body}) \leq 5, \\ \text{DQL agent}(s) & \text{otherwise.} \end{cases}$$

DQL moves are further checked against possible wall hits. If so, the BASE agent is used instead.
Where $\text{len}(s, \text{body}) \leq 5$ means that the overall length of the snake (head included) is less or equal than 5.
Training of the DQL sub-agent was done exclusively on the instances in which the body length was big enough to further solidify the network's knowledge.

Results

The training/evaluation was done on a 1000 board x 1000 moves set (totally 10E6 samples).

Fig. 1: Training avg. reward per iteration

Fig. 2: Training cumulative rewards per iteration

It is easy to see from the training plots that the final agent (HYB) takes the best from the BASE and the DQL agents. In Fig.1 it is clear how the evolution of the bots during training is a mixture of the good early start form the BASE agent and the strong late rewards for the DQL agent.

Further analyzing the final trained agents allows us to see a clear picture of the final evaluation. The evaluation was done on 1000 boards for 1000 iterations.

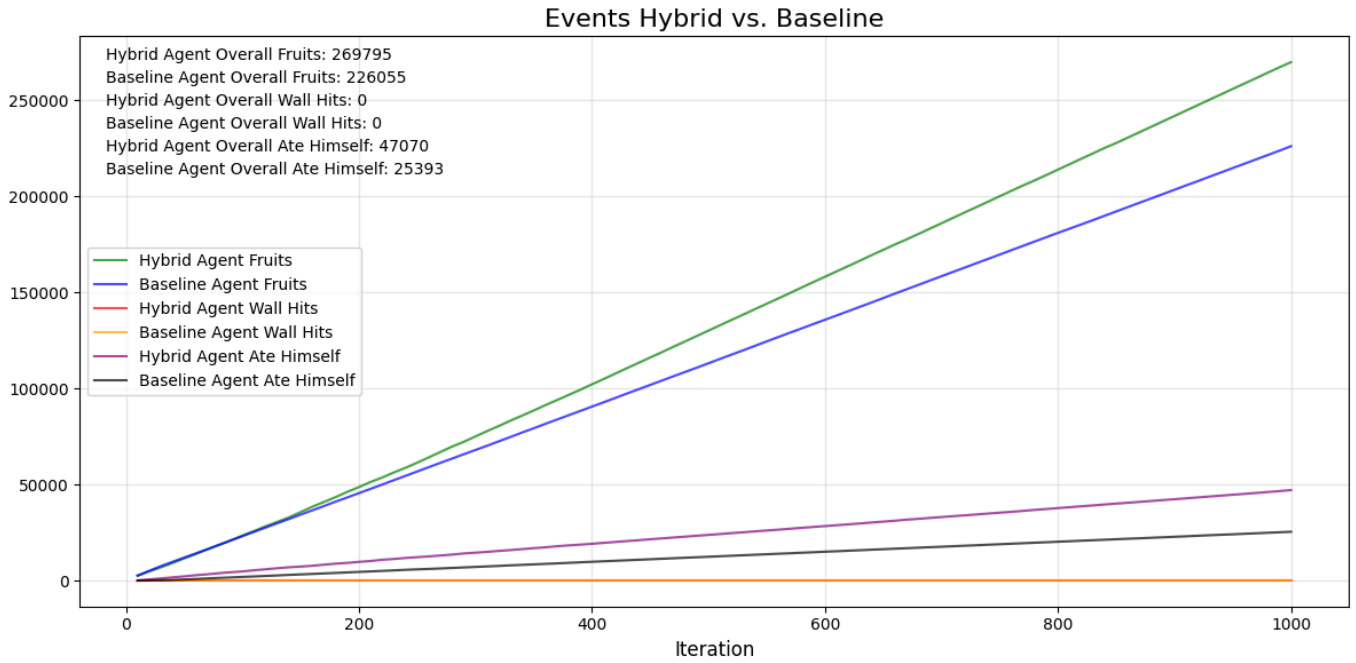


Fig. 3: Wall hits/Fruits/Body eating in the trained final models

Given the structure of the BASE and HYB agents, clearly no wall hit was possible. As such a perfect scoring of 0 was achieved in both.
I think the interesting part comes to the correlation on the number of fruits and the number of times the snake ate itself. The better results achieved by the HYB agents sets him apart with a seemingly linear proportional factor to the number of times the snake “failed” by eating itself.
Given the goals specified were clear, I indeed created a better strategy that might crash the snake into itself to get out of sticky situations but for the sake of eating more fruits in the long run. I find it interesting that the rules specified allowed for such deviation that the models greatly exploited them before I could think of them beforehand.

Overall the HYB agent achieves great results against all agents on all benchmarks and maximizes the specified goals.

Issues

The trained agents excels at the given tasks but is not an overall good “snake player”. This is due to the fact that eating yourself would conventionally dictate the end of the game, on the contrary with this rule it might be an incentive for better results in the long run. Perhaps changing the rewards could lead to more “traditionally acceptable” agents but it would not make sense in the scope of this project, as the goals did not care about whether the snake ate itself or not.

Conclusion

The final HYB agent is a strong competitor for the given task, highlighting how in Reinforcement Learning setting rewards and goals is key to the final solution found. Numerical results were significantly better both during training and evaluation with regard to the other agents.