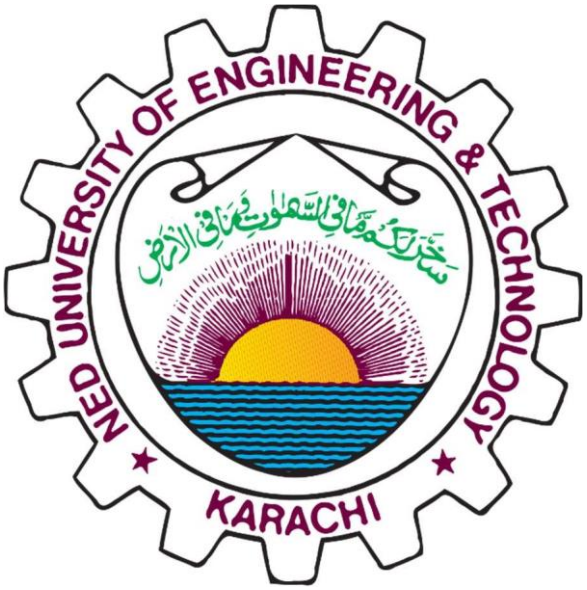
COMPLEX COMPUTING PROBLEM

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)

Contents

1. Introduction

Real-life Problem Inspiration

Set in a haunted environment, the player must escape a procedurally generated maze while being pursued by a monster with adaptive AI. The monster intelligently tracks the player's position and accelerates as it gets closer, creating a tense and immersive experience. The player must navigate using quick reflexes and strategic decisions, all while avoiding dead ends and staying ahead of the threat. This dynamic gameplay not only entertains but also demonstrates real-time system interactions and decision-making under pressure.

Real-life Problem Inspiration

The game mirrors real-life scenarios where AI must respond dynamically to user behavior in complex environments—such as robots navigating unknown terrain or security systems tracking intruders in real-time. The monster's pathfinding and proximity-based speed scaling showcase how intelligent systems can make real-time decisions to pursue or evade, similar to autonomous drones or surveillance bots adapting to human movement patterns.

Complexity of the Problem

Creating this game involved solving multiple layers of complexity: real-time input handling, adaptive AI for monster movement, and consistent frame-rate control. Ensuring smooth collision detection and balancing monster speed without making the game unfair required extensive testing and fine-tuning. Additionally, simulating OS-level behaviors like process control and dynamic memory allocation in a game environment added a challenging systems-level depth to the project.

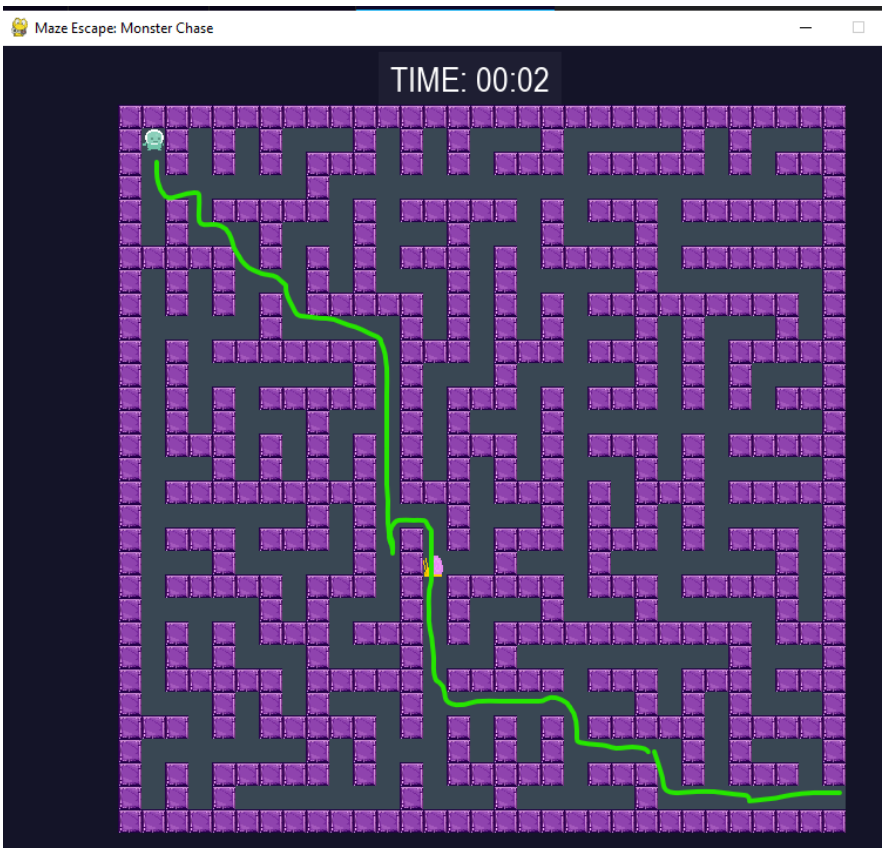
2. Key Features:

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

3. Maze Generation

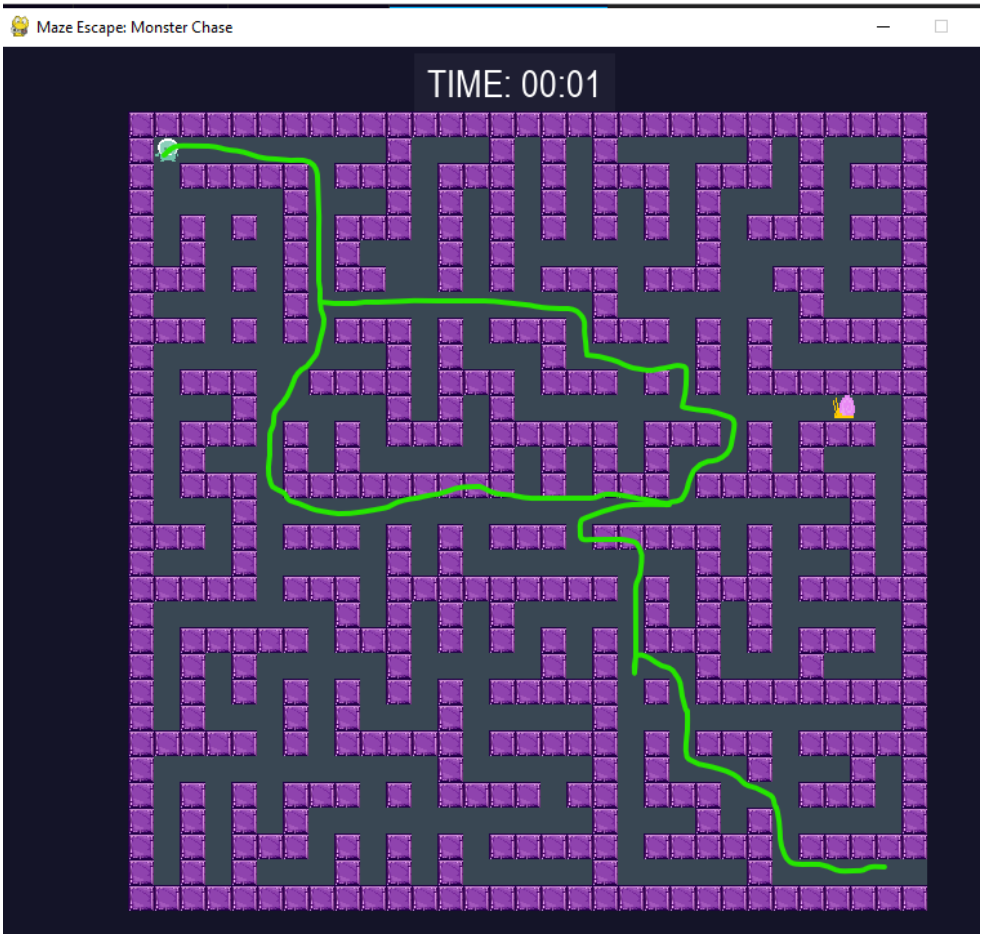
The maze was first generated using Prim's algorithm. It works by starting from a random cell, marking it as part of the maze, and then adding its neighboring walls to a list. At each step, a random wall is chosen from the list; if it divides a visited and unvisited cell, the wall is removed and the unvisited cell is added to the maze, this process repeats until all cells are connected.

The issue that we faced was Prims algorithm always generated single solution path, means there was no possibility that the player could escape at any point from the monster AI. So we decided to add the multiple paths logic once the whole maze was generated. The idea was to pick some number of walls randomly and then delete them making the multiple paths in between.



Prim's Algoithms Without Adding Multiple Paths (If you are divert from the exit path you end up at dead end)

Prim's Algorithm After adding multiple Paths (If you divert from the original path, there is possinlility



4. Player Movement

We trace the player's movement using event-driven input handling in Pygame, which listens for keyboard events such as arrow keys or WASD inputs. Each key press updates the player's position by moving one grid cell in the intended direction, provided there's no wall blocking the path. The player's current position is stored as (x, y) coordinates on the maze grid. Collision detection ensures the player cannot pass through walls, maintaining logical movement within the maze. These updates are continuously processed inside the game loop to provide real-time responsiveness and smooth navigation.

```
22      def handle_input(self, event, maze):
23          """Call this in your event loop (not update())"""
24          if event.type == pygame.KEYDOWN:
25              if self.move_cooldown <= 0:
26                  moved = False
27                  if event.key == pygame.K_LEFT:
28                      moved = self._try_move(-1, 0, maze)
29                  elif event.key == pygame.K_RIGHT:
30                      moved = self._try_move(1, 0, maze)
31                  elif event.key == pygame.K_UP:
32                      moved = self._try_move(0, -1, maze)
33                  elif event.key == pygame.K_DOWN:
34                      moved = self._try_move(0, 1, maze)
35
36                  if moved:
37                      self.move_cooldown = self.move_delay
38
```

5. AI Based Monster Tracking Player Logic

The monster uses a **Finite State Machine (FSM)** to manage its behavior, switching between different AI states based on the player's position. Initially, the monster is in an **Idle** state, where it moves slowly and randomly. When the player enters a certain detection radius, the monster transitions to an **Alert** state, actively scanning the maze and aligning itself toward the player's location. If the player is detected within a closer range, the monster enters the **Chase** state, using directional logic to move along the shortest path. Finally, when the player is extremely close, the monster enters a **Frenzy** state, increasing its speed drastically to maximize the chance of capture.

To calculate the distance, the monster uses the **Euclidean Distance Formula** based on both entities' coordinate.

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This allows the AI monster to measure proximity and decide when to escalate its behavior, ensuring dynamic and adaptive pursuit.

6. Game Architecture and Modules

The game follows a modular structure, with each component organized into specific files to handle distinct responsibilities.

utils/maze_generator.py

Generates the maze using Prim's Algorithm and handles path creation. Also includes logic to delete random walls and introduce multiple paths for better gameplay.

utils/monster.py

Defines the Monster class and its FSM-based AI behavior. Controls monster states like Idle, Alert, Chase, and Frenzy, and calculates movement based on player distance.

utils/player.py

Contains the Player class and handles keyboard-driven movement. Manages collision detection and updates player position in the maze.

main.py

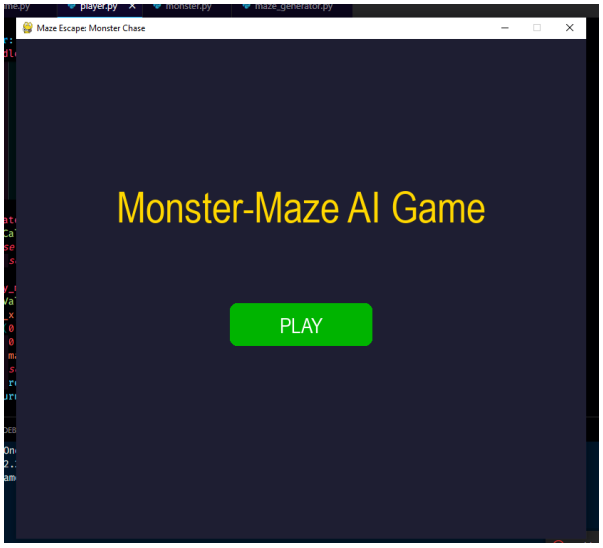
Acts as the game's entry point. Starts the program and calls necessary functions to launch the game.

game.py

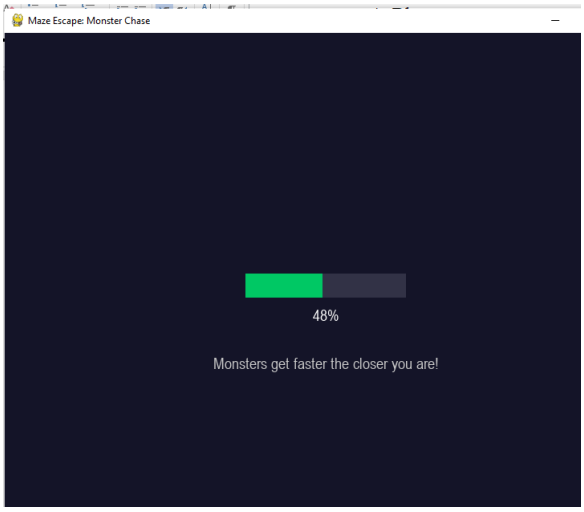
The core of the game where all objects are created and initialized. Handles game screens, event loop, rendering, and updates for each frame. Also the objects of each above classes are created here as well maintaining clean structure of the project.

7. Some Game play screens

Screen 1

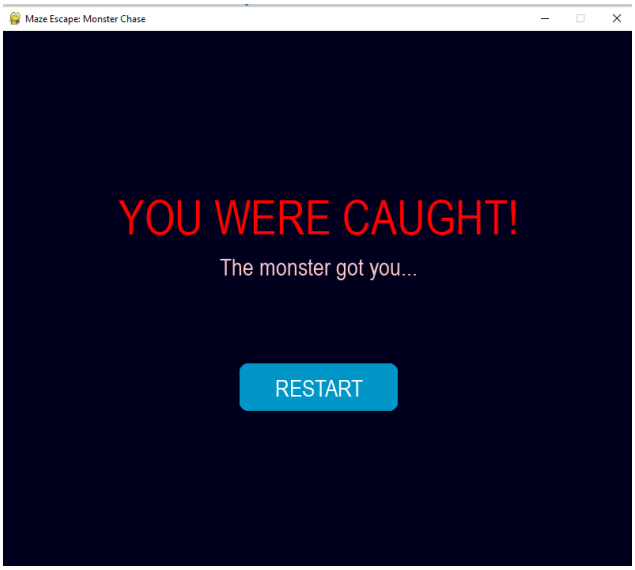


screen 2



Screen 3

screen 4



8. Limitations and Future Scope

Haunted Scape Maze lays a strong foundation for AI-driven gameplay but remains a basic prototype in its current form. The existing monster AI is effective but can be significantly improved using advanced algorithms like A* for smarter pathfinding and decision-making. Introducing more complex AI behaviors and emotional states can enhance the game's realism and challenge.

With further development, the game can evolve into a full-fledged AI-based horror survival title. Future versions could feature multiple monsters with distinct AI personalities, adaptive strategies, and cooperative hunting behaviors. Combined with power-ups, player abilities, and level progression, the enhanced AI would drive a dynamic, unpredictable, and immersive gaming experience fit for a commercial release.

9. Conclusion

The *Haunted Scape Maze* project highlights the application of **Artificial Intelligence** in creating dynamic and immersive gameplay. Central to this experience is the **Finite State Machine (FSM)**-based monster logic, which adapts the monster's behavior in real time based on the player's proximity. By transitioning through states such as *Idle*, *Alert*, *Chase*, and *Frenzy*, the monster showcases intelligent decision-making and reactive movement—hallmarks of modern AI systems. This approach mirrors real-world use cases where AI must adjust strategies based on changing inputs, such as surveillance bots or autonomous agents. The project effectively demonstrates how FSM and distance-based state transitions can form the backbone of engaging, adaptive game AI.

Moreover, the core idea of this project can translate into real-life industrial solutions where autonomous systems—like drones, security robots, or smart surveillance—must intelligently track, follow, or evade moving targets. With further enhancement, this prototype can serve as a model for AI-driven pathfinding and threat-response systems in fields such as robotics, defense, and smart infrastructure.