SNIOE

CSD311: ARTIFICIAL INTELLIGENCE

PROJECT REPORT

TEAM 24: SNAKE 15*15

PROF. SNEHASIS MUKHERJEE

HARSHIT GANGWAR	- 2110110621
ABHINAV SINGH	- 2110110061
ASHUTOSH KUMAR	- 2110110069
AMAN KUMAR	- 2110111204
ANUJ MAURYA	- 2110111214

Introduction

The classic game of Snake has been entertaining players for decades, and it continues to be a popular choice for both casual and competitive gamers. The game is simple to learn but difficult to master, making it a perfect challenge for players of all ages.

In a Snake game, the player controls a snake that moves around a 2d grid. The snake grows longer with each food cell it eats, and the goal is to make the snake as long as possible without hitting the walls or its own body.

The game starts with the snake occupying a single cell in the centre of the grid. The player uses the arrow keys to control the snake's direction, and the snake moves one cell in that direction each time the player presses a key.

Food cells are randomly placed on the grid, and when the snake eats a food cell, it grows one cell longer. The snake can also eat its own body, but if it does so, it dies.

The game ends when the snake hits a wall, hits its own body, or fills the entire grid. The player's score is the length of the snake at the end of the game.

Snake game is a fun and challenging game that can be enjoyed by people of all ages. It is a great way to improve hand-eye coordination and problem-solving skills.

Environment and Agent type:

In a 2D grid snake game, the environment is a grid where the snake moves, and each cell represents a spatial position. The agent, which is the player controlling the snake, navigates the grid to collect food and grow longer while avoiding collisions with itself and the grid boundaries. The environment is dynamic, influenced by the agent's actions, creating a strategic gameplay experience with the goal of maximizing the score by surviving and collecting food. The agent's decisions impact the state of the grid, making it a discrete and interactive environment for the snake game.

Performance measure:

The performance of the snake game is typically measured by the length of the snake at the end of the game. This is because the goal of the game is to grow the snake as long as possible. A longer snake indicates that the snake has been able to navigate the grid and avoid collisions effectively, while also consuming food in minimum number of steps.

Problem Analysis:

The goal of our project is to find an automatic solution for the Snake 15*15 game, where the snake should navigate through the grid to eat the food, grow longer, and avoid collisions with itself and the grid boundaries.

Challenges:

- **Limited Space**: The 15*15 grid provides a confined area for the snake to move, making it increasingly difficult to avoid its own body as it grows longer.
- **Snake Growth Mechanism**: The snake increases in length every time it consumes a food cell.
  As the snake grows, the challenge intensifies, necessitating careful planning to avoid collisions with the snake's own body.
- **Random Food Placement**: The unpredictable placement of food cells adds an element of randomness to the game, making it challenging to consistently reach new food sources.
- **Optimizing Moves**: The objective is to make the snake as long as possible using a minimal number of moves.

Solver Ideation:

The Snake game solver employs the A* algorithm for pathfinding to control the snake's movement towards the randomly positioned food on a grid. The Snake class handles the snake's state, direction, and movement, while the find_path_to_food method uses A* with a priority queue to compute the optimal path to the food. The heuristic function calculates the Manhattan distance between two points. The main loop iterates through events, updates the screen, and invokes the snake's movement and drawing functions. If the snake collides with the screen boundaries or itself, the game exits. The grid, snake, and food are drawn using Pygame, and the game runs at a specified frames-per-second rate. The solver ensures the snake efficiently navigates the grid to consume the food and grow in length.

Data Structures Used:

The Snake game code utilizes several data structures. The Snake class maintains the snake's state using a list (**self.positions**) to store its body positions. A deque (**visited**) is employed to keep track of visited nodes during A* pathfinding. The A* algorithm uses a priority queue implemented with a heap (**heap**) to efficiently explore possible paths. Additionally, a Pygame Rect (**food**) represents the position and dimensions of the food on the grid. The code leverages basic data structures like lists, deques, and heaps for efficient storage and traversal in the context of grid-based gameplay.

Deadlock Conditions:

The Snake game code employs deadlock conditions to handle situations where the snake cannot move further without colliding with the screen boundaries or itself. Specifically, if the next calculated position of the snake is outside the grid boundaries or collides with its own body, the code triggers a change in the snake's direction. If all potential directions lead to deadlock (collision), the game prints the snake's length and terminates. The deadlock conditions ensure that the game handles scenarios where the snake cannot progress, preventing infinite loops or unintended behaviours.

Algorithms Used:

The Snake game code employs the A* algorithm for pathfinding to navigate the snake towards the food on the grid efficiently. A* is a search algorithm that uses a heuristic to prioritize paths and explore them in a way that minimizes the total cost. In this context, the A* algorithm, implemented with a priority queue (heap), calculates the optimal path from the snake's head to the food, considering grid positions and avoiding obstacles. The heuristic function used in A* is the Manhattan distance, which measures the straight-line distance between two points on a grid. Additionally, the code utilizes basic algorithms for collision detection and deadlock resolution, such as checking if the next position is within grid boundaries and handling collisions with the snake's own body. Overall, the A* algorithm is the key algorithmic component for intelligent pathfinding in the Snake game.

Code Flow:

The Snake game code initializes the Pygame environment and defines the grid, cell size, and frame settings. It then creates a Snake object and a food object with initial positions. The main game loop runs, handling user events and updating the screen. Within each iteration, the grid and snake are drawn, and the snake's movement is controlled by the A* algorithm for pathfinding. The algorithm computes the optimal path to the food, and the snake follows this path, updating its position and length. If the snake collides with the screen boundaries or itself, the game terminates, printing the final snake length. The loop continues until the user exits the game window. The code structure ensures continuous gameplay with dynamic updates based on user input and intelligent pathfinding using the A* algorithm.

Memory Optimization used:

In the given Snake game code, memory optimization has been achieved using memory-efficient data structures. The Snake class utilizes a deque for storing body positions, providing constant-time complexity for adding and removing elements compared to a list. This minimizes memory overhead when updating the snake's positions. Additionally, the A* algorithm employs a priority queue implemented with a heap for efficient pathfinding, optimizing the storage and retrieval of nodes during traversal. By utilizing these memory-conscious data structures, the code aims to reduce unnecessary memory allocations and improve overall efficiency in handling the snake's state and pathfinding computations.