CSD311(ARTIFICIAL INTELLIGENCE)

CLASSIC SNAKE GAME(USING AI) PROJECT REPORT

Group 23

Group Members:

Prahansha Kumaravel	2110110376
Saransh Saxena	2110110465
Tarang Verma	2110110550
Mansi	2110110623
Abhiveer Singh	2110110820
Pragunjay Singh	2110110874

TABLE OF CONTENTS

Abstract

This project explores the integration of Artificial Intelligence (AI) techniques to automate and optimize the classic Snake Game, a timeless 2D grid-based challenge. By implementing heuristic search algorithms (Greedy Best-First Search, A* algorithm) alongside a Reinforcement Learning approach (Deep Q-Network), the AI system demonstrates enhanced gameplay decision-making. The project aims to maximize the snake's length while avoiding collisions, navigating dynamically generated environments, and adapting strategies over multiple games. Results reveal that heuristic-based algorithms provide basic strategic navigation, while reinforcement learning delivers superior adaptability and performance, achieving the highest scores and proving its potential for mastering complex gaming environments.

Introduction

The classic Snake Game, a simple yet engaging 2D grid-based arcade challenge, has been a popular source of entertainment for decades. The objective is to control a snake that grows in length each time it consumes food while avoiding collisions with walls or its own body. Despite its simplicity, the game's dynamic nature and increasing difficulty make it a compelling test of reflexes and strategic thinking.

Traditionally, players manually navigate the snake to achieve the highest possible score. However, this project focuses on the second scenario: leveraging Artificial Intelligence (AI) to automate the snake's movements and optimize its decision-making. Using Python as the primary programming language, we implemented various AI algorithms in the Pygame library, creating intelligent bots capable of playing the game.

The AI strategies employed include heuristic search methods like Greedy Best-First Search, A*, and an Improved A* algorithm, along with Reinforcement Learning (RL) using Deep Q-Networks (DQN). These methods enable the bot to navigate intelligently, avoid collisions, and maximize its score by evaluating the game state and making optimal decisions. The integration of PyTorch and

TensorFlow further enhanced the AI agent's learning and adaptability.

The ultimate aim is to equip the snake with AI capabilities that allow it to achieve maximum length while minimizing risky moves. This seemingly straightforward task involves intricate challenges, such as navigating dynamic environments, managing collisions, and predicting future states. By addressing these challenges, this project demonstrates the potential of AI in solving real-time decision-making problems, offering a fresh perspective on an enduring classic.

Problem Analysis

- Game Dimensions:
  - Height - 480 pixels = 24 blocks
  - Width - 640 pixels = 32 blocks
- Each block represents one unit of space occupied by the snake's body or the food particle.
- The snake can move up, down, left and right, and can only make right angle turns.
- The snake must go and eat the apple, which will increase its length by 1 block.
- The apple is produced randomly on the grid on any available empty space.
- The goal is to get the highest score possible.
- The game will end if:
  - The snake collides with the walls of the game
  - The snake collides with its own body
- When the snake dies, it reappears randomly on the grid with a length of 1.

Algorithm Overview

Greedy Best-First Search

The Greedy Best-First Search algorithm focuses on finding the shortest path to the food using Manhattan distance, calculated as:

$$|x1-x2|+|y1-y2|$$

Here, (x1,y1) represents the snake's head coordinates, and (x2,y2) represents the food's position coordinates.

- The algorithm prioritizes moving towards the food particle along the shortest path without considering potential collisions with the snake's body.
- It is implemented via a function, `move_towards_food`, which calculates the optimal direction for the snake's movement based on the food's position relative to the snake's head.
- If the snake's head is left of the food, it moves right; if right, it moves left.If the snake's head is above the food, it moves down; if below, it moves up.
- This method lacks foresight, often resulting in collisions, making it suboptimal for complex scenarios.

A* Search Algorithm

The **A*** Search algorithm combines the cost of the current path with a heuristic estimate of the distance to the goal (food). It employs the formula:

$f(n)=g(n)+h(n)$

where,g(n) is the cost of the path from the snake's starting position to the current food and h(n) is the heuristic estimate of the distance from the snake’s head to the food.

- It evaluates potential moves by considering both the cumulative distance already traveled and the estimated distance to the food.
- Guides the snake towards the food efficiently while minimizing risks.
- **Advantages**:
    - Balances safety and efficiency.
    - Finds optimal paths in static environments.
- **Limitations**:
    - Computationally intensive, especially for dynamic game states.

Reinforcement Learning (Deep Q-Network)

Reinforcement Learning (RL) is a branch of machine learning focused on training software agents to make decisions within an environment by maximizing cumulative rewards. This process involves teaching the agent appropriate behaviors through feedback on its performance, enabling it to improve over time.

Deep Q-Learning: This approach extends the capabilities of RL by integrating a deep neural network to predict and anticipate optimal actions. This combination allows the agent to make informed decisions, leveraging the power of deep learning to handle complex scenarios more effectively.

Training Process: During training, the agent interacts with a game environment. It begins by obtaining its current state, determining an action using the model, and executing that action. The environment then provides feedback in the form of rewards, updated states, and scores. This feedback loop is used to iteratively train and refine the model.

Components:

- Game Environment (Pygame): The environment provides the context for the agent's actions, delivering rewards and feedback after each step. Actions such as moving straight, turning right, or turning left are executed using a play step function that returns the reward, game status, and score.
- Model (PyTorch): The decision-making model is implemented as a Linear Q-Network (DQN) in PyTorch. It predicts actions

based on the agent's current state and is trained iteratively to improve its performance.

Rewards and Actions:

Rewards:
Eat food: + 10
Game over: − 10
Else:0

Action:
[1, 0, 0] → straight
[0, 1, 0] → right turn
[0, 0, 1] → left turn

State Representation:The state of the game, a critical input for the agent, comprises 11 boolean values reflecting various aspects, including dangers in different directions and the relative position of food.

[danger straight, danger right, danger left, direction left, direction right, direction up, direction down, food left, food right, food up, food down]

Deep Q-Learning Q-Value and Bellman Equation:

The Q-value represents the quality of a specific action in a given state and is iteratively updated using the Bellman equation:

$$NewQ(s,a)=Q(s,a)+\alpha[R(s,a)+\gamma \cdot \max Q'(s',a')-Q(s,a)]$$

Where:

- NewQ(s,a): Updated Q-value
- Q(s,a): Current Q-value
- R(s,a): Reward for the action
- maxQ′(s′,a′): Maximum future reward
- α: Learning rate
- γ: Discount factor

The simplified Q-update rule is:

Q=model.predict(state), NewQ=R+γ⋅max(Q(state′))

Loss Function

The training process uses a mean squared error loss function to minimize the difference between predicted Q-values and updated Q-values, ensuring the model learns accurate decision-making over time:

Loss=(NewQ−Q)^2

In summary this integrated approach combines reinforcement learning, deep Q-learning, and game environment simulation to train an agent for optimal decision-making in dynamic scenarios. The use of the Bellman equation and a well-defined loss function ensures effective learning and continuous improvement, making the agent increasingly adept at navigating complex environments.

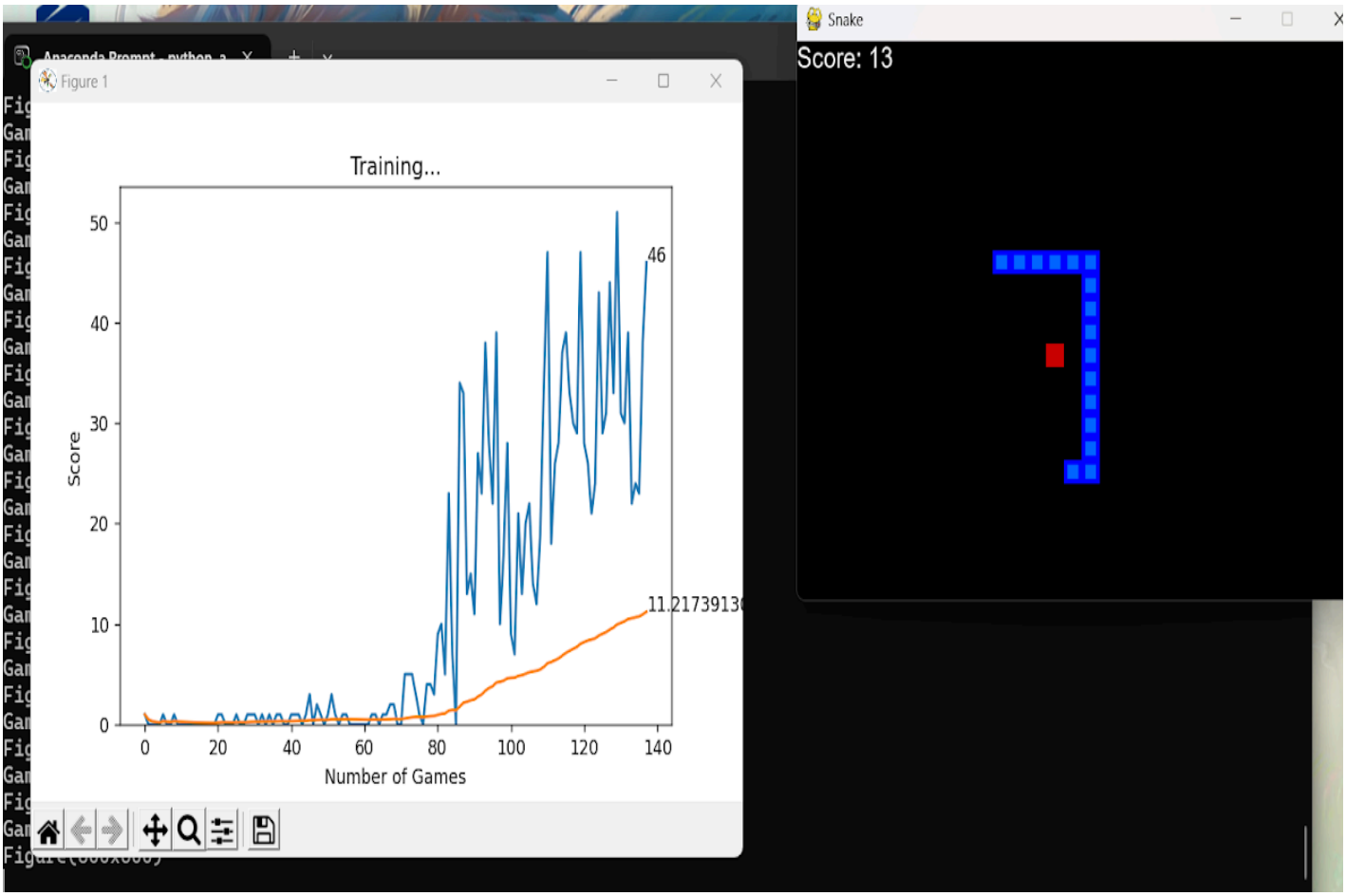
Observations

Our analysis highlights distinct strengths and limitations among the evaluated algorithms:

- Greedy Best-First Search provides quick solutions but lacks foresight, often resulting in lower scores due to its short-sighted decision-making approach.
- The A* algorithm improves performance by balancing cost and heuristic, leading to better decision-making compared to Greedy Best-First Search.
- Deep Q-Networks (DQN), as a reinforcement learning approach, displayed a learning curve that allowed it to progressively refine its strategies through iterative training. Over 200 evaluated games, DQN showcased remarkable adaptability, achieving competitive scores and demonstrating its capability to master complex environments effectively.

Results

DQN achieved the highest adaptability and competitive scores among all evaluated algorithms, effectively mastering complex environments.



Reinforcement Learning scores for 140 games

The project's code is available at this link: [Link]

Conclusion

The choice of algorithm depends on the desired balance between computational efficiency and adaptability. Greedy Best-First Search and A* are well-suited for scenarios requiring quick and simple solutions. However, Improved A* and DQN excel in strategic and adaptive decision-making, with DQN particularly standing out for its ability to refine strategies and adapt over extended training periods.