Bits manipulators

PAC-MAN Game

-The Team:

-Menna Abdo Saeed Ahmed 23102385

-Mohammed Ayman Mohammed 23101455

- kerolos Nader 23101441

-Wesam Kamal Nafea 23101501

-peter moawad magdalla 23101374

-Introduction :

The project is a simple simulation of the modern Pac-Man game using Python and the Pygame library. The game consists of Pac-Man, ghosts, walls, and pellets, and the main objective is for Pac-Man to collect all the pellets while avoiding the ghosts using A* algorithm.

Objective of the Script:

The objective of the script is to implement basic game mechanics, including:

- Moving Pac-Man towards pellets using A* search.
- Allowing ghosts to move randomly or use A* search to chase Pac-Man.
- Handling game-over conditions based on collisions with ghosts or when all pellets are collected.
- Displaying the game grid, including walls, pellets, Pac-Man, and ghosts.

This script demonstrates basic AI concepts, like pathfinding (A* algorithm), within a game environment.

Methodology

1. Initial Setup and Imports

```
import pygame
import random

# Initialize pygame
pygame.init()
```

We first import the pygame library for creating the game window and handling graphics.
random is used to randomly place walls and pellets on the grid.
pygame.init() initializes all the modules in Pygame.

2. Constants and Configuration

```
# Define screen dimensions and grid size
SCREEN_WIDTH = 400
SCREEN_HEIGHT = 400
GRID_SIZE = 20

# Define colors
PACMAN_COLOR = (255, 255, 0)  # Yellow for Pac-Man
GHOST_COLOR = (255, 0, 0)  # Red for Ghosts
WALL_COLOR = (0, 0, 255)  # Blue for Walls
PELLET_COLOR = (0, 255, 255)  # Cyan for Pellets
BACKGROUND_COLOR = (0, 0, 0)  # Black for the background

# Create game window
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption("Pac-Man Game")

# Define possible movement directions (right, left, down, up)
DIRECTIONS = [(0, 1), (0, -1), (1, 0), (-1, 0)]
```

-These are the basic settings for the game, including screen dimensions (SCREEN_WIDTH and SCREEN_HEIGHT), grid size (GRID_SIZE), and colors for different game elements (Pac-Man, ghosts, walls, and pellets).
- pygame.display.set_mode() creates the game window with the given screen dimensions.
-DIRECTIONS defines the possible movement directions for Pac-Man and ghosts.

3. Grid Creation Function

```
def create_grid():
    """
    Creates the grid for the game, populating it with walls and pellets.

    Returns:
        grid (list): A 2D list representing the game grid.
        pellets (list): A list of coordinates where pellets are located.
    """
    grid = []
    pellets = []
    for y in range(0, SCREEN_HEIGHT, GRID_SIZE):
        row = []
        for x in range(0, SCREEN_WIDTH, GRID_SIZE):
            if random.random() < 0.1:
                row.append(1)  # Wall
            else:
                row.append(0)  # Empty space
                if len(pellets) < 10 and random.random() < 0.1:  # Place pellets
                    row[-1] = 2  # Pellet
                    pellets.append((x // GRID_SIZE, y // GRID_SIZE))  # Store pellet position
        grid.append(row)
    return grid, pellets
```

-This function creates the game grid. It iterates over each cell and decides randomly whether it should be a wall (1), an empty space (0), or a pellet (2).
- Pellets are placed in random empty spaces, and their coordinates are stored in the pellets list

*4. A Search Algorithm**

```python
def a_star_search(start, goal, grid):
    """
    Performs A* search to find the shortest path from the start to the goal.

    Args:
        start (tuple): The starting position (x, y).
        goal (tuple): The goal position (x, y).
        grid (list): The game grid containing walls and empty spaces.

    Returns:
        list: The path from start to goal, or an empty list if no path exists.
    """
    open_nodes = [(start, 0)]  # Nodes to explore, with initial cost 0
    visited_nodes = {}  # Dictionary to track the cost of visited nodes
    path_tracker = {}  # Dictionary to track the path for reconstruction

    while open_nodes:
        open_nodes.sort(key=lambda x: x[1] + manhattan_distance(x[0], goal))  # Sort by cost + heuristic
        current_node, g_cost = open_nodes.pop(0)

        # If goal is reached, reconstruct the path
        if current_node == goal:
            path = []
            while current_node in path_tracker:
                path.append(current_node)
                current_node = path_tracker[current_node]
            path.reverse()  # Reverse the path to get it from start to goal
            return path

        visited_nodes[current_node] = g_cost  # Mark node as visited

        # Explore neighbors
        for dx, dy in DIRECTIONS:
            neighbor = (current_node[0] + dx, current_node[1] + dy)
            if (0 <= neighbor[0] < SCREEN_WIDTH // GRID_SIZE and
                0 <= neighbor[1] < SCREEN_HEIGHT // GRID_SIZE and
                grid[neighbor[1]][neighbor[0]] != 1):  # Ensure the neighbor is within bounds and not a wall

                new_cost = g_cost + 1  # Increment cost for moving to a neighbor

                if neighbor not in visited_nodes or new_cost < visited_nodes[neighbor]:
                    visited_nodes[neighbor] = new_cost
                    open_nodes.append((neighbor, new_cost))
                    path_tracker[neighbor] = current_node

    return []  # Return an empty path if no path found
```

- The A* search algorithm is used to find the shortest path from the start (Pac-Man or a ghost) to the goal (a pellet or Pac-Man).
- It combines two values: the cost of reaching a node (g_cost) and a heuristic (Manhattan distance) to the goal.
- The algorithm explores neighbors of the current node and continues until it reaches the goal or exhausts all options.

5. Pac-Man Class

```python
class PacMan:
    """
    Represents the Pac-Man character, its position, and score.
    """
    def __init__(self):
        self.x = SCREEN_WIDTH // 2 // GRID_SIZE
        self.y = SCREEN_HEIGHT // 2 // GRID_SIZE
        self.score = 0

    def move(self, goal, grid):
        """
        Moves Pac-Man towards the goal using A* search.

        Args:
            goal (tuple): The goal position (x, y).
            grid (list): The game grid.
        """
        path = a_star_search((self.x, self.y), goal, grid)
        if path:
            next_step = path[0]  # Move to the next step in the path
            self.x, self.y = next_step

    def collect_pellet(self, grid, pellets):
        """
        Collects a pellet if Pac-Man is on its position and updates the score.

        Args:
            grid (list): The game grid.
            pellets (list): The list of remaining pellets.
        """
        if (self.x, self.y) in pellets:
            pellets.remove((self.x, self.y))  # Remove pellet from the list
            self.score += 1  # Increase score
```

-The PacMan class represents the Pac-Man character. It contains attributes for position (x and y) and score.
- move() uses the A* search to move Pac-Man toward a target (typically a pellet).
- collect_pellet() checks if Pac-Man is on a pellet's position and updates the score.

6. Ghost Class

```python
class Ghost:
    """
    Represents a ghost in the game, which can move towards Pac-Man.
    """
    def __init__(self, x, y, is_ai_ghost=False):
        self.x = x
        self.y = y
        self.is_ai_ghost = is_ai_ghost  # Whether this ghost uses AI

    def move(self, pacman, grid):
        """
        Moves the ghost towards Pac-Man. If AI ghost, uses A* search.

        Args:
            pacman (PacMan): The Pac-Man object.
            grid (list): The game grid.
        """
        try:
            if self.is_ai_ghost:  # AI-controlled ghost
                path = a_star_search((self.x, self.y), (pacman.x, pacman.y), grid)
                if path:
                    next_step = path[0]  # Move to the next step in the path
                    self.x, self.y = next_step
            else:  # Random movement for non-AI ghost
                if random.random() < 0.4:  # 40% chance to use A* for random ghost
                    path = a_star_search((self.x, self.y), (pacman.x, pacman.y), grid)
                    if path:
                        next_step = path[0]
                        self.x, self.y = next_step
                    else:
                        dx, dy = random.choice(DIRECTIONS)  # Move randomly
                        self.x += dx
                        self.y += dy
                else:
                    dx, dy = random.choice(DIRECTIONS)  # Move randomly
                    self.x += dx
                    self.y += dy
        except Exception as e:
            print(f"Error moving ghost at ({self.x}, {self.y}): {e}")  # Error handling for unexpected issues
```

-The Ghost class represents a ghost in the game.
-It can move randomly or chase Pac-Man using the A* search if is_ai_ghost is True.
- Ghosts that don't use AI move randomly with some probability of switching to pathfinding (A* search).

7. Main Game Loop

```
while running:
    try:
        screen.fill(BACKGROUND_COLOR)

        # Check for game over condition (all pellets collected)
        if len(pellets) == 0:
            game_over = "Pac-Man Won!"
            running = False

        # Move Pac-Man towards the first pellet (if any)
        if pellets:
            pacman.move(pellets[0], grid)
            pacman.collect_pellet(grid, pellets)

        # Move ghosts and check for collision with Pac-Man
        for ghost in ghosts:
            ghost.move(pacman, grid)
            if ghost.x == pacman.x and ghost.y == pacman.y:
                game_over = "Game Over!"
                running = False

        # Draw the grid (walls)
        for y in range(len(grid)):
            for x in range(len(grid[y])):
                if grid[y][x] == 1:  # Wall
                    pygame.draw.rect(screen, WALL_COLOR, pygame.Rect(x * GRID_SIZE, y * GRID_SIZE, GRID_SIZE, GRID_SIZE))
                elif grid[y][x] == 0:  # Empty space
                    pygame.draw.rect(screen, BACKGROUND_COLOR, pygame.Rect(x * GRID_SIZE, y * GRID_SIZE, GRID_SIZE, GRID_SIZE))

        # Draw pellets
        for pellet in pellets:
            pygame.draw.rect(screen, PELLET_COLOR, pygame.Rect(pellet[0] * GRID_SIZE + 4, pellet[1] * GRID_SIZE + 4, GRID_SIZE - 8, GRID_SIZE - 8))

        # Draw Pac-Man
        pygame.draw.rect(screen, PACMAN_COLOR, pygame.Rect(pacman.x * GRID_SIZE, pacman.y * GRID_SIZE, GRID_SIZE, GRID_SIZE))

        # Draw ghosts
        for ghost in ghosts:
            ghost_color = GHOST_COLOR if not ghost.is_ai_ghost else (0, 255, 0)  # AI ghosts are green
            pygame.draw.rect(screen, ghost_color, pygame.Rect(ghost.x * GRID_SIZE, ghost.y * GRID_SIZE, GRID_SIZE, GRID_SIZE))

        # Display the score
        score_font = pygame.font.Font(None, 30)
        score_text = score_font.render(f"Score: {pacman.score}", True, (255, 255, 255))
        screen.blit(score_text, (10, 10))

        # Display game over message
        if game_over:
            font = pygame.font.Font(None, 50)
            text = font.render(game_over, True, (255, 255, 255))
            screen.blit(text, (SCREEN_WIDTH // 2 - text.get_width() // 2, SCREEN_HEIGHT // 2 - text.get_height() // 2))

        pygame.display.flip()  # Update the display
        clock.tick(10)  # Set the game speed

    except Exception as e:
        print(f"Error during game loop: {e}")
```

- This is the main game loop. It continuously updates the screen, moves Pac-Man and the ghosts, and checks for collisions.
- If Pac-Man collects all pellets or collides with a ghost, the game ends.

Flowchart of the game logic:

-Initialization: Set up the screen, grid, and characters

-Movement: Pac-Man and ghosts move based on A* search or random choices.

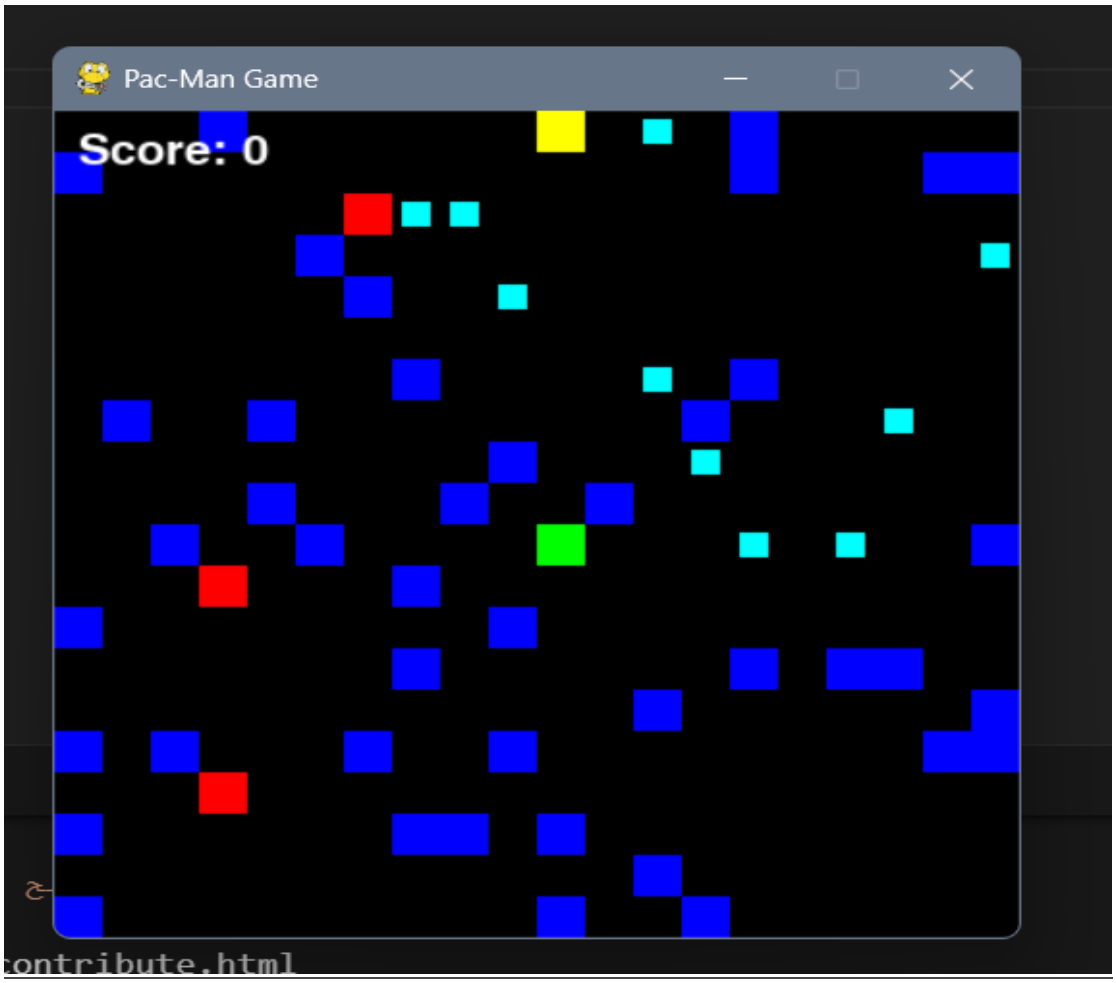
-Game Conditions: Check for game-over (Pac-Man wins or loses).

-Rendering: Draw elements on the screen (walls, pellets, Pac-Man, ghosts).

-Repeat: The loop continues until a game-over condition is met.

Results

When the script is run, you will see something like this on the game screen:



Graphical Interface:

-A screen is displayed with a grid consisting of walls (in blue) and pellets (in cyan) that Pac-Man collects.
-Pac-Man (in yellow) and the ghosts (in red) are shown on the screen. Pac-Man and the ghosts move within the grid.
-When all pellets are collected, a message appears stating that Pac-Man won.
-If Pac-Man collides with one of the ghosts, a "Game Over" message is displayed.

Score:

The score is updated each time Pac-Man collects a pellet.

Movement of Pac-Man and Ghosts:

-If the ghost is using the A* algorithm, it moves intelligently to get closer to Pac-Man.
- Other ghosts move randomly.

Observed Issues or Limitations:

The probability of Pac-Man winning is low because one ghost moves using the algorithm, while the other three move randomly, making the game's challenge difficult.

Roles and Contributions

(23102385)

Create the game grid with walls and pellets.

Report

(23101455)

PacMan Class

(23101441)

Ghost class

(23102501)

Implement the A* algorithm for pathfinding.
Define utility functions like Manhattan distance.

(23101374)

Game Loop and Rendering

Conclusion

The game features Pac-Man navigating a grid, collecting pellets, and avoiding ghosts. One ghost uses the A* algorithm to intelligently chase Pac-Man, while the others move randomly. The goal was to demonstrate the use of the A* algorithm for pathfinding and to simulate basic AI behavior for the ghosts.

Results Achieved:

- Pac-Man moves towards the nearest pellet using the A* algorithm.

- Ghosts either chase Pac-Man using A* or move randomly.

- The game ends when Pac-Man collects all the pellets or collides with a ghost.

-The score updates as Pac-Man collects pellets.

Future Improvements:

Add sound effects for actions like collecting pellets or encountering ghosts.

Implement difficulty levels by varying the number of ghosts or speed of movement.

Enhance the game's visual appeal with better graphics and animations.

References

We used **pygame** library and **A* Algorithm** which we learned in labs

And **Random module** to generate random numbers