COMPLEX COMPUTING PROBLEM

Artificial Intelligence & Expert Systems AI&ES (CT-361)

GROUP MEMBERS

Ahsan Ali (CT-22080)

Mudassir Anis (CT-22073)

Farman Ali (CT-22079)

Kumail Raza Walji (CT-22078)

1. Introduction

Haunted Scape Maze is a thrilling 2D maze game developed using Python and Pygame. The game demonstrates key Operating System concepts such as system calls and memory management through an interactive horror-based environment. The player must escape a procedurally generated haunted maze while being chased by a monster. The game integrates file handling, process control, and real-time input/output to reflect the role of system calls in dynamic environments like games.

2. Game Overview

In *Haunted Scape Maze*, the player navigates a haunted labyrinth. The objective is to reach the exit before the monster catches the player. The monster chases the player using a smart movement algorithm. The maze is generated randomly every time the game is started, ensuring a unique experience for each run.

Key Features:

- Procedural maze generation.
- Player and monster character logic.
- Real-time keyboard input handling.
- Chase mechanics using player tracking.
- Increasing difficulty with monster speed based on proximity.

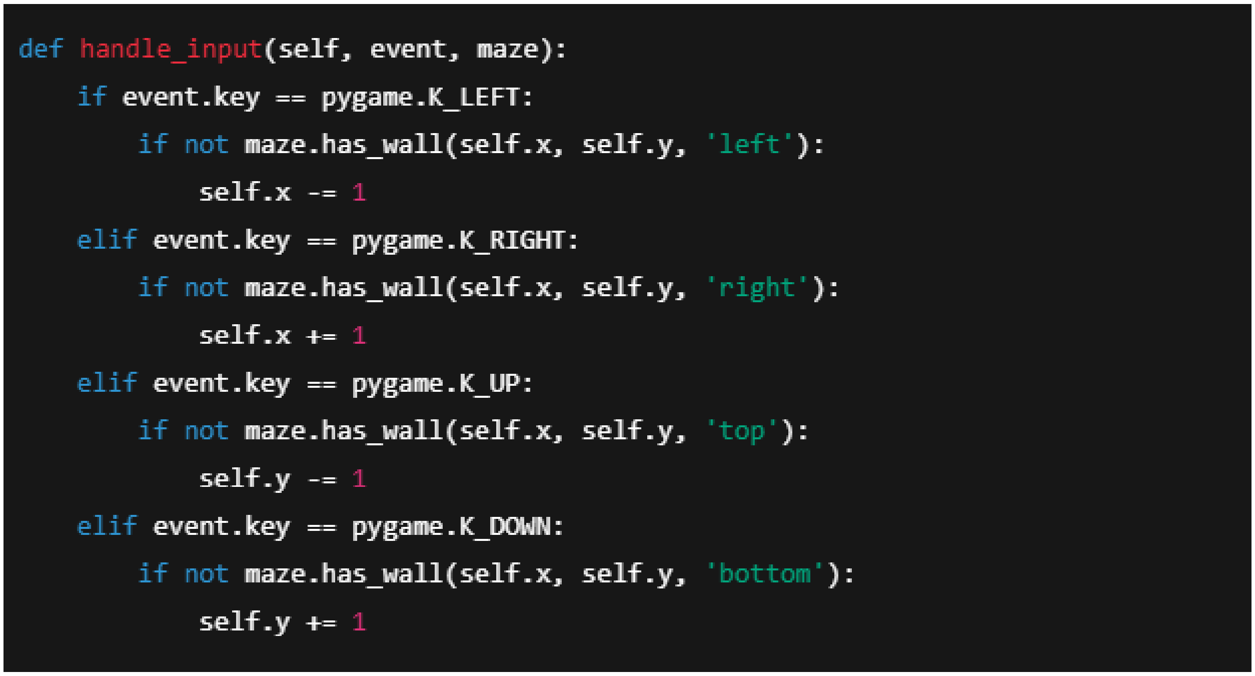
3. Player Movement

The player is controlled using the keyboard arrow keys or WASD. The game uses event-driven input handling where the OS captures key presses and sends them to the game loop. The movement is grid-based, meaning each key press moves the player one cell in the direction, unless blocked by a wall.

Movement Logic:

- Movement is restricted by walls.
- Position is represented in `(x, y)` coordinates.
- Movement direction is only allowed if there's no wall.

Code Snippet:



```python
def handle_input(self, event, maze):
    if event.key == pygame.K_LEFT:
        if not maze.has_wall(self.x, self.y, 'left'):
            self.x -= 1
    elif event.key == pygame.K_RIGHT:
        if not maze.has_wall(self.x, self.y, 'right'):
            self.x += 1
    elif event.key == pygame.K_UP:
        if not maze.has_wall(self.x, self.y, 'top'):
            self.y -= 1
    elif event.key == pygame.K_DOWN:
        if not maze.has_wall(self.x, self.y, 'bottom'):
            self.y += 1
```

4. Monster Movement

The monster automatically chases the player. It moves slightly slower than the player at first, but as it gets closer, its speed increases, making the game more intense.

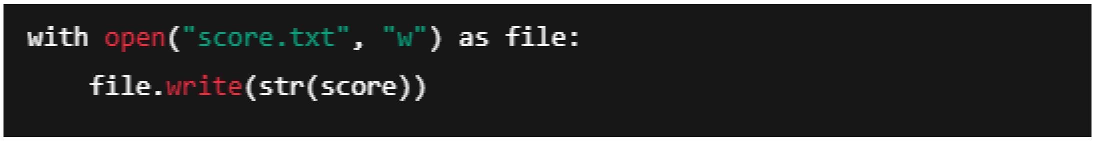
Movement Mechanics:

- The monster tracks the player’s position.
- It moves based on direction calculation (e.g., toward the shortest path).
- The speed increases based on distance from player.
- The movement uses the `update()` function which takes `dt` (delta time) as input for frame-based animation.

5. System Calls Used

1. File I/O (read/write)

The game saves the top score or game stats to a file.

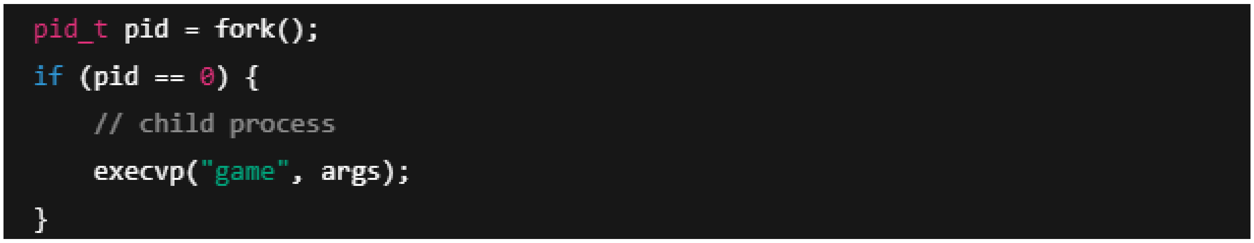


```
with open("score.txt", "w") as file:
    file.write(str(score))
```

System Call Used: `write()`, `open()`, `close()`

2. Process Control

When the game starts or ends, the system creates or terminates the game process.

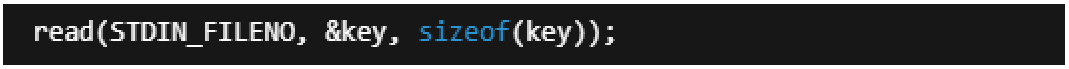


```
pid_t pid = fork();
if (pid == 0) {
    // child process
    execvp("game", args);
}
```

System Call Used: `fork()`, `exec()`, `exit()`

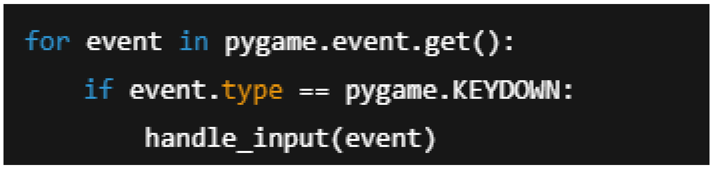
3. Input/Output

Keyboard inputs are captured using event listeners.



```
read(STDIN_FILENO, &key, sizeof(key));
```

In Python, this is done via:

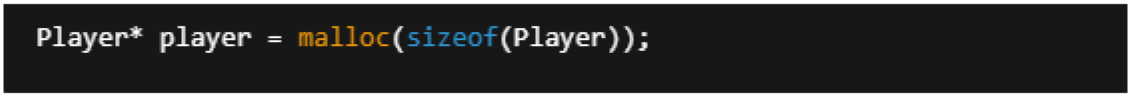


```
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN:
        handle_input(event)
```

System Call Used: `read()`

4. Memory Allocation

Objects like `Player`, `Monster`, and `Maze` are dynamically created.

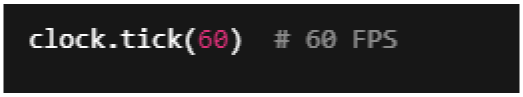


```
Player* player = malloc(sizeof(Player));
```

System Call Used: `brk()`, `mmap()`

5. Timer/Sleep

To control game speed and frame rate.



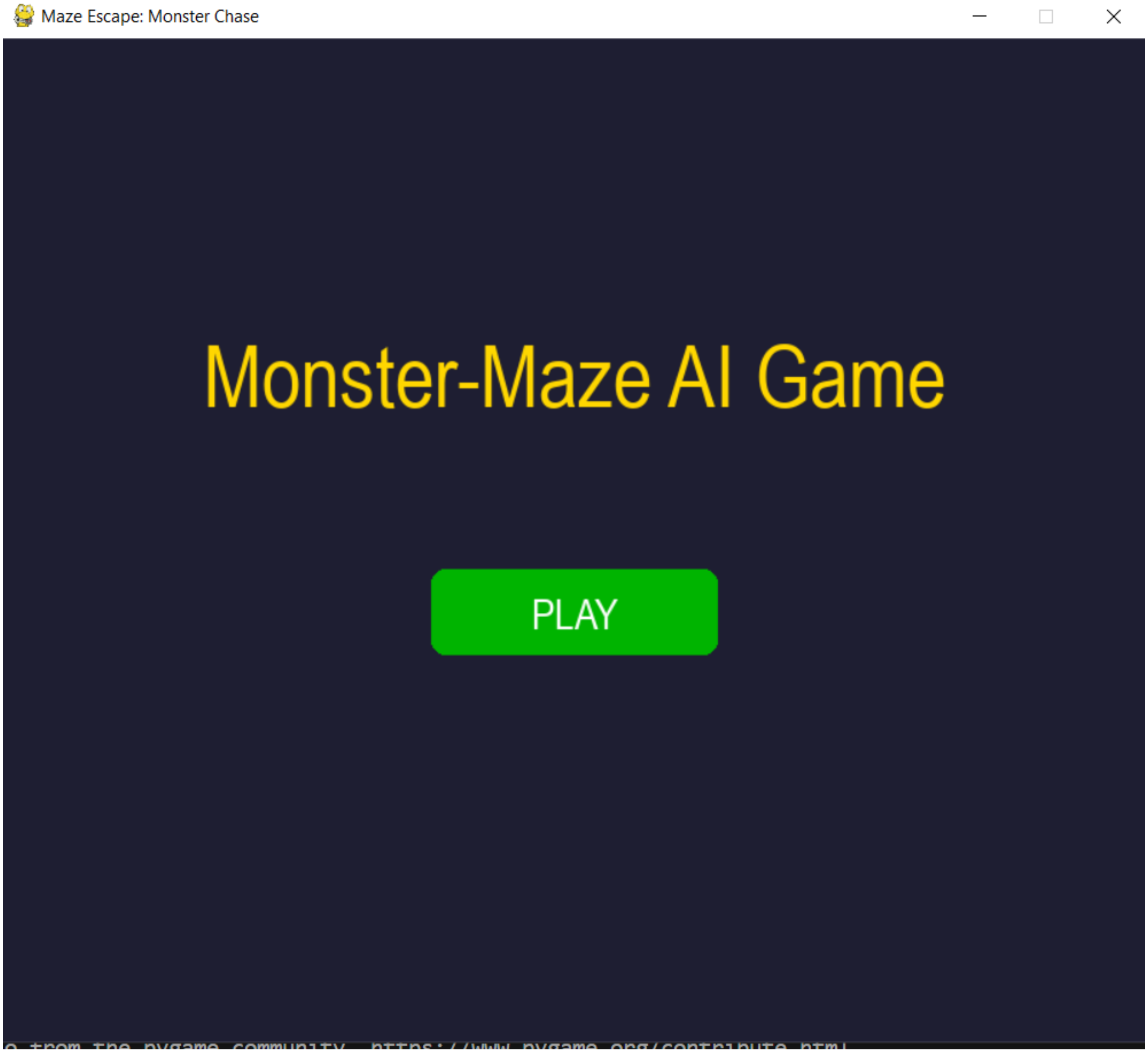
```
clock.tick(60)  # 60 FPS
```

System Call Used: `nanosleep()`, `alarm()`

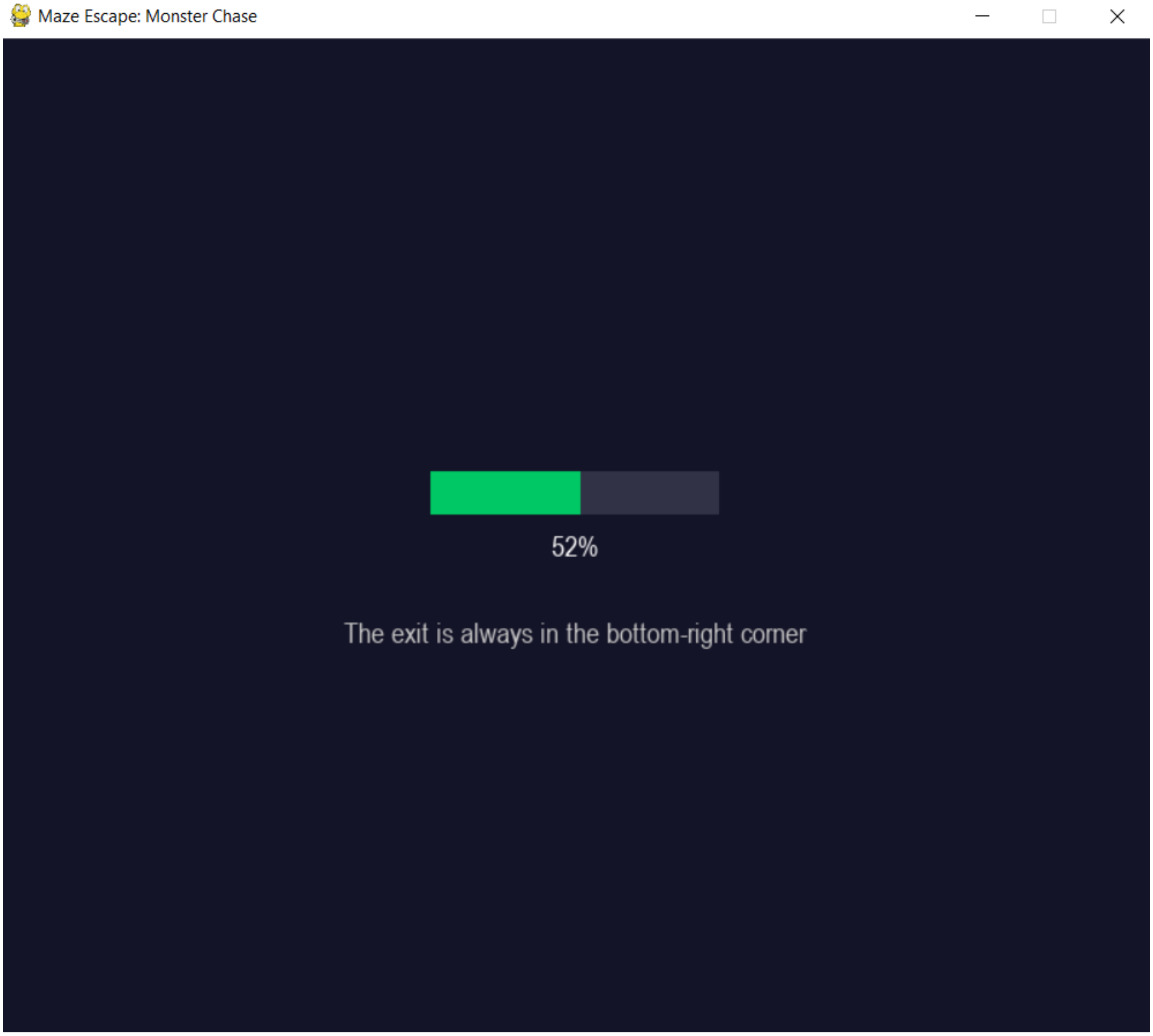
6. Challenges Faced

- Implementing smooth collision detection with maze walls.
- Calibrating monster speed dynamically without unbalancing gameplay.
- Ensuring responsive keyboard controls with real-time feedback.

Some Screen shot of the Game UI and screens:



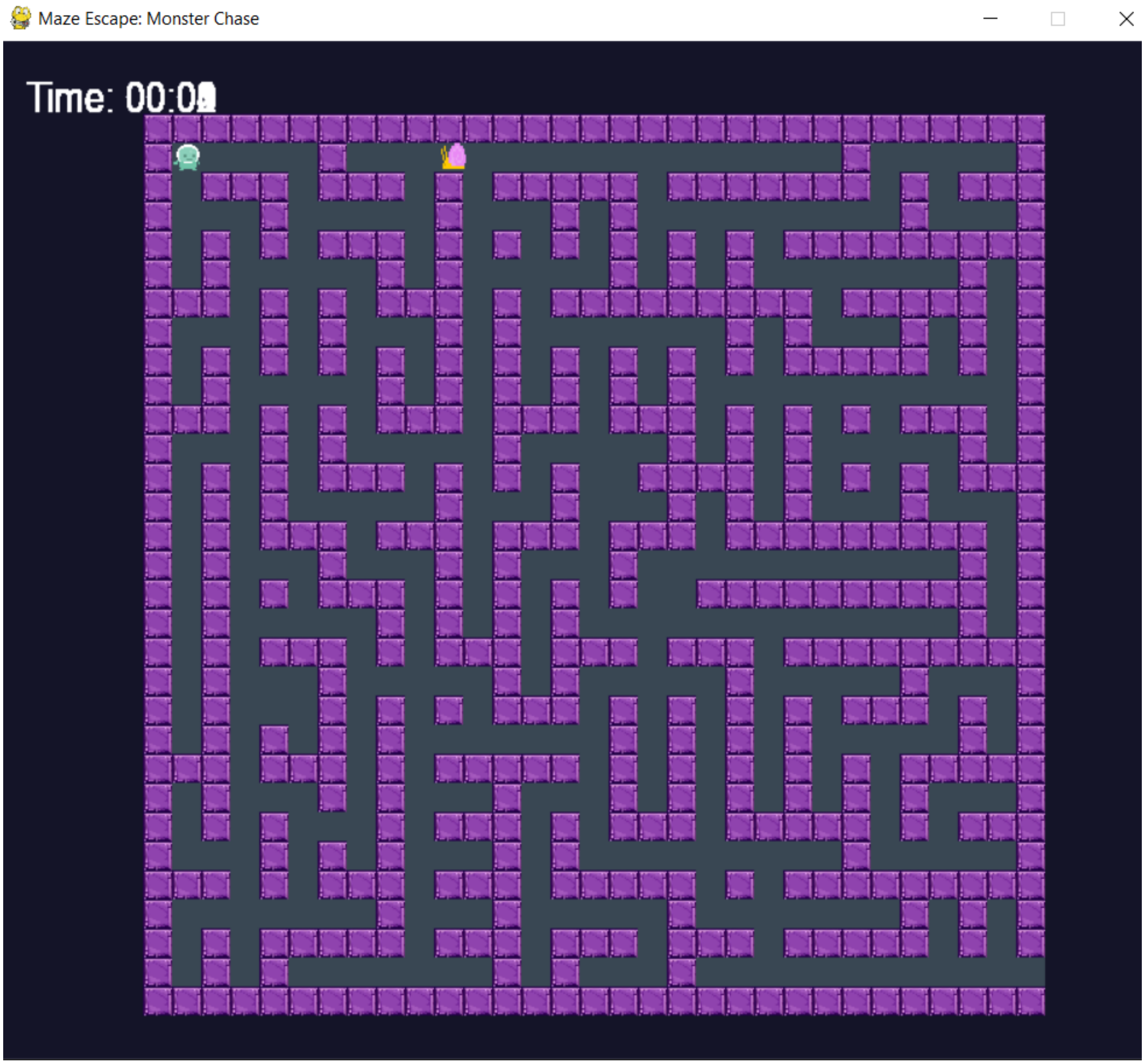
```
o from the pygame community. https://www.pygame.org/contribute.html
```

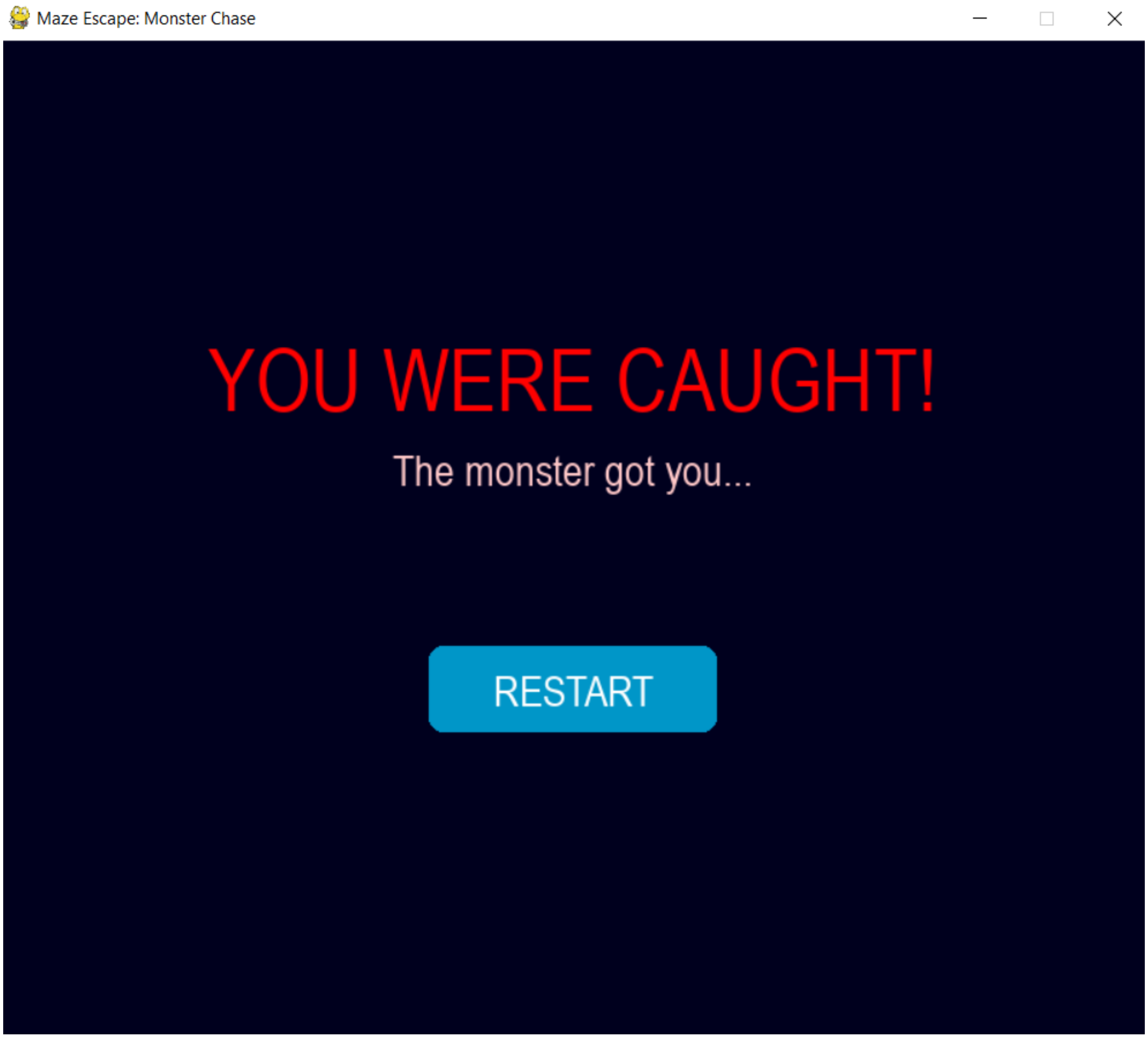


Maze Escape: Monster Chase

52%

The exit is always in the bottom-right corner


Maze Escape: Monster Chase
Time: 00:0



YOU WERE CAUGHT!

The monster got you...

7. Conclusion

Haunted Scape Maze successfully demonstrates how core operating system concepts like system calls, file I/O, and process control can be applied in an interactive and engaging way. By linking theory with practice, this project helped us understand the real-world relevance of OS functions and their behavior in dynamic applications like games.