भारतीय सूचना प्रौद्योगिकी अभिकल्पना, एवं विनिर्माण संस्थान, कांचीपुरम

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, KANCHEEPURAM

Reinforcement Learning Project Report

Project Title: Snake Game Using Deep -Q Learning

Team Members

Sl. No.	Name	Roll Number
1	Swaraj Gambhir	CS21B2024
2	L. Devansh	CS21B2023
3	Anmol Kethan	CS21B2027
4		

RL Project Road Map

Problem Statement

Using reinforcement learning, create a Snake game-playing agent who will aim to score by consuming food pellets while staying out of collisions. Explore novel methods for autonomous gameplay by utilizing neural network architectures and RL techniques, contributing to AI research in gaming settings.

Introduction

A project based on reinforcement learning (RL) is focused on creating a Snake game-playing agent. We define the game environment, establish actions, and create a Q-network architecture that facilitates the agent's decisions. The agent's performance can be improved through the use of Deep Q-Learning, experience replay and training. We aim to balance exploration and exploitation while striving for optimal gameplay. We aim to develop an autonomous agent that can maximize scores without causing collisions by utilizing evaluation and parameter tuning. The endeavor highlights RL's aptitude for handling intricate gaming situations and supports advancements in AI research related to autonomous gameplay.

Approach

This method employs Deep Q-Learning to train an AI agent for autonomous Snake game play. Actions include up, down, left, or right movements in the grid-based environment. Q-values, approximated by a neural network, guide actions. Experience replay buffers past experiences for efficient training. An epsilon-greedy strategy balances exploration and exploitation. Performance evaluation ensures adept navigation, measuring average score and gameplay, with adjustments for optimal results.

- **Environment Definition:** Establishes the game on a grid, where actions involve movements: up, down, left, or right.

- **Q-Network:** Utilizes a neural network to approximate Q-values, representing expected rewards for actions in specific states.

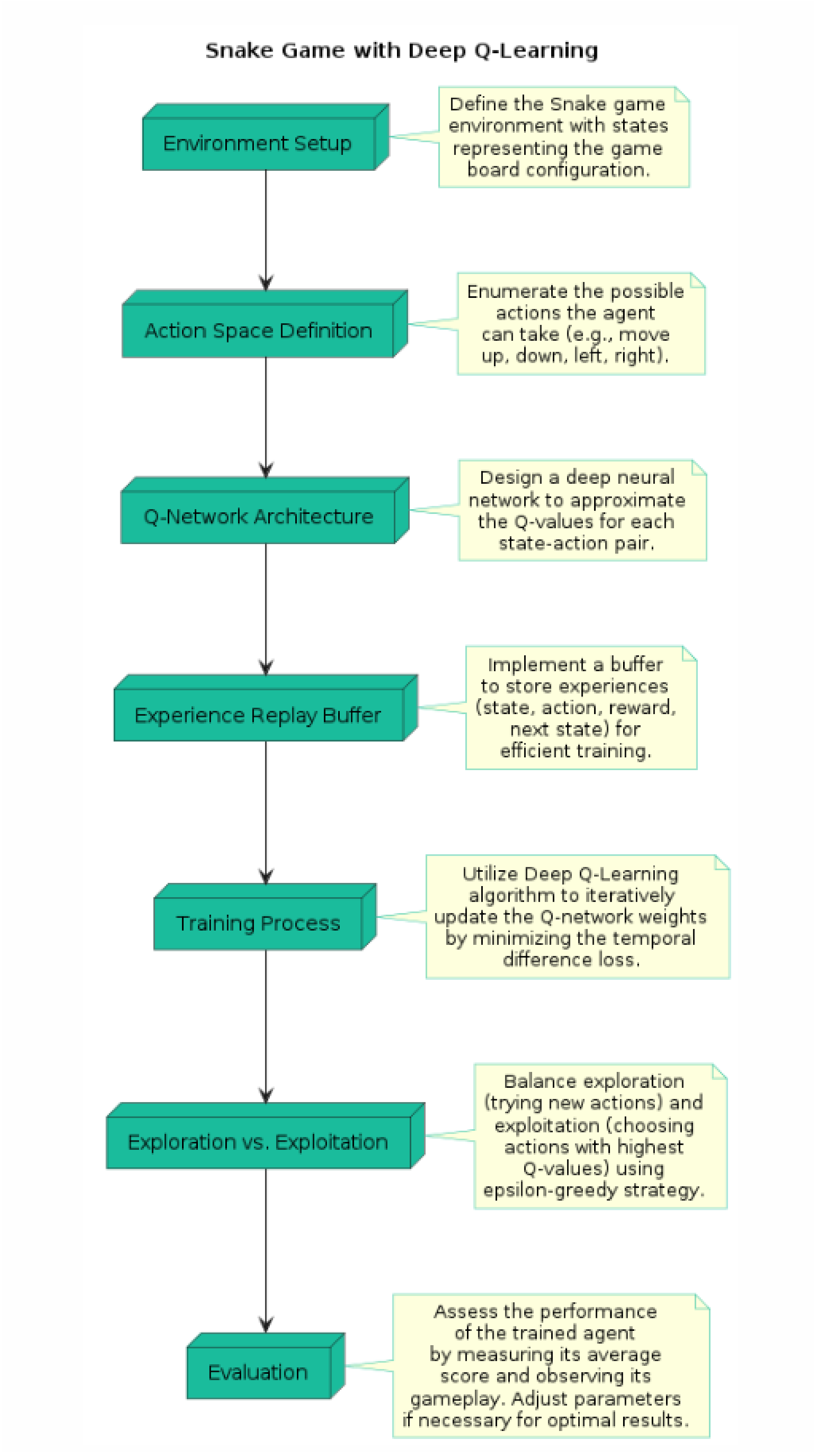
- **Experience Replay:** Stores past experiences in a buffer for efficient training, enhancing learning stability.

- **Training with Deep Q-Learning:** Iteratively updates the Q-network to minimize temporal difference loss, aligning predicted rewards with actual outcomes.

- **Exploration vs. Exploitation:** Implements an epsilon-greedy strategy to balance exploration (trying new actions) and exploitation (choosing actions with highest Q-values).

- **Performance Evaluation:** Assesses the agent's performance through measuring average score and observing gameplay, enabling adjustments for optimal results.

Overall Working Diagram


Snake Game with Deep Q-Learning
Environment Setup
Define the Snake game environment with states representing the game board configuration.
Action Space Definition
Enumerate the possible actions the agent can take (e.g., move up, down, left, right).
Q-Network Architecture
Design a deep neural network to approximate the Q-values for each state-action pair.
Experience Replay Buffer
Implement a buffer to store experiences (state, action, reward, next state) for efficient training.
Training Process
Utilize Deep Q-Learning algorithm to iteratively update the Q-network weights by minimizing the temporal difference loss.
Exploration vs. Exploitation
Balance exploration (trying new actions) and exploitation (choosing actions with highest Q-values) using epsilon-greedy strategy.
Evaluation
Assess the performance of the trained agent by measuring its average score and observing its gameplay. Adjust parameters if necessary for optimal results.

Design Details

1) Environment:

The Snake game environment consists of a grid with boundaries, snake body segments, food pellets, and potential collision hazards.

2) States:

States represent the current configuration of the game board, including the positions of the snake segments, food, and potential collision hazards such as walls or itself.

3) Actions:

Actions encompass movements the snake can take: up, down, left, or right. These actions are discrete choices defining the snake's direction.

4) Reward:

The reward system provides feedback to the agent. Positive rewards are given for consuming food pellets, negative for colliding with walls or itself. A large reward is assigned for completing the game by achieving a specific score.

5) Agent Definition (Mathematical Formulation):

The agent is represented as a Deep Q-Network (DQN), utilizing neural networks. It accepts game states as input and outputs Q-values for each possible action. The agent's policy is to select actions greedily based on these Q-values, incorporating exploration via an epsilon-greedy strategy.

Justification of Approach:

The choice between value-based and policy-based methods depends on the Snake game's characteristics:

Value-Based Methods:

- Suitable for Snake game's discrete state/action spaces and deterministic policy needs.
- DQN excels due to its focus on discrete spaces, efficiently handling Snake's limited action set.

DQN's Advantages for Snake Game:

- Discrete State/Action Spaces: Snake has clear states (grid configurations) and actions (up, down, left, right), ideal for DQN.
- Sample Efficiency: DQN's experience replay enhances learning from past interactions efficiently, vital for resource-intensive games like Snake.
- Focus on Value Learning: Maximizing score aligns with DQN's value-based approach, learning optimal actions for reward maximization.

Policy-Based Approaches and Considerations:

A3C: While suitable for large state spaces, A3C's asynchronous nature may be computationally expensive for Snake. Stability during training could be a concern.

PPO: More stable than A3C, but may require additional hyperparameter tuning and could be less sample-efficient compared to DQN for Snake game scenarios.

6) Deep Q-Network (DQN):

DQN is chosen for its efficient approximation of the Q-function, enabling effective learning in complex environments like Snake.

7) Replay Memory:

Replay memory stores past experiences, facilitating learning from diverse transitions and reducing correlation between updates.

8) Epsilon-Greedy Strategy:

Balancing exploration and exploitation, epsilon-greedy strategy ensures gradual shift from exploration to exploitation as the agent learns about the Snake game environment.

9) Neural Network Representation:

Neural networks approximate the Q-function, capturing game dynamics for informed decision-making based on current states.

10) Training Process:

The agent undergoes training using supervised learning and reinforcement learning techniques, enhancing its policy through repeated interactions and reward feedback from the Snake game environment.

Work Done and Results

1) Implementation of DQN for Snake Game:

Initialization:

- The Linear_QNet class initializes a neural network with input size, hidden size, and output size parameters.
- It defines the neural network architecture using linear layers with ReLU activations.
- PyTorch modules are used for defining the network architecture and the forward pass.
- The model parameters are optimized using the Adam optimizer.
- The class also includes a method for saving the model checkpoints using the PyTorch save method.

Training Method:

- The QTrainer class trains the DQN model by computing target Q-values and minimizing the squared error loss.
- Given batches of states, actions, rewards, next states, and terminal flags, target Q-values for next states are computed.
- The maximum Q-value for each next state is extracted to compute the target.
- The neural network is trained by minimizing the squared error between predicted and target Q-values using backpropagation.
- The Adam optimizer is used to update the model parameters based on the calculated gradients.

2) Training Process:

Conditional Training Start:

- Training commences once a set number of episodes have been accumulated, ensuring a sufficient number of experiences in the replay memory before training begins.

Sampling from Replay Memory:

- During each training step, a batch of experiences is randomly sampled from the replay memory, comprising tuples of (last_state, last_reward, last_action, current_state, terminal).
- Random sampling helps in decorrelating subsequent experiences and stabilizing the training process.

Batch Processing:

- The sampled batch is preprocessed to extract states, rewards, actions, next states, and terminal flags efficiently.
- Components of the batch are converted into numpy arrays to facilitate TensorFlow operations.

Forward Pass and Backpropagation:

- The neural network undergoes training using the sampled batch of experiences.
- The train_step method of the QTrainer class orchestrates a single training step, updating network weights through backpropagation.
- This method returns the updated training step count and the associated cost, aiding in monitoring and logging.

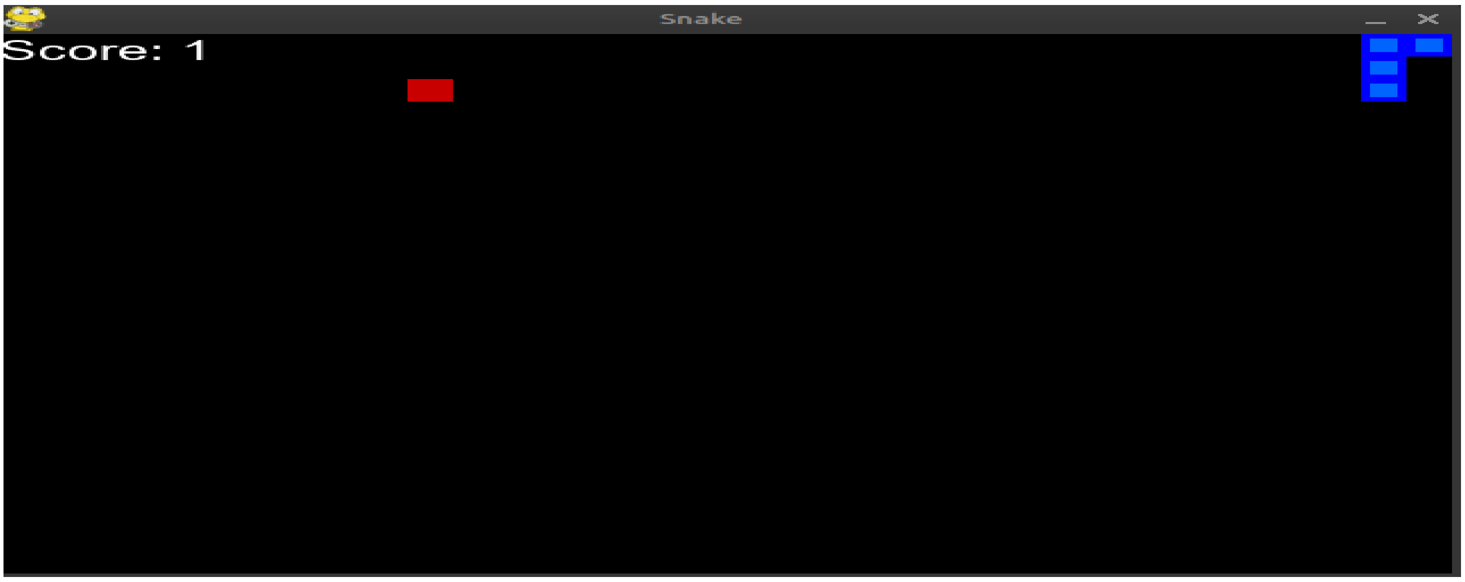
Epsilon Decay:

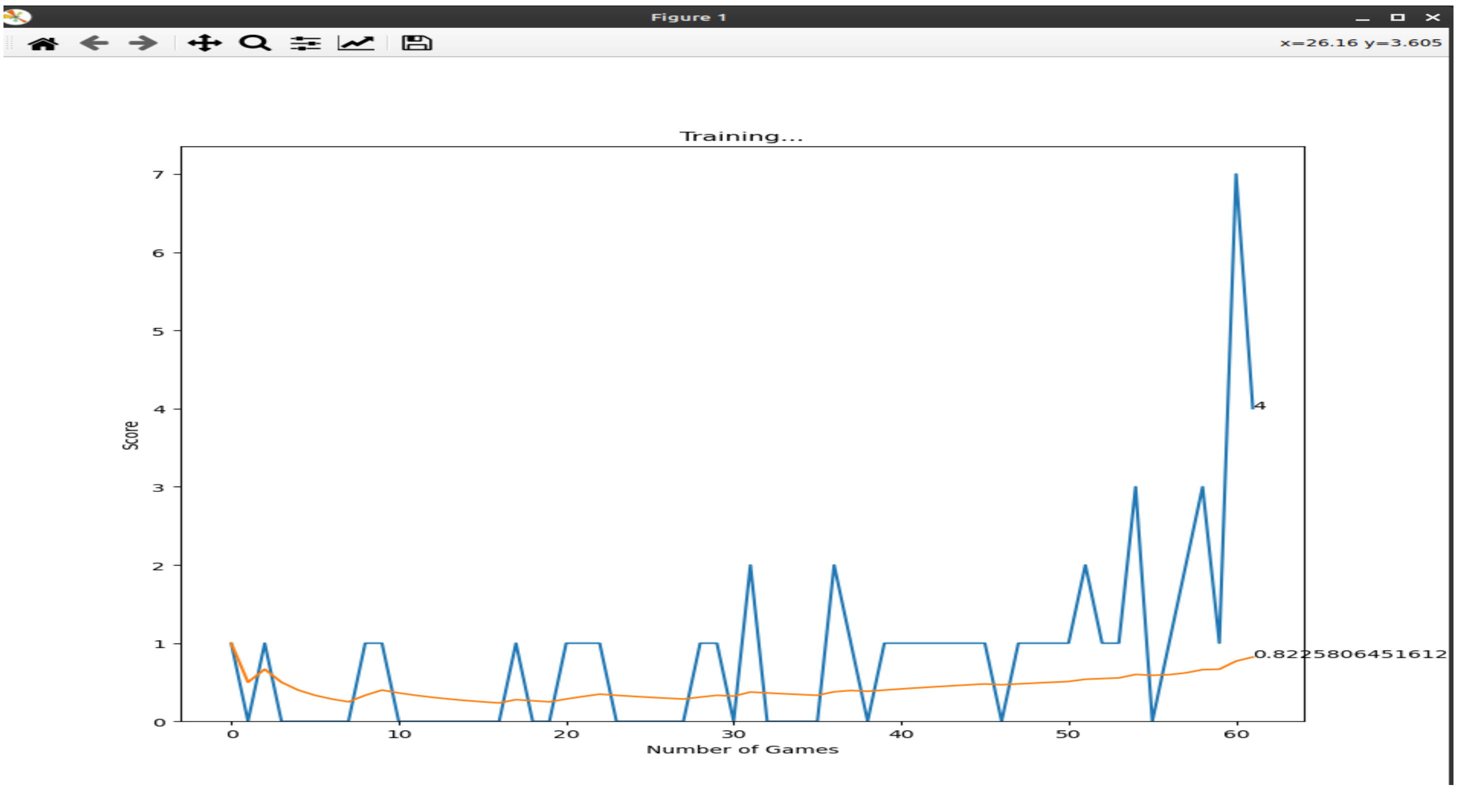
- The exploration-exploitation balance is managed by linearly decaying the epsilon-greedy exploration parameter over time.
- Epsilon gradually decreases from an initial value to a specified final value across a predetermined number of steps, ensuring a transition from exploration to exploitation.

3) Agent's Progress:

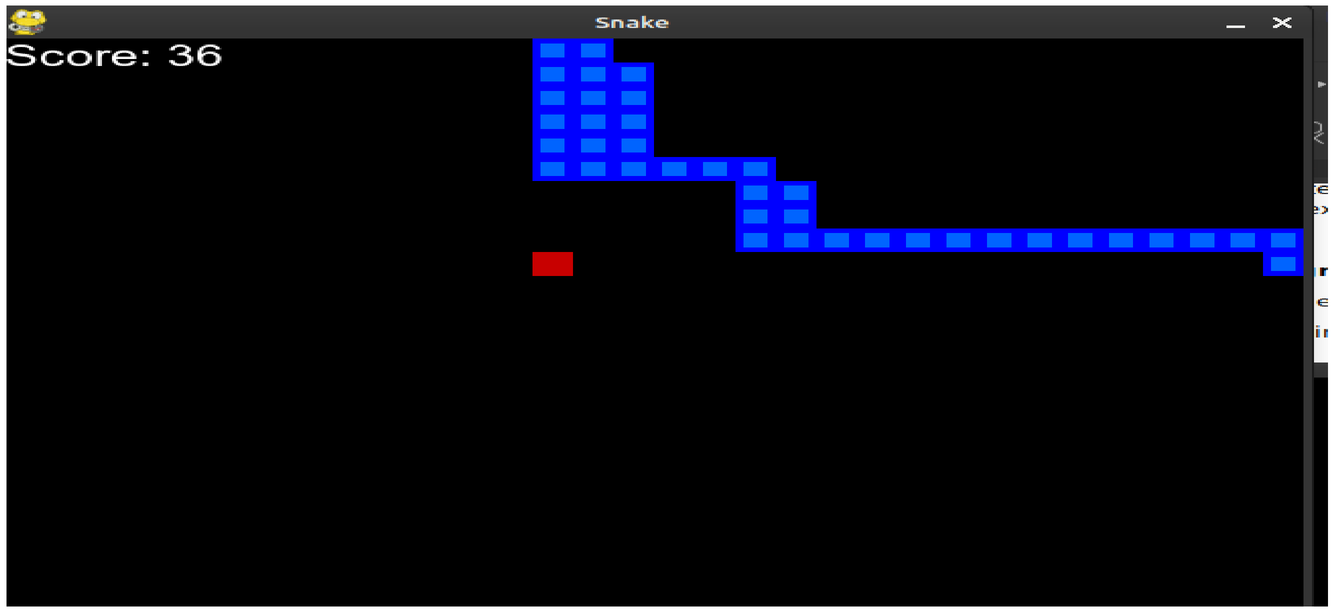
Initial tests show encouraging evidence of the agent's ability to navigate the

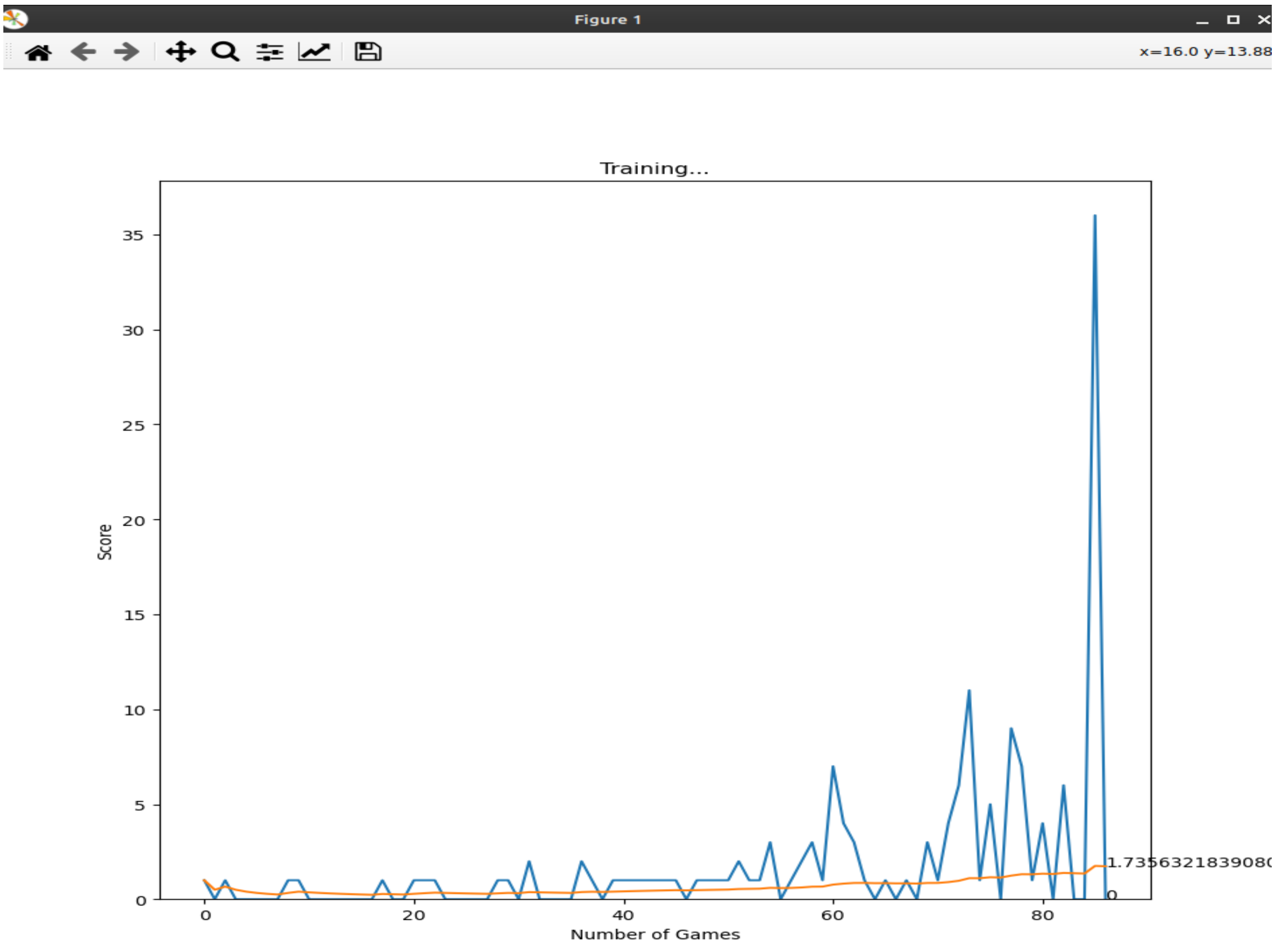
Snake game environment, avoid obstacles effectively and collect rewards.





- Image 1 displays the Snake game interface with a black background. It features the food as a red square and the snake represented by a blue square. A score of 1 is shown in the top-left corner. Image 2 exhibits a training plot for a Deep Q-Learning agent playing Snake. Initially fluctuating, the agent's scores stabilize and improve over time, peaking around the 50th game. These visuals encapsulate the agent's progression in mastering the Snake game.





- Image 1 displays an advanced stage of the Snake game, with the snake (blue squares) having consumed 36 food pieces, elongating its body. Image 2 illustrates the training progress of the Deep Q-Learning agent, showing an upward trend in scores over games played. The agent's growing proficiency is evident, despite occasional score fluctuations, culminating in a notable score of approximately 31 around the 60th game.

4) Performance Metrics:

Game Execution Loop:

The execution loop iterates through a specified number of games.

For each game:

- Depending on the training count, the loop determines whether to suppress output and graphics.
- The game initializes using the reset() method of the GameAI class, creating a new game instance with the specified settings.
- The play_step() method is called on the game instance to start the game simulation.

Recording Game Data:

- If recording is enabled, the loop records the game data by generating a filename based on the game number and current time. It then saves the recorded game components (state, actions, rewards) using pickle for later analysis.

Performance Evaluation:

After running all games, if there are evaluation games (numGames - numTraining > 0):

- Average Score: Calculates and prints the average score across all games.
- Scores: Prints the scores achieved in each game.
- Win Rate: Calculates and prints the ratio of wins to the total number of games as a percentage.
- Record: Summarizes the game outcomes, indicating whether each game resulted in a win or loss.

5) Challenges Faced:

Complexity of Environment:

Representing and modeling dynamic Snake game environments accurately poses challenges due to varying layouts and changing conditions. Handling factors like snake movement, food placement, and collision detection adds complexity to the task.

Agent Exploration:

Learning Snake game involves balancing exploration and use in a productive manner. To maximize food capture, it is important to use exploratory strategies such as epsilon-greedy and optimize parameters for thorough exploration.

Action Space:

To access the discrete action space in Snake game, one must select actions such as moving up, down, left, or right. To effectively navigate the game environment and learn effective strategies, the agent must create a proper representation of the action space and manage these discrete actions efficiently.

Memory Management:

Efficient replay memory is crucial for storing and sampling experiences during training in the Snake game. Managing memory constraints and optimizing usage pose challenges, especially with prolonged agent-environment interactions generating substantial experience volumes.

Hyperparameter Tuning:

The learning rate, discount factor, batch size, and network architecture parameters in DQN for Snake game are hyperparameters that have a significant impact on the agent's learning dynamics and performance, leading to time-consuming optimization. Detailed experimentation is necessary for accurate tuning.

Computational Complexity:

Developing deep neural network reinforcement learning for Snake game requires computational power. Mini-batch training and GPU acceleration are methods that reduce complexity, while model compression and lightweight architectures support efficient training under resource limitations.

Contributions:

Team Member	Contribution
Swaraj Gambhir (CS21B2024)	Graphics creation and utilities,DQN Architecture creation,Layout coding, GUI creation,and randomization of agent
Devansh L (CS21B2023)	Fine Tuning,Training of Snake DQN Agent, game training and state-action creation,Game rules.
Anmol Kethan (CS21B2027)	Game Environment creation (including states,actions etc) ,Game Implementation,Running and evaluation