School of Mechanical & Manufacturing Engineering (SMME),

National University of Science and Technology (NUST),

Sector H-12, Islamabad

Program: BE-Aerospace Section: AE-01

Session: Fall 2024 Semester: 3rd

Course Title: Fundamentals of Programming II (CS-109)

Project Report

“ *Pacman Simplified* ”

Name: Muhammad Awais Ali

CMS: 461465

Table of Contents

Pygame Pacman Simplified Game

Introduction

This project implements a simplified version of the classic Pacman game using the pygame library. The game includes interactive elements such as walls, blocks (food), ghosts, and a player-controlled Pacman. The core game logic involves Pacman navigating a grid, eating blocks, avoiding ghosts, and earning points. The following report provides an in-depth analysis of the libraries, classes, and functionalities used in the implementation.

1. Libraries Used

Pygame Library `import pygame`

Pygame is a Python library designed for writing the codes for video games. It includes computer graphics and sound libraries, which are extensively used in this project. Key components of pygame used in this game are:

1. **pygame.init()** `pygame.init()`
    - Initializes all pygame modules. It is necessary to call this function before using other pygame functionalities.
    - Sets up the environment for the game, including display, sound, and input handling.
2. **pygame.display**
    - **pygame.display.set_mode()**: Creates the game window of specified dimensions (400x400 pixels in this case).
    ```
    screen_size = (400, 400)
    screen = pygame.display.set_mode(screen_size)
    ```
    - **pygame.display.set_caption()**: Sets the title of the game window.
    ```
    pygame.display.set_caption('Pacman Simplified')
    ```
    - **pygame.display.set_icon()**: Sets the game window's icon (loaded from pacman.png).
3. **pygame.Surface**
    - Used for creating game objects like walls and blocks. It acts as a 2D image object on which colors, shapes, or sprites can be rendered.
4. **pygame.sprite.Sprite** `(pygame.sprite.Sprite)`
    - A core pygame class that simplifies managing game objects. It provides methods for rendering, grouping, and collision detection. In this game it used for the same purposes like the detection of the wall for both pacman and the ghosts.
    - All major components of the game (Pacman, ghosts, walls, blocks) are subclasses of pygame.sprite.Sprite.
5. **pygame.mixer** `pygame.mixer.music.load('pacman.mp3')`
    - Handles audio in the game.
    - **pygame.mixer.init()** initializes the mixer module.

- **pygame.mixer.music.load()** loads background music (pacman.mp3).
- **pygame.mixer.music.play()** starts the background music on a loop.

6. **pygame.time.Clock**

```
# Setup the clock for frame rate control
clock = pygame.time.Clock()
```

   - Manages the frame rate of the game.
   - **clock.tick(30)** limits the game to run at 30 frames per second. Which is also used in this game.

7. **pygame.font**

```
font = pygame.font.Font(None, 15)
```

   - Used to render text for displaying the score, game over messages, and instructions for replaying the game.

8. **pygame.event**

```
pygame.event.get()
```

   - Handles user inputs such as key presses or window close events.
   - **pygame.KEYDOWN**: Detects when a key is pressed, enabling movement control of Pacman.

```
event.type == pygame.KEYDOWN:
```

9. **pygame.sprite.Group**

```
all_sprites = pygame.sprite.Group()
walls = pygame.sprite.Group()
blocks = pygame.sprite.Group()
ghosts = pygame.sprite.Group()
```

   - Groups multiple sprites for efficient rendering and collision detection.
   - Used for managing walls, blocks, ghosts, and all game objects.

10. **pygame.sprite.spritecollide()**
    - Detects collisions between sprites.
    - Used to check:
      - Pacman colliding with walls.
      - Pacman eating blocks.
      - Ghosts colliding with walls or Pacman.

Random Library

```
import random
```

- The random module is part of Python's standard library and is used for generating random choices.
- **random.choice()**: Randomly selects a movement direction for ghosts ("up", "down", "left", or "right").

```
self.direction = random.choice(["up", "down", "left", "right"])
```

- **random.randint()**: Determines the duration for which a ghost maintains a specific direction. In this case not the time is used but the coordinates are used means that if the ghosts for this length of the coordinates, then it changes direction.

```
self.change_direction_timer = random.randint(30, 60)
```

2. Game Classes

The game logic is divided into classes for better modularity and reusability.

- **Wall Class** `class Wall(`
  - Represents walls in the game that Pacman and ghosts cannot pass through.
  - **Attributes:**
  - image: A pygame.Surface object to define the wall's appearance.
  - rect: Defines the position and size of the wall. `self.rect.topleft = (x, y)`
  - **Constructor Parameters:**
  - x, y: Top-left corner of the wall.
  - width, height: Dimensions of the wall.
  - color: RGB color of the wall. In this game blue color is assigned to the walls.
- **Player Class** `class Player(`
  - Represents Pacman, controlled by the player.
  - **Attributes:**
  - image: A scaled-down pacman.png image.
  - rect: Represents Pacman's position and size.
  - speed: Movement speed of Pacman (6 units per frame).
  - **Methods:**
  - __init__: Initializes Pacman's position, size, and appearance.
  - update(x_change, y_change): Updates Pacman's position based on input from arrow keys.
- **Block Class** `class Block(`
  - Represents the blocks (food) that Pacman eats to earn points.
  - **Attributes:**
  - image: A small yellow rectangle (7x7 pixels).
  - rect: Position of the block.
  - Blocks disappear when Pacman collides with them. In this code the True means that when the collision is done between pacman and block then the block will disappear.

```
blocks_hit = pygame.sprite.spritecollide(pacman, blocks, True)
score += len(blocks_hit)
```

- **Ghost Class** `class Ghost(`
  - Represents the ghosts that Pacman must avoid.

- **Attributes:**
    - image: Ghost image loaded from a file (e.g., Blinky.png).

```
ghost_positions = [(130, 130), (280, 280), (180, 180), (30, 30)]
ghost_images = ["Blinky.png", "Inky.png", "Clyde.png", "Pinky.png"]
```

    - positions: Position of all ghost is defined in the coordinates, it means when the game starts then each ghost will start its movement from that specific coordinates.
    - rect: Position of the ghost.
    - speed: Movement speed (3 units per frame).

```
self.speed = 3
```

    - direction: Current movement direction (e.g., "up", "down", etc.).
    - walls: Reference to the wall group for collision detection.
    - change_direction_timer: Timer for changing movement direction.
- **Methods:**
    - __init__: Initializes ghost position, speed, and image.
    - update(): Moves the ghost in its current direction. If it collides with a wall, it changes direction.

3. Game Setup

- **Walls**
    - Walls are created at predefined positions and added to the walls sprite group.
    - The walls form a boundary around the screen and obstacles within the play area.
- **Blocks**
    - Blocks are placed at predefined positions within the grid.
    - Each block is added to the blocks sprite group, and their collision with Pacman

```
blocks_hit = pygame.sprite.spritecollide(pacman, blocks, True)
score += len(blocks_hit)
```

 increments the score.
- **Ghosts**
    - Ghosts are placed at specific starting positions.
    - Each ghost is initialized with a unique image and added to the ghosts sprite group.

4. Game Loop

The game_loop() function contains the main game logic. Key elements include:

Event Handling

- Detects key presses to control Pacman's movement.
- Handles the QUIT event to close the game.

```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_y:
        return True
    elif event.key == pygame.K_n:
        return False
```

Movement and Collision

- Updates Pacman's position based on player input.

```
if pygame.sprite.spritecollide(pacman, walls, False):
    pacman.update(-x_change, -y_change)
```

- Checks for collisions with walls and reverts movement if necessary. When Pacman collides with a wall, it has already moved into the wall in the current frame. Here -x_change and -y_change is used after the collision. This effectively moves Pacman back to its position before the collision occurred.
- Detects collisions with blocks to increment the score.
- Updates ghosts' positions and changes their direction upon wall collision.

Rendering

- Draws all sprites and the score on the screen.
- Displays "Game Over" or "You Win" messages when applicable.

Replay Option

- After the game ends, the player is given the option to replay or quit using the play_again() function.

```
def play_again():
    while True:
        screen.fill(black)
        display_message("If You want To Play Again then Press P and To Quit The Game Press Q.", white)
        pygame.display.flip()
```

5. Sound and Graphics

Sound

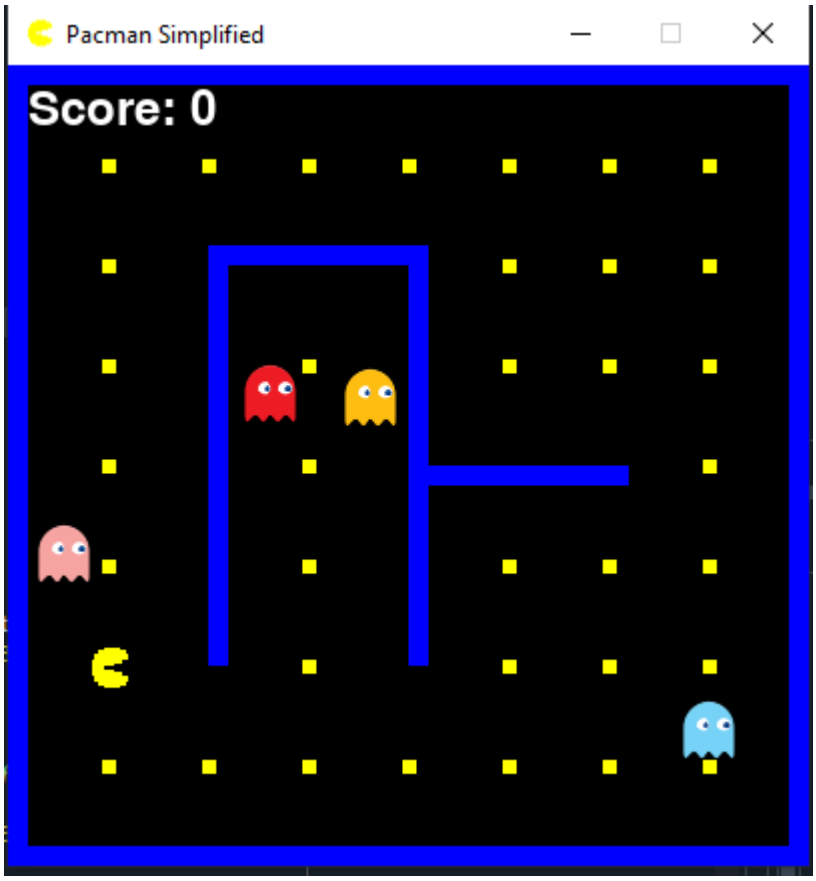
- Background music (pacman.mp3) is loaded and looped using pygame's mixer module.
- Enhances the immersive experience of the game.

```
pygame.mixer.init()
pygame.mixer.music.load('pacman.mp3')
pygame.mixer.music.play(-1, 0.0)
```

Graphics

- Sprites for Pacman and ghosts are loaded as images (pacman.png, Blinky.png, etc.).
- Walls and blocks are drawn using pygame.Surface objects filled with specific colors.

6. Output



7. Conclusion

This project demonstrates the use of pygame to create a simplified version of Pacman with modular design and engaging gameplay. Key features include smooth sprite movement, collision detection, and dynamic ghost behavior. The implementation showcases the versatility of pygame in handling graphics, input, and audio for game development. This structure provides a foundation for further enhancements, such as adding levels, more ghosts, or power-ups.