UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+
Accredited University

PROJECT REPORT:-
Smart Snake Game with BFS AI

Submitted by: Amandeep Kumar Mishra

UID: 24MCI10172

Sub – AI-Lab

Program: MCA (AI/ML)

Semester: 1st

Group: 1-B

Academic Year: 2024–25

Institute: Chandigarh University

Table of Contents

Introduction

The Smart Snake Game with BFS AI is an advanced version of the classic Snake game, where the snake is controlled by a Breadth-First Search (BFS) algorithm to find the optimal path to food. This AI-driven version demonstrates the application of search algorithms in game development, highlighting how intelligent agents can make decisions in dynamic environments. It serves as a useful project for learning about game logic, pathfinding, and artificial intelligence in Python.

Objective

The objective of this project is to implement a smart AI system using Breadth-First Search (BFS) to navigate the snake towards its target (food) while avoiding collisions. The project aims to enhance understanding of pathfinding algorithms and their application in real-time game environments.

Technologies Used

• Python

• Pygame library

• Breadth-First Search (BFS) algorithm

Project Descriotion

This Smart Snake Game includes:

- A visual grid of 30x20 cells.
- A snake that starts small and grows with each food item it consumes.
- An AI-driven movement mechanism powered by BFS.
- Real-time pathfinding that helps the snake reach the food using the shortest path while avoiding collisions.

Code

```
import pygame
import sys
import random
from collections import deque

pygame.init()

GRID_SIZE = 20
GRID_WIDTH = 30
GRID_HEIGHT = 20

SCREEN = pygame.display.set_mode((GRID_SIZE * GRID_WIDTH,
GRID_SIZE * GRID_HEIGHT))
pygame.display.set_caption("Smart Snake Game with BFS AI")

WHITE = (255, 255, 255)
GREEN = (0, 200, 0)
RED = (200, 0, 0)
BLACK = (0, 0, 0)
GRAY = (100, 100, 100)

clock = pygame.time.Clock()
FPS = 10

DIRECTIONS = {
    "UP": (0, -1),
    "DOWN": (0, 1),
    "LEFT": (-1, 0),
    "RIGHT": (1, 0)
}

def bfs(start, goal, snake_body):
    queue = deque([start])
    visited = set()
    parent = {}
```

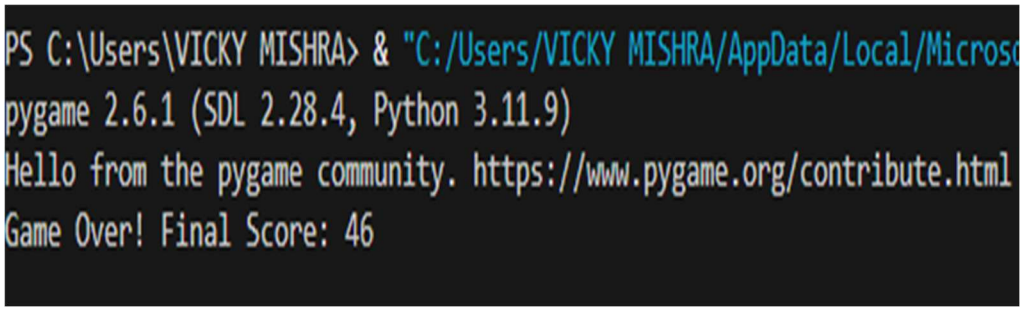
```
    visited.add(start)

    while queue:
        current = queue.popleft()
        if current == goal:
            break

        for dir in DIRECTIONS.values():
            neighbor = (current[0] + dir[0], current[1] + dir[1])
            if (0 <= neighbor[0] < GRID_WIDTH and
                0 <= neighbor[1] < GRID_HEIGHT and
                neighbor not in visited and
                neighbor not in snake_body):
                visited.add(neighbor)
                parent[neighbor] = current
                queue.append(neighbor)

    path = []
    if goal in parent:
        current = goal
        while current != start:
            path.append(current)
            current = parent[current]
        path.reverse()
    return path

class SnakeGame:
    def __init__(self):
        self.reset()

    def reset(self):
        self.snake = [(5, 5)]
        self.direction = "RIGHT"
        self.spawn_food()

    def spawn_food(self):
        while True:
```

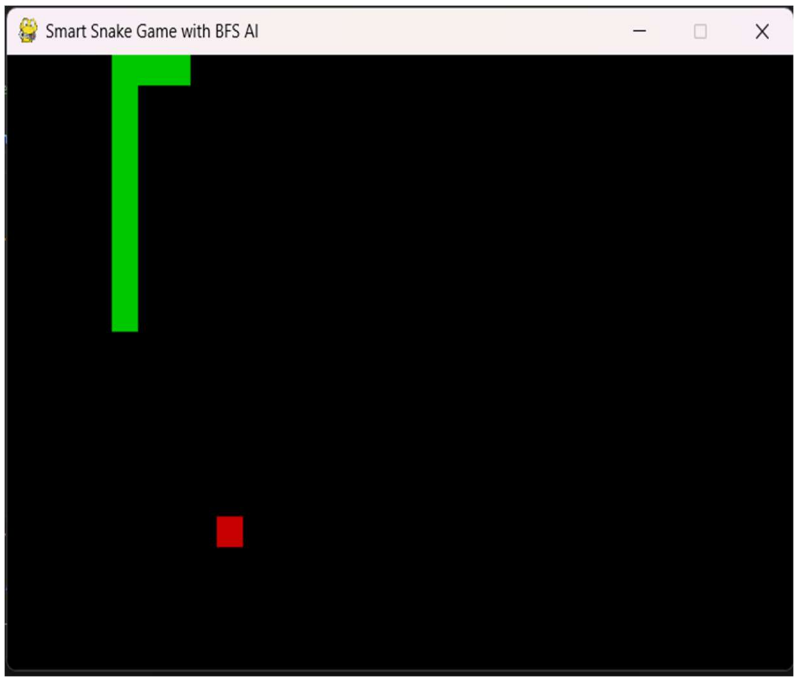
```
            self.food = (random.randint(0, GRID_WIDTH - 1), random.randint(0,
GRID_HEIGHT - 1))
            if self.food not in self.snake:
                break

    def move_snake(self, next_pos):
        if next_pos == self.food:
            self.snake.insert(0, next_pos)
            self.spawn_food()
        else:
            self.snake.insert(0, next_pos)
            self.snake.pop()

    def draw(self):
        SCREEN.fill(BLACK)
        for segment in self.snake:
            pygame.draw.rect(SCREEN, GREEN, (segment[0]*GRID_SIZE,
segment[1]*GRID_SIZE, GRID_SIZE, GRID_SIZE))
        pygame.draw.rect(SCREEN, RED, (self.food[0]*GRID_SIZE,
self.food[1]*GRID_SIZE, GRID_SIZE, GRID_SIZE))
        pygame.display.update()

    def game_over(self):
        head = self.snake[0]
        return (head in self.snake[1:] or
                head[0] < 0 or head[1] < 0 or
                head[0] >= GRID_WIDTH or head[1] >= GRID_HEIGHT)

    def run(self):
        while True:
            clock.tick(FPS)
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

            path = bfs(self.snake[0], self.food, set(self.snake))
```

```
            if path:
                next_step = path[0]
                self.move_snake(next_step)
            else:
                dx, dy = DIRECTIONS[self.direction]
                next_pos = (self.snake[0][0] + dx, self.snake[0][1] + dy)
                self.move_snake(next_pos)

            if self.game_over():
                print("Game Over! Final Score:", len(self.snake) - 1)
                pygame.time.wait(2000)
                self.reset()

            self.draw()

if __name__ == "__main__":
    SnakeGame().run()
```

Output & Result

The snake successfully navigates towards the food using the shortest possible path calculated by the BFS algorithm. The game continues until the snake collides with the wall or itself. This demonstrates how AI can enhance gameplay and decision-making in real-time applications.





Advantages

- Demonstrates use of BFS in real-time applications.
- Enhances logical and algorithmic thinking.
- Interactive and visually engaging.

Applications

- Educational tools for teaching pathfinding algorithms.
- Foundations for AI in gaming.
- Concept learning for robotics navigation.

Limitations

- BFS may not be optimal for very large or complex environments.
- Game ends if the snake traps itself or there is no valid path.

Conclusion

This project demonstrates the practical application of pathfinding algorithms in a classic game scenario. By integrating the BFS algorithm into the Snake game, the snake acts intelligently and autonomously, improving the gameplay experience and showcasing how AI concepts can be applied to real-time problems.

References

- Pygame Documentation: https://www.pygame.org/docs/
- Python Official Docs: https://docs.python.org/
- BFS Algorithm: https://en.wikipedia.org/wiki/Breadth-first_search