Intelligent Agents on Snake game

Designing Intelligent Agent Coursework

AUTHOR
Figo Aranta
20403319
psyfa2@nottingham.ac.uk

University of Nottingham
Computer Science with Artificial Intelligence

Table of Contents

1 Introduction

The snake game was first developed by Tameloi Armanto and introduced in 1997 on a Nokia phone. The game consists of a *n x m* grid, a snake, and food. The game's mechanism is to control the snake to collect food on a map and avoid collision with a wall or its own body. For every food collected, the snake's length will increase by one pixel or one cell in the grid. The goal of the game is to collect as much food until the snake's body fills the entire space and the game ends.

1.1 Objectives

This project aims to develop multiple intelligent agents to play the game of snake and measure the performance of each agent in a different environment, e.g., different width and height grid sizes. The main questions that this project tries to answer: are the agents able to adapt to a different environment? How do different approaches to building the agents compare in terms of performance?

However, few objectives need to be accomplished beforehand to achieve its primary goal and to answer the questions above.

1. Develop the game environment, i.e., the snake game.
2. Develop multiple Deep Q- Learning algorithm agents to play the game.
3. Develop an agent that uses the Hamiltonian algorithm to construct a path in an environment.
4. Develop an agent that uses an informed search technique to collect food in the environment, i.e., A* algorithm.
5. Run several experiments of the agents playing the game, in a different environment and evaluate their performance.

2 Existing Approaches

Many of the studies focused on the approach of deep reinforcement learning. Upon researching the currently available approaches, there was indeed a limited study on this subject. However, this section will discuss a few notable pieces of research based on deep reinforcement and other techniques such as genetic algorithms and various searching algorithms.

In a study in 2021, Sebastianelli et al. applied a Deep Q-learning approach to the snake game [1]. The proposed system consists of a snake, fruit, and a 20 x 20 grid game board. The snake is rewarded with 10 points if it successfully eats the fruit or -10 if it bumps into a wall or its tail, and the state in each game step is an 11 binary values vector. Furthermore, the neural

network model architecture in the Deep Q-learning algorithm has 11 neurons in the input layer, 100 neurons in every three hidden layers, and three neurons in the output layer. The activation function used is rectified linear unit plus dropout and SoftMax for hidden and output layers. Overall, they achieved the best model with a mean score of 56.

Białas, however, used a different approach for this problem; instead of using a reinforcement learning algorithm, he employed a genetic algorithm to train the Neural Network [2]. Białas referred the Neural Network as the DNA in this project and that this DNA is the brain for decision making and the most crucial part of the snake. Białas conducted two architectures with different parameters for the project. The difference between the two is in the hidden nodes and population size. The first has six hidden nodes and a 2000 population size, while the second has 8 and 1000. Though it is not stated what the size of the environment is, the second model has better performance with an average score of around ~35.

Appaji used multiple searching algorithms to play the snake game and compare their performances with a human agent in his thesis project [3]. The search algorithms include Breadth-First Search, Depth First Search, Best First Search, A* Search, and Hamiltonian Search. After running the experiment, Appaji concluded that the A* Search algorithm outperforms all other search algorithms, including the human agent.

To conclude, various studies on intelligent agents to play the snake game have not shown to achieve a "perfect agent." Even with the help of modern technology of machine learning, the results still come up short. Therefore, this project aims to not only develop an agent to play the game of snake but also try to develop the "perfect" agent that could finish the game without dying.

3 Design & Implementation

This section will discuss how the experiment pipeline, environments, and agents are designed and built. In high level language, it covers how the experiment is conducted to test the agent's performance. The primary programming language used to construct the system is python. Due to the simple nature of python's syntax and its many powerful libraries for machine learning, this project uses the language to ease the construction of the environment and neural network model.

3.1 Environment

As stated previously, this project uses python as its main language; therefore, Pygame – a python module for video games – is utilized for building the environment. The environment consists of a $w \times h$ space, a snake, and food. The following section will discuss further the environment space and the snake. However, it is worth noting that since building the game environment is not the main priority, the environment for this project is adapted from an external source [1] to speed up the process of achieving the main objectives. Thus, this project

does not build the environment from scratch but instead modifies and adapts from an existing one.

3.1.1 Environment Space

The environment space is a rectangle/square space with a specified width and height. Because this project tries to answer how different environment affects each agent's performance, three environments with different width and height will be constructed. The environments consist of (400 x 400) pixels, (320 x 320), and (240 x 240). However, each block in the grid, including the snake's body, has a size of **20 by 20 pixels**; therefore, in a 400x400 environment, the actual environment size for the snake is 20 x 20 blocks.

3.1.2 Snake

The snake is the entity in the environment that the agents control to collect the food. At the beginning of each game, the snake starts off with a length of 3, where it consists of 1 block for the head and 2 blocks for the body. For every food collected, the length of the snake is incremented by 1 until it fills the entire space. When the snake collides with walls or its own body, it dies and the game ends.

3.2 Experiment pipeline

The pipeline experiment pipeline in this project is manually built from scratch. In the system pipeline, an agent will be tested in an environment for **ten** iterations, and for each iteration, the agent's score is stored in a list. At each iteration, agents are expected an action, based on a given state in the environment. After the 10th iteration, the system then moves to the next agent and repeats the process until all agents in the particular environment are evaluated. This process is repeated for every environment; the results are used for analysis, which will be discussed in Section 5. This pipeline can be dynamically scaled for more than the currently presented agents without changing much of the code.

```
for environment in environments:
    for agent in agents:
        while n_game<10:
            state_old = agent.get_state(game)
            action,path = agent.get_action(state_old,True)
            reward, game_over, score = game.play_step(action,path,n_game+1)
            if game_over:
                n_game+=1
                scores.append(game.score)
                game.reset()
```

Listing 1: Simplified python code for conducting experiment for every agent in every environment.

3.3 Agent

This section will discuss the principles/algorithms behind the agents and the state information that the agents receive in more detail. The agent is the system responsible for outputting the next move based on the current game state. This project will experiment on four different agents. The first two agents use a Deep-Q learning algorithm, the third agent uses a Hamiltonian algorithm, and the last agent uses an A* algorithm.

3.3.1 Deep Q-Learning

Deep Q-learning is a deep reinforcement learning that utilises a neural network model. It is an advanced version of reinforcement learning. Reinforcement learning aims to maximise its total reward (Q-value) across an episode. An episode is all that occurs within the environment between the initial and final or terminal states. The agent generates this Q-value by performing a sequence of actions, and the Q-values are stored in a table. However, since the environment in this project has so many possible states. The amount of time and memory required to explore each state and store the Q-value would be unrealistic; therefore, deep q-learning is utilised.

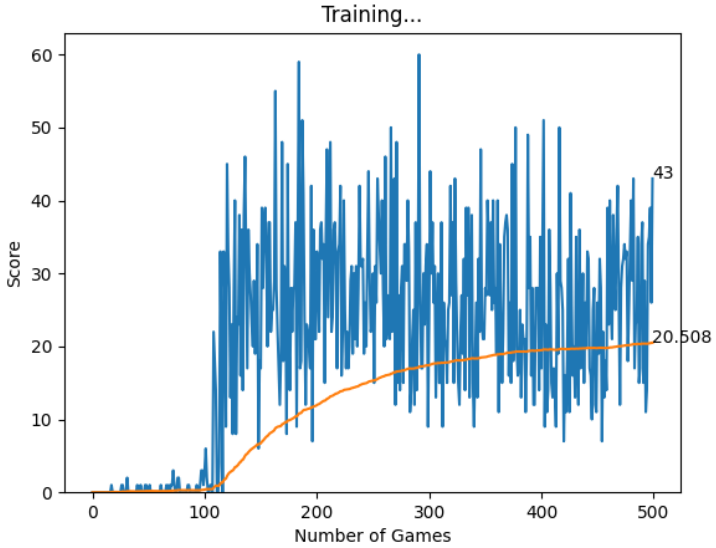
As stated previously, this project uses two Deep Q agents. Although the learning principles are the same, the main difference lies in the neural network model. The model of the first agent has 11, 256, 3 neurons in the input, hidden, and output layer, respectively, whereas the second model has 23, 529, 3. The state information in the first agent is illustrated in Figure 1.

$$
\begin{aligned}
x(k) = [&danger\ straight,\\
&danger\ right,\\
&danger\ left,\\
&moving\ left,\\
&moving\ right,\\
&moving\ up,\\
&moving\ down,\\
&food\ left,\\
&food\ right,\\
&food\ up,\\
&food\ down].
\end{aligned}
$$

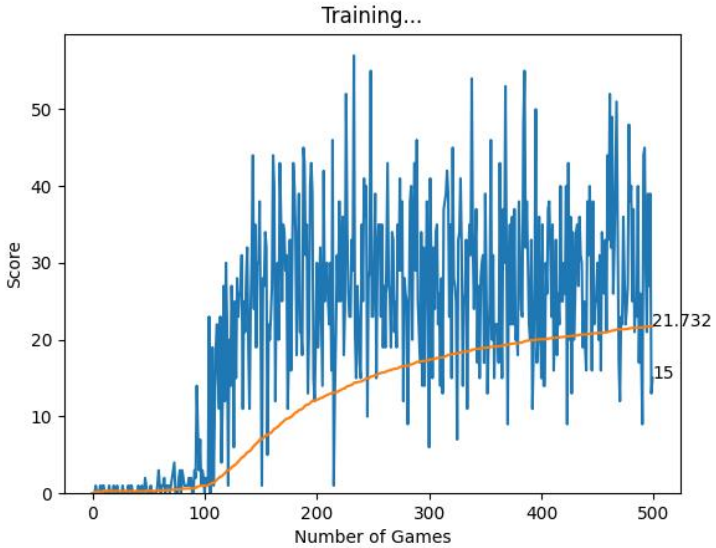
Figure 1. State information in the first Agent.

If the snake took another step forward and collided with either a wall or its own body, "*danger straight*" would be true. The same goes with the direction and food, i.e., if the direction of the snake is moving towards the right side and food is below the snake, then the values for "*direction left*" and "*food down*" would be *1*. The state information for the second agent is

similar; however, instead of only looking at one block ahead in the left, right, and straight direction, it has information of 5 blocks ahead in those directions.



(a). Agent 1 with 11 neurons



(b). Agent 2 with 23 neurons

Figure 2. Deep Q-learning agents after 500 number of games

The agents of Deep Q algorithms were trained in a 400 x 400 environment for 500 games before being evaluated. As can be seen from Figure 2, the performance of both agents increases after about 100 games. The agents take a 1D array at each game step in the training process, which contains the information based on its current state. Next, the input is passed to the neural network will output a prediction of a 1D array, where all elements are 0 except for one element, which will be 1. The index where the value of 1 is, indicates the direction. Based on the action,

the agent is given a reward of +10 if it eats food and -10 if it collides with a wall or its body. Using all the state information, action, and reward, the Deep Q agents update their model weights using a bell-man equation, Eq.1. Moreover, after 500 game iterations, the second agent seems to have a slightly higher average score than the first agent. This may indicate a correlation between better performance and more inputs; however, this will be discussed further in Section 5.

$$V(s) = max_a(R(s,a) + \gamma\, V(s')) \quad (1)$$

Pytorch is used to develop the neural network model in the Deep Q-learning algorithm. The code architecture for constructing a Deep Q-learning agent is adapted and modified from [1].

3.3.2 Hamiltonian

A *Hamiltonian cycle* is a cycle that goes through each node exactly once, and a Hamiltonian algorithm constructs a Hamiltonian cycle in a graph. In this project, the Hamiltonian agent will first construct an adjacency matrix of its environment graph and generate a Hamiltonian path that leads to a cycle. Since generating a Hamiltonian cycle is an NP-Hard problem, the algorithm used to generate is partially hard-coded. Finally, once a cycle has been found, the agent maps each point in the environment (x, y) with an incremental value that starts from 1 before playing the game; this value is referred to as the Hamiltonian path. The mechanism of this agent is similar to the Deep Q agent, i.e., it receives a state and decides its next direction.

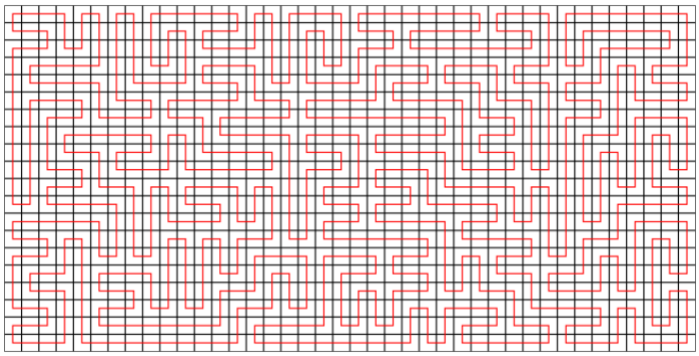


Figure 3. Hamiltonian Cycle in a graph source:(https://codeforces.com/blog/entry/79788?f0a28=1)

The state received by this agent has the information on the previous Hamiltonian path, the agent's current direction, and the size of the environment. Note that since each point in the environment has been mapped with a value and the agent receives information of its previous Hamiltonian path, the agent will use that information and find the next Hamiltonian path that leads to a cycle.

3.3.3 A*

A* is a searching algorithm used to discover the shortest path between two points in a graph. Like a Hamiltonian agent, the A* agent first constructs its environment in an adjacency matrix,

and for each game step, the agent generates the shortest path between the snake's head and the food, with the snake's body and walls being the obstacle to avoid.

Additionally, the mechanism of this agent is also similar to Deep Q and Hamiltonian, where it reasons its next direction based on the given current state. The state information received by this agent is the food position, the snake's current position, the snake's direction, and danger information at the front, left, and right of the snake. Since it has information about its current position and apple, the agent uses an A* algorithm to get all the data points of its environment that lead to the food, with the shortest path.

4 Results and Analysis

Before discussing the agent's performance further, this section will first address the limitations of each agent that become apparent during or before the analysis.

4.1 Agent Limitations

Deep Q Agent One limitation of Deep Q agents is that they both tend to trap the head of the snake inside its body. One might speculate that this is due to the lack of awareness of its own body, and the problem could be mitigated by utilising different model architecture, e.g., CNNs, or RNNs.

Hamiltonian Agent The limitation of this algorithm is that to construct a Hamiltonian cycle, one side of the grid (width or height) must be an even number. If there is no Hamiltonian cycle, the agent will play by always choosing a random move. However, to combat this limitation, this project ensures that at least one side of the environment grid is even. The second limitation of this agent is that since it uses a pure Hamiltonian cycle algorithm, the agent will visit every node once. Therefore, the time for the agent to complete the game takes a considerable amount of time.

A* Agent Despite its powerful searching speed, this agent also has one limitation. If the snake has trapped itself inside its body, the agent cannot find a path between its head and the food. Hence, to address this problem, the agent was programmed to resort to a semi-random move, i.e., a random move, but it tries to avoid a collision.

4.2 Experiment results

The analysis of agents includes Mean, Standard Deviation, Sum of all the scores, Minimum Score, Median Score, and Maximum Score in 3 types of environments: (400 x 400), (320 x 320), and (240 x 240). However, this project will focus more on discussing the mean score, which will be referred to as the performance. Furthermore, as stated in section 3.2, the experiment runs on ten iterations for each agent and environment.

Table 1 shows the descriptive statistics of the result scores for each agent in different environments. Since the Hamiltonian agent guarantees a perfect game if a condition is met, it is not fair to compare the Hamiltonian agent with the rest; therefore, this section will exclude the discussion of the Hamiltonian agent. Nonetheless, for illustration purposes, it is included in the Table.

One of the most notable things from the Table is that the Deep Q agent with more significant input (Deep Q-2) has a better average score than Deep Q-1 in all environments. This suggests that if the agent were given more information about its surrounding environment, it would increase its score. The difference between the two agents is more evident as the environment space shrinks. This makes sense because agents with more input will have more awareness in smaller environments. Hence, this leads to a better decision. From this, it can be concluded that there is a clear relationship between increased input and improved performance.

(400 x 400) Grid Size							
Agents	Mean	Std	Sum	Minimum	Median	Maximum	Total game
Deep Q-1	27.9	7.2	279	9	31.5	43	10
Deep Q-2	28.2	10.1	282	10	26.0	42	10
Hamiltonian	397	0.0	3970	397	397	397	10
A*	59.4	14.3	594	29	61.5	75	10

(a). 400 x 400

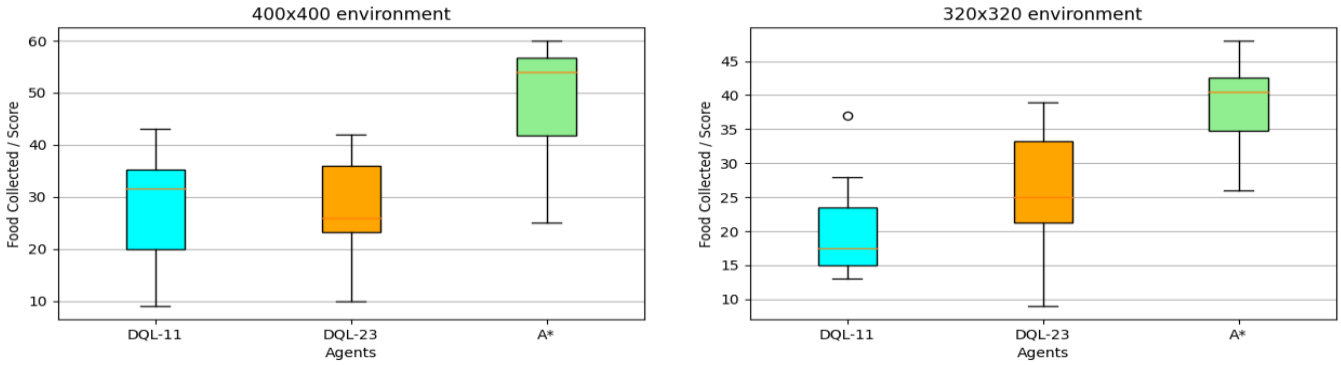
(320 x 320) Grid Size							
Agents	Mean	Std	Sum	Minimum	Median	Maximum	Total game
Deep Q-1	20.4	7.2	204	13	17.5	37	10
Deep Q-2	25.6	9.3	256	9	25.0	39	10
Hamiltonian	253	0.0	2530	253	253	253	10
A*	40.9	11.8	409	22	40.5	57	10

(b). 320 x 320

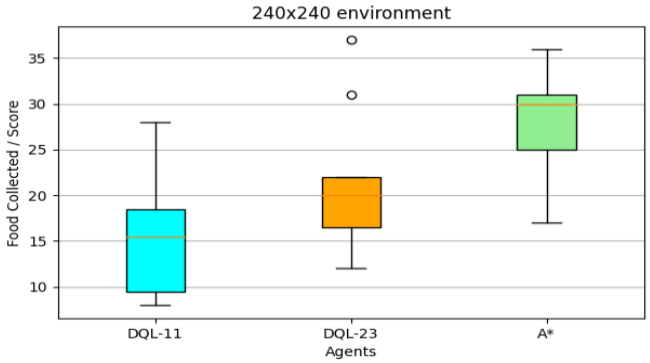
(240 x 240) Grid Size							
Agents	Mean	Std	Sum	Minimum	Median	Maximum	Total game
Deep Q-1	15.2	5.9	152	8	15.5	28	10
Deep Q-2	21.1	7.3	211	12	20.0	37	10
Hamiltonian	141	0.0	1410	141	141	141	10
A*	31.4	5.8	314	21	30.5	42	10

(c). 240 x 240

Table 1. Agents' statistic results in 3 different environments.



(a). 400 x 400 environment (b). 320 x 320 environment



(c). 240 x 240 environment

Figure 4. Agents' results overview in 3 different environments (without Hamiltonian).

Interestingly, the A* agent significantly outperformed Deep Q agents in all environments. However, the gap in performance shrinks as the environment gets smaller. This is because the A* agent only cares about collecting the food with the shortest path without considering it may trap itself with no way out once the food is collected. Therefore, the smaller the environment is, the faster the snake gets bigger, in proportion to its environment, and the higher the chance it will trap itself, leading to a decrease in performance.

Another intriguing finding is that the A* agent has a relatively high standard deviation score in larger environments; this implies that there is an inconsistency in performance. One would expect this anomaly is due to the randomness of food spawning. Finally, when comparing the mean score in proportion to the environment size, all of the agents (except Hamiltonian) have higher scores in smaller environments.

5 Discussion

The early proposed questions can now be answered from the results obtained through the analysis phase. Firstly, are the agents able to adapt to a different environment? It is apparent that all agents can perform relatively well regardless of the environment. Although the Deep Q agents were trained in 400 x 400 grid size, they still achieved a higher mean score –in proportion to the environment– in a 320 x 320 and 240 x 240 space. Thus, when the learning is transferred to a different environment version, it can not only adapt, but also have an increase in performance. Secondly, how do different approaches to building the agents compare in terms of performance? Based on the results, the Deep Q agent with 23 inputs has an overall higher mean score than the 11 inputs agent. The improvement becomes much more evident as the environment shrinks, which leads to a 27% improvement for Agent Deep Q-2 in a 240 x 240 space. Despite that, the Deep Q agents are still outperformed by the A* agent by a considerable margin. The A* agents can achieve 31.2, 15.3, and 10.3 higher scores than the best Deep Q agent in 420 by 420, 320 by 320, and 240 by 240 environments. This implies that deep reinforcement learning is disadvantageous compared to the A* search algorithm when playing the snake game.

The Hamiltonian agent, on the other hand, outshines both the A* and Deep Q agents by completing the game entirely. However, one main drawback for this agent is that it takes a tremendous time to complete the game and generating a Hamiltonian cycle is highly dependent on the environment size. Nevertheless, since the Hamiltonian agent can complete the game without dying, it is the clear winner in this project

6 Conclusion

To summarize, this project was successful. The project's primary goal is to develop multiple intelligent agents to play the snake game and answer the questions proposed early in the report. There are various agents developed in this project, one of which can play the game perfectly, i.e., the Hamiltonian agent. The project can also conclude that Deep Q agents could play relatively well, although they were trained in a different environment. Additionally, although the A* algorithm is theoretically supposed to perform "worse" in a smaller environment (in this project), it managed to surpass all agents (except Hamiltonian) regardless of the environment. In this manner, the project has accomplished its goal.

However, even though all agents have a reasonably good performance, they are still far from perfect. The limitation for both Deep Q and A* agents is that they tend to trap the snake inside its body or in the corner of the environment. Furthermore, The Hamiltonian agent takes a substantial amount of time to complete the game since it needs to explore every node in the environment, and sometimes, a cycle may not be found if a certain condition is not met. Thus, this introduces future work opportunities to perhaps, explore different algorithms, e.g., Perturbated Hamiltonian Cycle algorithm, Dynamic Hamiltonian Cycle Repair algorithm, Different Neural Network architecture (CNNs, RNNs) or even combining multiple algorithms into one, for example, the Deep Q and A* algorithm.

Link to source code: **https://github.com/figoaranta/intelligent_agents_snake_AI**

7 Bibliography

[1] P. Loeber, “GitHub,” 20 December 2020. [Online]. Available: https://github.com/python-engineer/snake-ai-pytorch. [Accessed 1 May 2022].

[2] A. Sebastianelli, M. Tipaldi, S. L. Ullo and L. Glielmo, “A Deep Q-Learning based approach applied to the Snake game,” *2021 29th Mediterranean Conference on Control and Automation (MED),* 2021.

[3] P. Białas, “Implementation of artificial intelligence in Snake game using genetic algorithm and neural networks,” Gliwice, 2019.

[4] N. S. D. Appaji, “Comparison of Searching Algorithms in AI Against Human Agent in Snake Game.,” Faculty of Computing, Blekinge Institute of Technology, Sweden, 2020.