CDS524 Assignment Report:

Reinforcement Learning Snake Game

**1. Introduction**

This report documents the design and test of a Snake game based on

reinforcement learning using Q-learning to train an intelligent agent. The

goal of the project is to demonstrate how an agent learns to navigate in a grid world, collect rewards, and evade punishment through trial and error. Deep Q-Networks (DQN) are implemented in the project for

approximating the Q-function to enable the agent to select actions in a high-dimensional state space.

**2. Game Design**

**2.1 Objective and Rules**

The Snake game is a classic arcade game where a player controls a snake that moves in a finite grid. The main objective is to eat food randomly placed on the grid without hitting the walls and the body of the snake. The snake increases in length every time it eats food, making it harder to maneuver.

The rules of the game:

1.The snake keeps moving in one of the four directions: up, down, left, or right

2.The snake's direction is alterable for the player/agent but never reversible

3.The score increments by one and the length of the snake increases when it eats food

4.Collision with walls or with the snake's own body will end the game

5.The goal is to achieve the highest score by eating as much as possible

**2.2 State and Action Space**

The state space is a collection of 11 binary values for:

1. Danger in front (1 if collision imminent, 0 otherwise)

2. Danger on the right (1 if collision imminent, 0 otherwise)

3. Danger on the left (1 if collision imminent, 0 otherwise)

4. Moving direction is left (1 if true, 0 otherwise)

5. Moving right now (1 if true, 0 otherwise)

6. Moving up now (1 if true, 0 otherwise)

7. Moving down now (1 if true, 0 otherwise)

8. Food is to the left of the snake's head (1 if true, 0 otherwise)

9. Food is to the right of the snake's head (1 if true, 0 otherwise)

10. Food is above the snake's head (1 if true, 0 otherwise)

11. Food is below the snake's head (1 if true, 0 otherwise)

The action space consists of three actions:

1. Move straight [1,0,0]

2. Turn right [0,1,0]

3. Turn left [0,0,1]

This simplified state allows the agent to learn the essential elements of the game state without unnecessary complexity.

**2.3 Reward Function**

The reward function is designed to get the agent to eat food and not collide:

1. +10 points for eating food
2. -10 points for colliding with walls or the snake's body
3. 0 points for any other action

A time penalty is also added: if the snake fails to eat food within a specific number of moves (100 times the snake's length), the game is terminated with a negative reward. This is done to prevent the snake from circling around the grid indefinitely.

**3. Q-Learning Implementation**

**3.1 Deep Q-Network Architecture**

Traditional Q-learning uses a table to memorize Q-values for all state-action pairs. In our implementation, however, due to the vast state space, a neural network is used to approximate the Q-function.

The DQN used has the following architecture:

1. Input layer: 11 neurons (equal to the state space)
2. Hidden layer: 256 neurons with ReLU activation
3. Output layer: 3 neurons (equal to the action space)

This architecture can trade off between training complexity and efficiency. ReLU activation function was applied to the hidden layer since it is non-linear and performs quite well in deep learning models.

**3.2 Learning Algorithm**

The implementation of Q-learning is derived from the principle of adhering to the normal Bellman equation:

Q(s,a) = r + γ \* max(Q(s',a'))

1. s is the current state

2. a is the action taken

3. r is the reward received

4. s' is the next state

5. γ (gamma) is the discount factor and is 0.9

6. max(Q(s',a')) is the maximum Q-value for all actions in the next state

A discount factor of 0.9 was employed to favor long-term rewards but also appreciate immediate rewards where necessary.

**3.3 Exploration**

We use an epsilon-greedy policy to balance exploration and exploitation:

- Epsilon starts high (80) and decreases with each game played

- With probability epsilon/200, choose a random action (exploration)

- Otherwise, choose the best Q-value (exploitation)

This policy allows the agent to learn various strategies early during training and increasingly focus on exploiting its knowledge with continued training.

**4. Game Interaction Design**

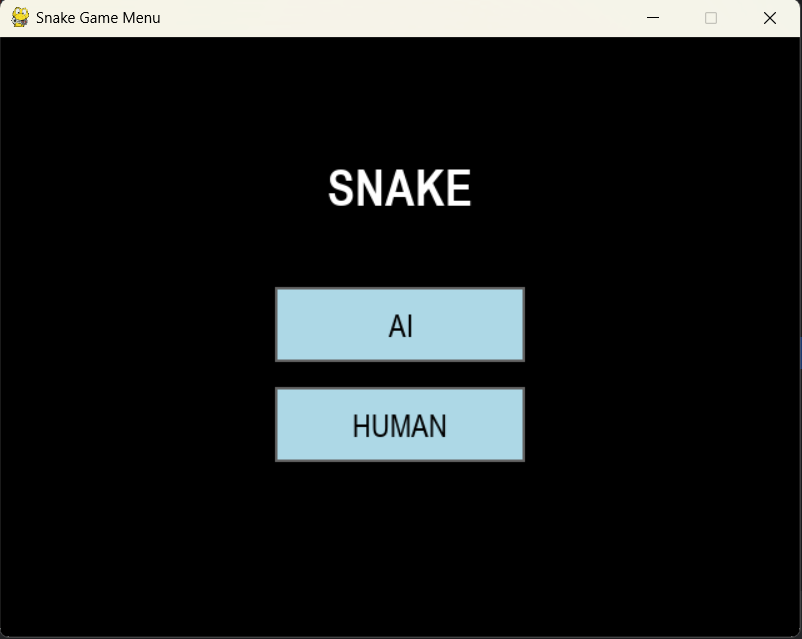
**4.1 User Interface**

The game features a graphical user interface made using the Pygame library. The UI components are:

- A menu screen with "AI" and "Human" options

- Interactive buttons with hover effect for better user experience

- A game screen showing the snake, food, score, and game round



**4.2 Visual Elements**

Visual elements feature:

- Blue snake body and green snake head with white outlines for visibility

- Red food objects

- Black background for visibility

- Game round and score display at top of window



**4.3 Game Mode**

Two modes are introduced within the game:

1. AI Training Mode: Snake is played by Q-learning algorithm with speeded-up gameplay to expedite training

2. Human Play Mode: Snake is played by human player through arrow keys, with slowed-down gameplay for usability

This two-mode mode enables demonstration of the reinforcement learning algorithm and playing interactively.

**5. Learning Progress**

The agent demonstrates clear learning progress in training:

- In early games, movement is largely random with a lot of collisions

- As training goes on, the agent learns to avoid walls and even itself

- In later training, the agent masters the skill of successfully chasing food in an efficient way

**6. Challenges and Solutions**

Throughout the development of this reinforcement learning snake game, several significant challenges were encountered. These challenges required careful consideration and innovative solutions

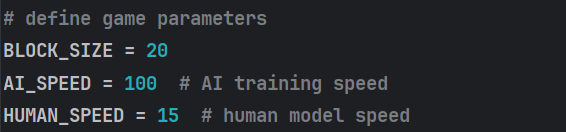
**6.1 Game Speed Management**

**Challenge:**

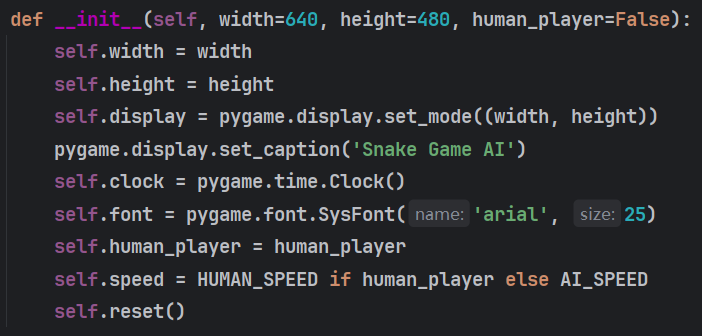
The initial implementation faced a critical dilemma regarding game speed. During AI training, the standard game speed was too slow, resulting in excessively long training periods before meaningful learning progress could be observed. However, when the game speed was increased to accelerate AI training, it became unplayable for human players, as the snake moved too quickly for effective human reaction time.

**Solution:**

To address this challenge, a dual-speed system was implemented:



The game initialization was modified to select the appropriate speed based on the current mode:



This solution allowed for rapid AI training while maintaining comfortable playability for human players.

**6.2 State Representation Design**

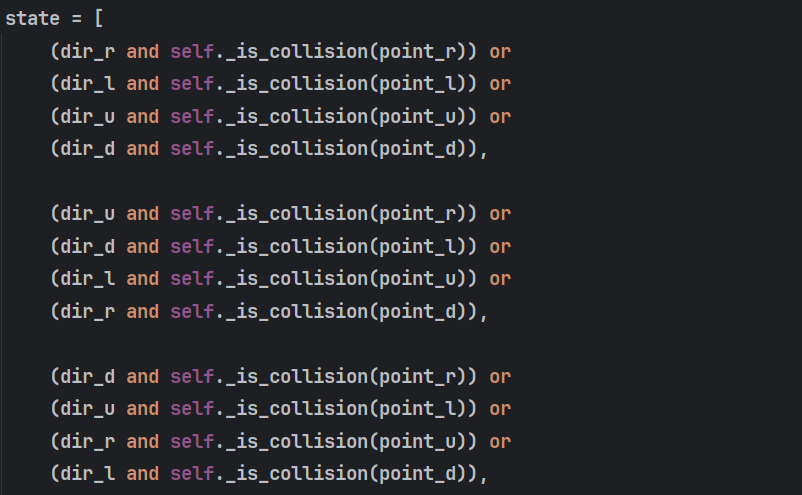
**Challenge:**

It was hardest to find a good state representation. The naive approach would be to represent the entire game grid, but then the resulting state space would be too large, computationally costly, and difficult for the neural network to learn well. At the same time, a very abstract state representation might miss crucial information that is necessary for making a good choice.

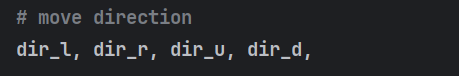
**Solution:**

After experimentation with various state representations, a compact yet informative 11-dimensional binary state vector was designed, focusing on only the most relevant information:

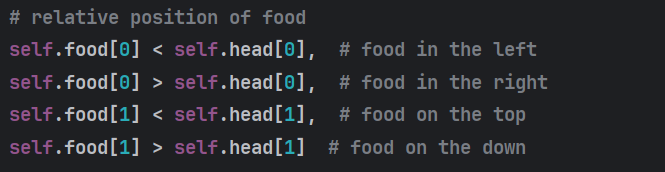
1. Danger detection in three directions



1. Current movement direction :



1. Food relative position



This optimized state representation provided sufficient information for effective learning while keeping the state space manageable. It enabled the neural network to focus on the most critical aspects of the game state: avoiding collisions, maintaining awareness of current direction, and locating food.

**7. Conclusion**

The project is successful in demonstrating the application of reinforcement learning techniques on a classic arcade game. The Q-learning algorithm, as implemented with a deep neural network, learns effectively to traverse the world of the Snake game, avoid collisions, and garner rewards.

The two-mode interface provides support for both illustration of the learning algorithm and interactive game play, making it a complete learning tool for reinforcement learning principles. Future work would include the addition of more complex neural network architectures, deeper state representation, and other reinforcement learning algorithms such as experience prioritized Deep Q-Networks or policy gradient methods.