Game description

The maze game that we have created is a simple Maze solving Game. Its more like a puzzle a player needs to solve in order to reach to the end of the level. For completing the level, the player must retrieve a key that's opens the lock to the next level. Upon clearing the level, the next level will be automatically created using maze creation algorithms.

The starting of the game has simple levels. But the later levels have obstacles which makes the more difficult. Since the obstacles only change their state when the player moves, the player must watch for the patterns to reach the destination in the shortest number of steps. If the player runs into any obstacles (traps or guards), the player will restart at the first position in that maze and lose one life.

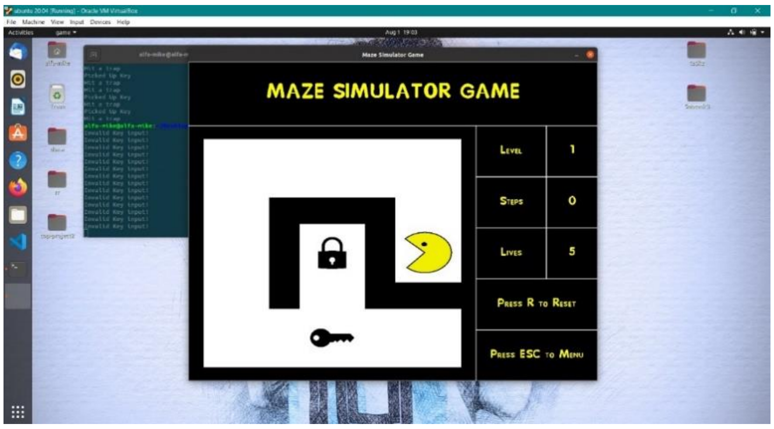
When all lives have been lost, the game is over, and the only options are to quit or return to the front end menu and try to get even further.

Overview

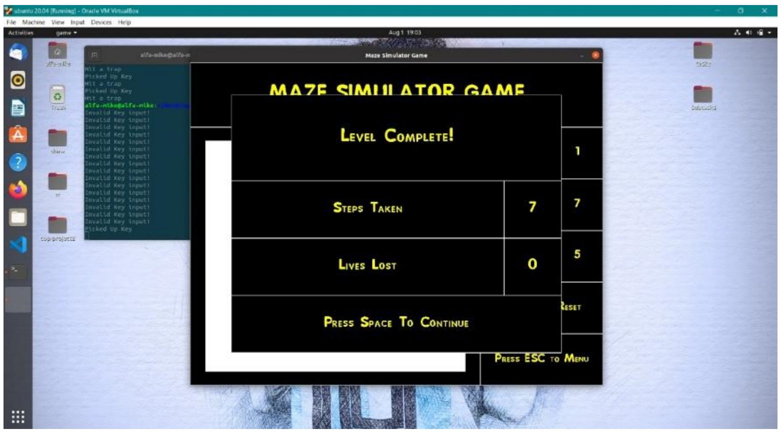
Game Rules

- The game is a single – player 2-D Maze game in which the player has a specific no. of lives in which the player has to reach as much level as he can reach.
- The Game starts with the game logo showing on the screen.

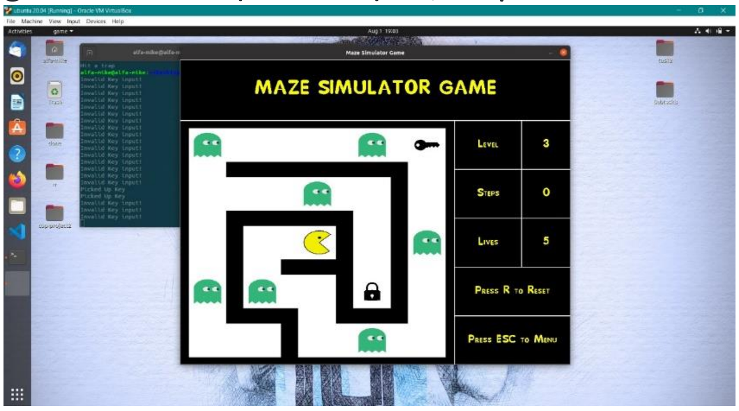
- We need to press any key to start the game.
- The game level - 0 which is the first level of the game (typically the basic one with least no. of rooms)



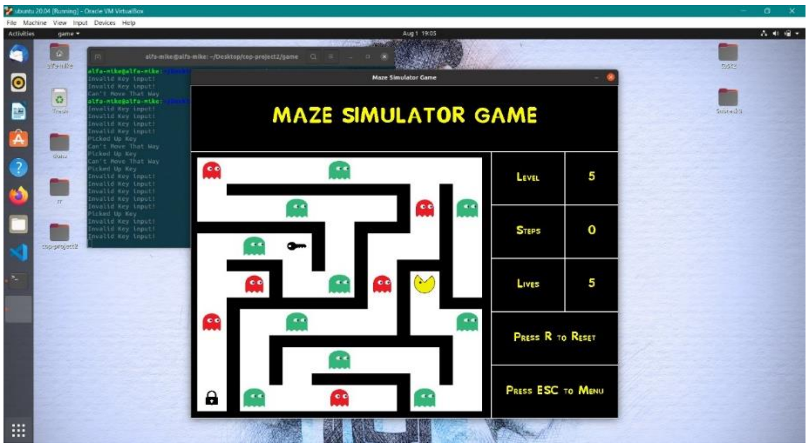
- To complete a level, the player has to first collect the key and go the Door (Lock) which is the door for the next level.



- On completion of a level, new Maze is automatically created based on level number and a new window pops up.
- On completing a level, the game shows details like, steps taken to reach Door, life lost in particular level.
- After a threshold level, more difficulty is incorporated by introducing new game features (obstacle) viz, Traps and Guards.

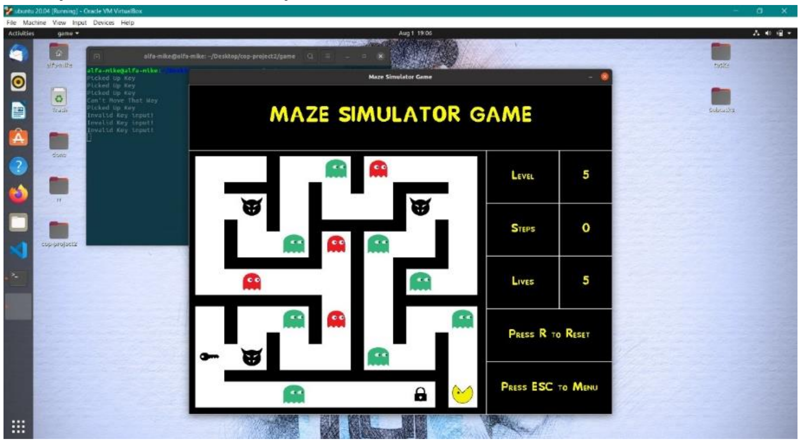


- The Traps are static obstacle and have two states – active(red colour) and inactive(green colour). If a player encounters an active Trap, he lost one life and returned to the starting position.



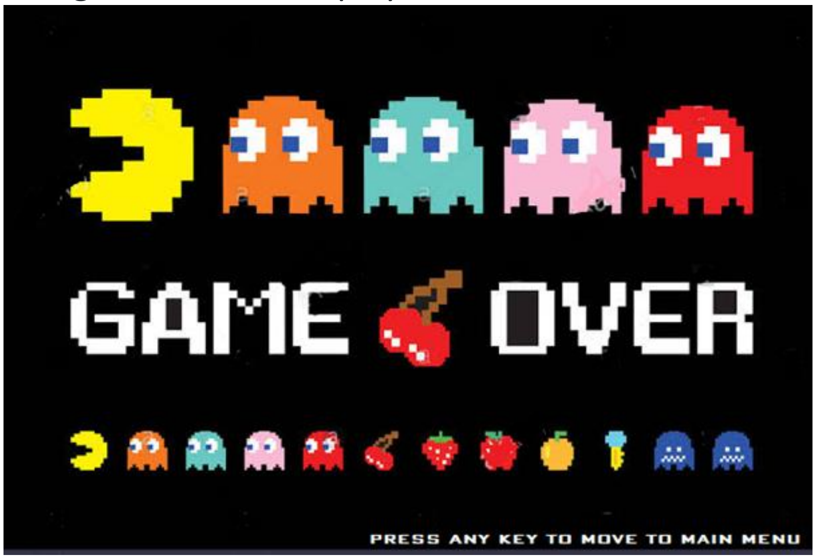
```
alfa-mike@alfa-mike:
Invalid Key input!
Invalid Key input!
Can't Move That Way
alfa-mike@alfa-mike:
Invalid Key input!
Invalid Key input!
Invalid Key input!
Invalid Key input!
Picked Up Key
Can't Move That Way
Picked Up Key
Can't Move That Way
Picked Up Key
Invalid Key input!
Invalid Key input!
Invalid Key input!
Invalid Key input!
Invalid Key input!
Picked Up Key
Invalid Key input!
Invalid Key input!
Invalid Key input!
```

- The Guards are moving obstacles and moves alternatively in three connected rooms adjacent to their spawn rooms. On hitting any of the guard, player loses one life and returned to starting position.
- Player can reset a particular level at the cost of a life.



```
alfa-mike@alfa-mike:
Picked Up Key
Picked Up Key
Picked Up Key
Can't Move That Way
Picked Up Key
Invalid Key input!
Invalid Key input!
Invalid Key input!
```

- The game ends when player loses all his lives.



Features

- Front End Menu
    - Goes into the gameplay
    - Exits the game
- Interactive Player
    - Movement based off arrow keys
    - Begins with a number of lives and loses them when you:
        - Reset the maze
        - Hit any obstacle
    - When the player is out of lives, the game is over and the game returns to the Front End
- Procedural Game levels
    - Procedurally generates mazes that become increasingly larger
    - Spawns obstacles in randomized locations in the level
    - Must find key to get to next level

Maze-Creation

1. The Maze is created using the Back-tracking algorithm which is similar to the Depth-First Search.
2. The Maze class creates the actual Maze for the Game. It begins the process by calling a function createrooms() which creates a grid of rooms, giving each a unique position and storing all of them in a vector.
3. The CarveMaze() actually goes for to create the Maze. It selects one of the rooms at random and stores it in a vector current_room and uses backtracking to carve out the Maze.
4. It then selects on the adjacent rooms to the current_ room and removes the wall between them, also removing the previous room from the vector.
5. It continues this until it encounters a room with no adjacent_rooms, at which it will pop off the top element of the vector, and check if the previous room in the path had any adjacent_rooms. It continues when all the rooms are in the Maze.

6. As the level goes on increasing, the difficulty of the maze also increases. The difficulty as in the no. of rooms increases gradually which increases the difficulty to solve the Maze.

SDL libraries Included

SDL2 was used to actually display the game on screen. To display images and text onto the screen, I needed to bring in other SDL2 resources. The SDL2 libraries implemented here are :

- SDL2
- SDL2_main
- SDL2_image
- SDL2_ttf
- SDL2_mixer

Note :

We tried to implement sockets and sound to make our game multiplayer and more interesting. But by the end of stipulated time, we could not properly process the implementation, so we decided to skip for those parts and tried to make the single player game more appealing.

Our vision was to make of a single player game and then convert it into 2-player by using sockets but we didn't have proper idea about using of UDP-Sockets at that time. After making the game, when we researched about the multiplayer scene, we also went through the sdl3 provided in the course website(which is the implementation of a 2-player gun game). But from last 15-20 days we were constantly messed up with our intern tests, so we couldn't focus on the project.

And when we received the mail regarding the deadline, we worked on it but couldn't complete it within the given time, so we decided to skip the multiplayer part as it wasn't complete. So we didn't commit those changes in the GitHub. And since we had applied for I grade, we had no friends who could properly direct us through the issue.