Artificial Intelligence & Programming
Institut Supérieur des Arts Multimédia Mannouba ISAMM
MSc Data Science & Information Retrieval
Fall Semester 2024
Imen Labidi, Roua Ben Ahmed & Yacine Montacer

Technical Report

1 Introduction

We were assigned the classic snake game where our playable character is a snake on a 2D grid, and we have to turn either left or right to eat the fruit without running into the perimeter boundaries on the (x, y) axis and without running into the snake's body causing the game to be over. As the snake ingests apples, its length continues to increment along with our score. The objective is to ingest as many apples as possible before running into one of the game-over conditions.

When we first started with the source code assigned to us and understood the concepts, we wanted to implement the game by ourselves to test our learning. So, we followed a GeeksForGeeks tutorial initially step by step, as it was our first time using the pygame package. And we chose to continue by adding extra features which are shown in the 3 difficulty levels in our game: Easy, Medium and Hard.

Easy mode has just the normal rules as the initial game we were working with. Medium mode had a mine spawning randomly alongside the apple. We also coded a likely event where the fruit spawn tends to happen near the borders or near the snake's body. We also implemented a Hard mode where up to 5 mines can spawn simultaneously with the apple. Additionally, each difficulty mode has a higher snake speed than the previous one. These conditions make it more challenging to play and to automate the game.

While working on automating the snake, we tried implementing Reinforcement Learning with Tensorflow - specifically Deep-Q Learning. We also use the path-finding algorithm A* with a Manhattan distance heuristic.

2 Game Definition

Our game is a variation of the classic "Snake" game, with additional features and challenges depending on the difficulty level.

2.1 Snake Movement

The snake, which is our agent, moves in a two-dimensional grid within a predefined window. The player controls the direction of the snake, which can turn either left or right, until it runs into its own body, one of window perimeters or a mine. The snake's speed gets faster as the difficulty gets harder.

2.2 Fruit

The snake consumes fruit that appears randomly in the grid. Upon eating fruit, the snake's length increases, and a point is scored. On Medium & Hard difficulties. The Fruit is likely to spawn somewhere near the snake's body or near the borders, making avoiding those end-game conditions more challenging.

2.3 Mines

Mines are introduced as obstacles in the environment. Depending on the difficulty level, there can be from 0 to 10 mines in the grid. Colliding with a mine results into the game ending. Medium difficulty only has one mine, while Hard can have up to 10 appearing simultaneously with the Fruit.

2.4 Gameplay & Environment

The difficulty of the game can be set via the difficulty_level variable in the main-menu. The game is implemented using the Pygame library for rendering graphics, handling user inputs, and maintaining the game loop. The snake's body and the fruit are drawn in a grid, and the game continuously checks for collisions, rewards, and penalties based on the current state of the game.

3 Automation

We chose to automate the snake-game with Reinforcement Learning RL, specifically a Deep Q-Network DQN. In this section we will go over some general definitions along with our specific implementation and metrics.

3.1 Q-Learning

In Q-Learning, the AI learns the optimal actions through interacting with the game environment using a reward system with the function calculate_reward(). Each action taken by the snake (UP, DOWN, LEFT, RIGHT) can result in a positive or negative feedback. Eating an apple results in a positive reward, while hitting a wall, a mine or the snake results in a negative reward. There is also a small negative reward for taking a step to de-incentivize long paths. We would also like to add a reward for getting closer to the apple, which we have not been able to implement yet.

3.2 Deep Q-Network

The AI uses a neural network to approximate the Q-values for each action based on the current game state. The state is represented as a binary vector of 13 conditions indicating the snake's position relative to mines, the Fruit, grid boundaries, and its own body. The Q-Network has two hidden layers with 128 units each and ReLU activation. The outer layer has 4 units corresponding to the four possible directions the snake can go in.

3.3 Path-finding: A*

The AI makes use of the A* algorithm to plan a path towards the fruit. A* computes the optimal path based on the snake's current position and the fruit's position while avoiding obstacles. This is used in conjunction with the Q-Learning model to guide the snake's movement. Of course, since we are dealing with a grid and the only obstacles are the mines and the boundaries, we chose to use Manhattan distance for the A* heurisitc value.

3.4 State

The state includes 13 binary features indicating whether the snake is to the left, right, above or below the fruit; whether the snake is near the four grid boundaries; whether the snake's body occupies adjacent cells and whether there are mines in the snake's proximity.

3.5 Exploitation and Exploration

To balance exploration and exploitation in training, we used an Epsilon-Greedy approach. With probability Epsilon, the AI chooses a random action, and with probability 1 - Epsilon the AI selects the action with the highest Q-value predicted by the neural network. Epsilon decays slowly over time to diminish exploration and promote exploitation as the tests run. We started with Epsilon = 1 and Epsilon-Decay = 0.99. This way Epsilon decays slowly so we can explore more over 10,000 episodes. The reward system incentivizes the AI to eat the fruit and avoid obstacles as Epsilon decays. There is a small penalty with each step to encourage efficiency.

3.6 Training

During each epsiode, the AI model use Q-Learning by transitioning states and calculating rewards with each action taken. The Q-Network uses the Bellman equation to estimate the future reward. The model is training using Mean Squared Error MSE loss. An Adam optimization is used for more efficient training. As of yet, we are still running the model, but we will save it when it finishes running so we can evaluate it and make improvements.

4 Final Thoughts

We only generated up to 5 mines on the AI game, as opposed to up to 10 mines on the game we actually play to try and reduce complexity. This added complexity made it more difficult to follow along with other snake-game automation tutorials. This was a very interesting project to work on for us as a first step into AI applications. We don't believe this solution is efficient or effective enough and there is a lot we want to improve with the incentive structure and other code blocks. And we look forward to learning more about the theoretical aspects of AI which delve a lot into statistics and Computer Science, as well as engineering applications.