ST. XAVIER'S COLLEGE

(Affiliated to Tribhuvan University)

Maitighar, Kathmandu

PROJECT REPORT
ON
OBJECT FINDER

For the partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Information Technology awarded by Tribhuvan University

Under the supervision of

Er. Bishnu K.C.
Lecturer/Supervisor
Department of Computer Science

Submitted By:

Raman Shakya (T.U Roll No: 5-2-282-97-2022)
Rayan Karki (T.U Roll No: 5-2-282-98-2022)
Saugat Thapa (T.U Roll No: 5-2-282-103-2022)
Srijal Manandhar (T.U Roll No: 5-2-282-106-2022)

Submitted To:

ST. XAVIER'S COLLEGE
Department of Computer Science
Maitighar, Kathmandu,
Nepal

ST. XAVIER'S COLLEGE
MAITIGHAR, KATHMANDU, NEPAL
Post Box :7437
Contact: 4221365,4244636
Email: ktm@sxc.edu.np

सेन्ट जेभियर्स कलेज
माईतीघर, काठमाडौं, नेपाल
पो.ब.नं. : ७४३७
फोन : ४२२१३६५,४२४४६३६
ईमेल : ktm@sxc.edu.np

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the Department of Computer Science for acceptance, a project proposal entitled **"Object Finder"** submitted by **Raman Shakya, Rayan Karki, Saugat Thapa** and **Srijal Manandhar** for the partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Information Technology awarded by Tribhuvan University.

……..
Er. Bishnu K.C.
Lecturer
Department of Computer Science
St. Xavier's College

ABSTRACT

"Object Finder" is a computer graphics project which demonstrates the generation of interactive 3D mazes using OpenGL API for rendering. The project utilizes Depth First Search (DFS) Algorithm to dynamically create maze structures, offering users an immersive environment to navigate around. Controlled through a first-person perspective with standard keyboard inputs (WASD for movement, QE for camera angle adjustment), users explore the maze to locate randomly spawned objects.

The project's key features include dynamic maze generation, real-time rendering with OpenGL, first-person camera controls, and random object spawning. Beyond entertainment, this project serves as an educational tool for those interested in computer graphics and game development, providing a blend of maze algorithm and interactive gameplay to showcase modern graphics programming techniques.

This project has potential applications in gaming, virtual reality, 3D rendering, and computer graphics. It is also possible to add more optimized algorithm for maze generation and additional features to enhance its capability.

Keywords: *Object Finder, OpenGL, Computer Graphics, 3D Rendering, Depth First Search (DFS) Algorithm, Maze Generation*

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Background

In the world of computer graphics and game design, creating immersive virtual environments has long fascinated both developers and users. Mazes, with their intricate paths, offer a unique mix of challenge and intrigue that has captured the public's imagination for generations. Traditionally, maze generation and rendering have been core topics in computer science, with researchers seeking efficient ways to create complex layouts and display them realistically.

Against this backdrop, the "Object Finder" project aims to develop an interactive 3D maze experience using OpenGL and the DFS algorithm. This project explores the intersection of algorithmic ingenuity and creative expression, showcasing the potential of computer graphics in gaming, education, and beyond.

1.2 Problem Statement

Despite the longstanding interest in maze generation and rendering, there remains a gap in the availability of interactive 3D maze experiences that seamlessly integrate modern graphics programming techniques with such algorithms.

Moreover, the educational potential of such projects is often underexplored, with few resources providing insights into the principles of computer graphics and game development through practical, hands-on experiences. Therefore, the problem at hand is to develop a comprehensive solution that addresses these challenges, culminating in the creation of an interactive 3D maze experience that not only showcases the potential of computer graphics in gaming and education but also serves as a platform for exploring the intricacies of maze algorithms and rendering techniques in a dynamic and immersive manner.

1.3 Features

In the realm of computer graphics and game development, the "Object Finder" project stands as an ambitious endeavor to merge cutting-edge rendering techniques. Here are some key features of the "Object Finder" project:

- Dynamic maze generation using the Depth-First Search (DFS) algorithm
- Real-time rendering with OpenGL for immersive visualization
- Interactive first-person gameplay with intuitive keyboard controls (WASD for movement, QE for camera angle adjustment)
- Random spawning of objects within the maze to enhance challenge and exploration
- Educational value as a tool for learning computer graphics and game development principles
- Customizable settings for maze size, difficulty level, and rendering quality
- Modular architecture for easy expansion and addition of new features and enhancements.

CHAPTER 2

PROJECT DISCUSSION

2.1 Objectives

The objective of this project is to develop an interactive 3D maze experience utilizing OpenGL for rendering and the Depth-First Search (DFS) algorithm for maze generation. The primary aim is to create a dynamic and immersive environment where users can navigate through procedurally generated mazes while seeking randomly spawned objects. Additionally, the project seeks to serve as an educational tool, providing insights into computer graphics and game development principles through practical implementation.

2.2 Purpose

The primary purpose of this project is to deliver an engaging and immersive entertainment experience through a maze-solving game. Players are tasked with navigating through intricate mazes, employing strategic thinking and problem-solving skills to locate a hidden object within the maze.

2.3 Implementation

Implementing a maze-solving game involves several key components, including generating the maze, rendering it visually, controlling player movement, and detecting collisions with walls and the hidden object.

CHAPTER 3

METHODOLOGY

3.1 Overview

The methodology of this project involves the development of an interactive 3D maze experience using a combination of DFS maze generation algorithm, 3D rendering computer graphics technique, and player movement controls. The project aims to create a dynamic and immersive environment where users can navigate through procedurally generated mazes while interacting with objects. The overall process consists of components namely Block, Object, Player, World where maze generation and maze management falls under World, 3D rendering of object and its logic for random spawn, design of the blocks falls under Block and Object, player movement and controls falls under Player, and in main file all interactive elements are integrated together.

3.2 Software Used

For the implementation of the project, several software tools and libraries are utilized. The primary programming language for development is **C++**, chosen for its versatility and performance. Additionally, the project makes use of the **OpenGL** library for graphics rendering, providing the necessary functionalities for real-time rendering of the 3D maze environment. Other supporting libraries include **GLUT** for window and input management. **VS Code** or **Dev C++** was chosen as code editor.

GitHub was used for collaboration and code sharing between the team members and contribute everyone's source code in real time environment and **Discord** was used as the communication medium.

3.3 Maze Generation and Map

The Depth First Search (DFS) back tracing algorithm is used for maze generation. An object is placed randomly inside the maze, which if the player comes in contact with, upgrades the player to the next level. A larger random maze is created with each level increment, and objects are placed randomly likewise. The maze's two-dimensional representation is concurrently mapped in the terminal window as the player progresses through each level.

A method *resetLevel* initializes the game level by clearing the existing map and objects, creating new instances and then *generateLevel* method recursively constructs the game map by randomly placing passages within a grid ‘#’ and utilizing this *drawLevel* creates the whole map.

Pseudo Code

```
void generateLevel(char grid, int i, int j) {
        if (i==currentLevel-2 && j==currentLevel-2) return;

        orientation = ((0, -1), (0, 1), (-1, 0), (1, 0)];
        orientation.shuffle();

        int iter;
        for ((dirX, dirY) of orientation) {
                int destI = i+2*dirX, destJ = j+2*dirY;
                if (destI > 0 && destJ > 0 &&
                    destI < size && destJ < size &&
                    grid[destI][destJ]=='#')
                {
                        grid[i+dirX][j+dirY] = ' ';
                        grid[destI][destJ] = ' ';
                        generateLevel(grid, destI, destJ);
                }
        }
}
```

3.4 3D Rendering

The functions *set3Dmode* and *render* are created for rendering the game environment in three dimensions. It adjusts the OpenGL settings to create a realistic 3D view of the game world. By setting the appropriate parameters, it ensures that objects in the scene appear correctly relative to each other and the viewer's perspective. This function enables depth testing, which allows objects closer to the viewer to occlude objects that are farther away, enhancing the visual realism of the game. Additionally, it establishes a perspective projection, simulating the way objects appear smaller as they move away from the viewer.

All the blocks like walls or obstacle are rendered by definition of its size, position and vertices.

3.5 Collision detection

A function *correctPlayer* is used to update the player's collision with the walls or interactions with the collectable object. If the player collides with collectable object, it triggers the level advancement. However, for each adjacent cell around the player's current position, this function also checks if it's a wall ('#') and if it's the case it adjusts the player's position to prevent them from moving through the wall. This mechanism ensures that player cannot move through the walls.

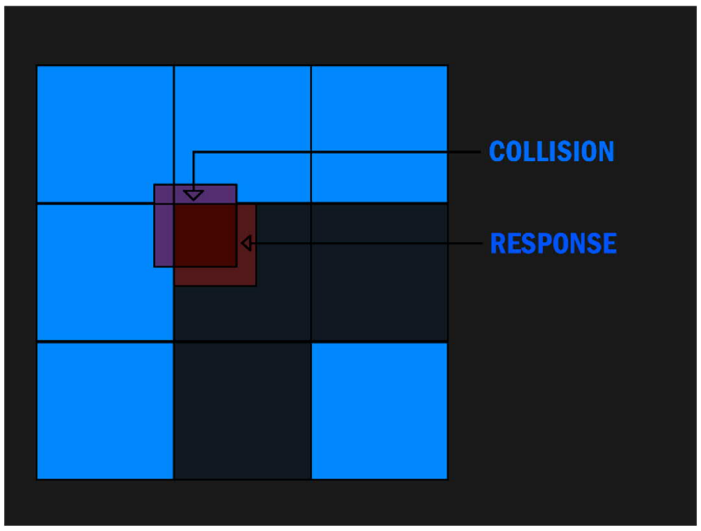

Fig 3.1: Collision Detection

Pseudo Code

```
void correctPlayer() {
        int posx = round(playerX/WIDTH), posz = round(playerZ/WIDTH);
        if (grid[posx][posz]=='o') nextLevel();

        int upBlock = grid [posx-1][posz] == '#',
            downBlock = grid [posx+1][posz] == '#',
            leftBlock = grid [posx][posz+1] == '#',
            rightBlock = grid [posx][posz-1] == '#';

        if (upBlock && playerX < (posx-width/2)*WIDTH)
                playerX = (posx-width/2)*WIDTH;
        if (downBlock && playerX > (posx+width/2)*WIDTH)
                playerX = (posx+width/2)*WIDTH;
        if (leftBlock && playerZ > (posz+height/2)*WIDTH)
                playerZ = (posz+height/2)*WIDTH;
        if (rightBlock && playerZ < (posz-height/2)*WIDTH)
                playerZ = (posz-height/2)*WIDTH;
}
```

3.6 Player Movement and Perspective View

For the purpose of basic player movement and camera perspective in virtual world a player header was defined which encompasses some functions like *moveFront, moveBack, moveLeft, moveRight, lookLeft, lookRight*.

The *WASD* keys are employed for basic player movement control, while the *QE* keys manages camera direction. The private variable of class stored all data including player's position, direction and other values necessary for movement and camera rotation.

3.7 State Machine

The game implements a state machine to manage the states of game, namely *GAME*, *MAIN_MENU*, *PAUSE_SCREEN* and *CONTROL_SCREEN* each represented by an integer value. At the heart of this state machine is the state variable which keeps track of the current state of the game. During the initialization of game, initial state is set to Main Menu. The *key* function handles keyboard inputs for different game states: in the main menu, it responds to 'p' for play, 'i' for controls, and 'e' for exit. The *draw* function renders the screen based on the current state, clearing it and setting the appropriate mode before calling the relevant drawing function or rendering the game world.

States Defined

```
struct State {
        int MAIN_MENU = 0;
        int GAME = 1;
        int PAUSE_SCREEN = 2;
        int CONTROL_SCREEN = 3;
} states;
```

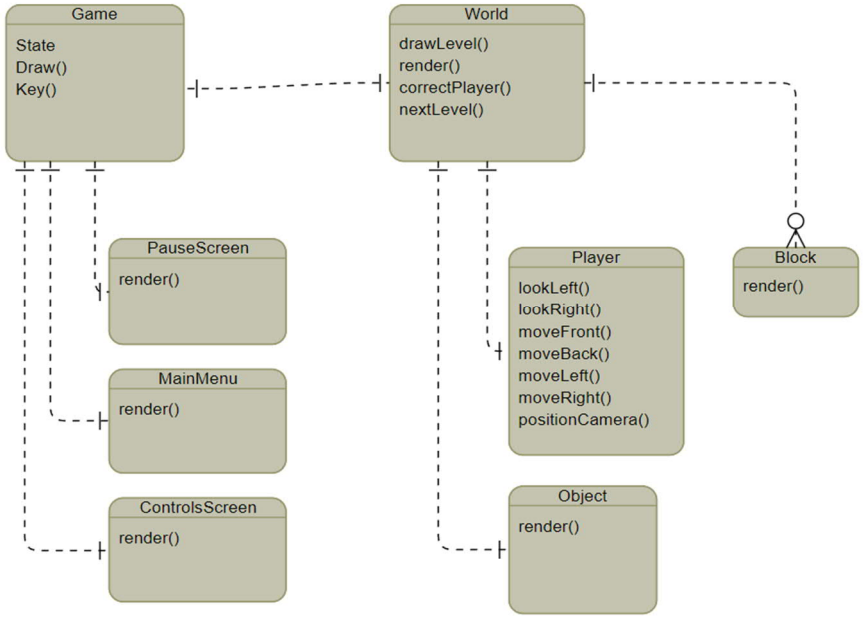

Fig 3.2: Entity Relationship Diagram

CHAPTER 4

OBSERVATIONS

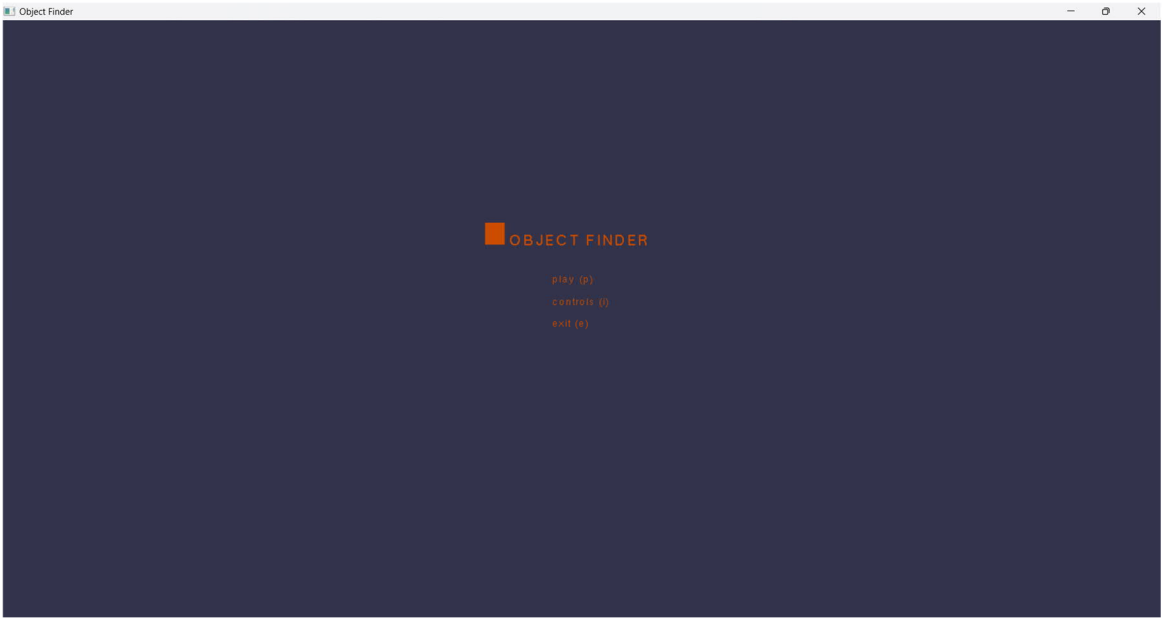

Fig 4.1: Main Menu

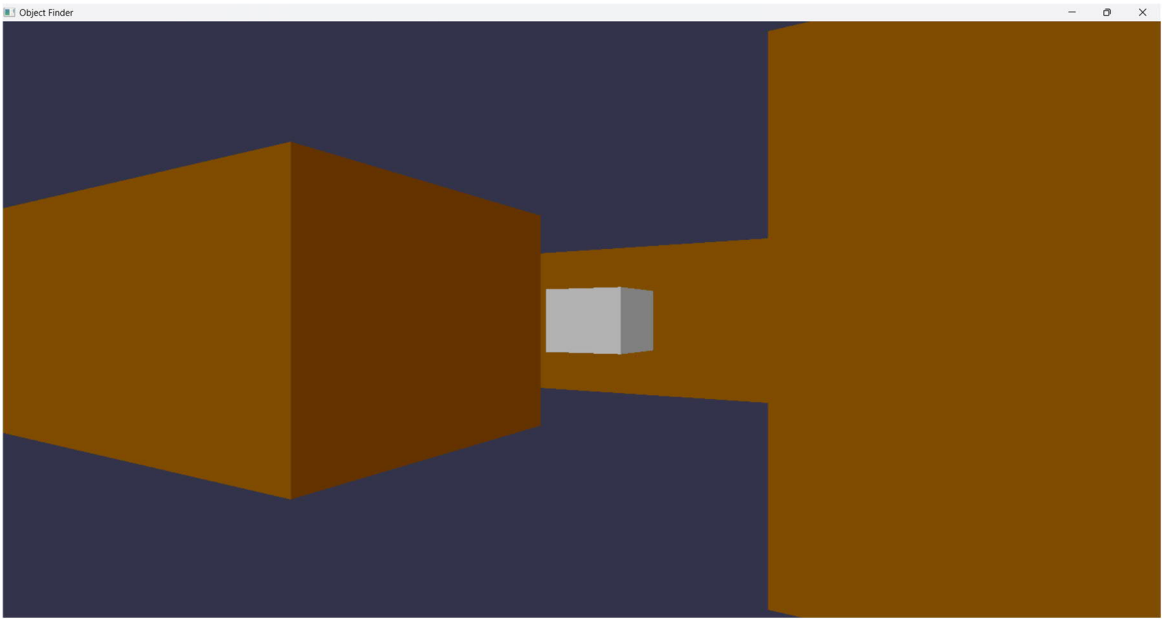

Fig 4.2: Collectable Object

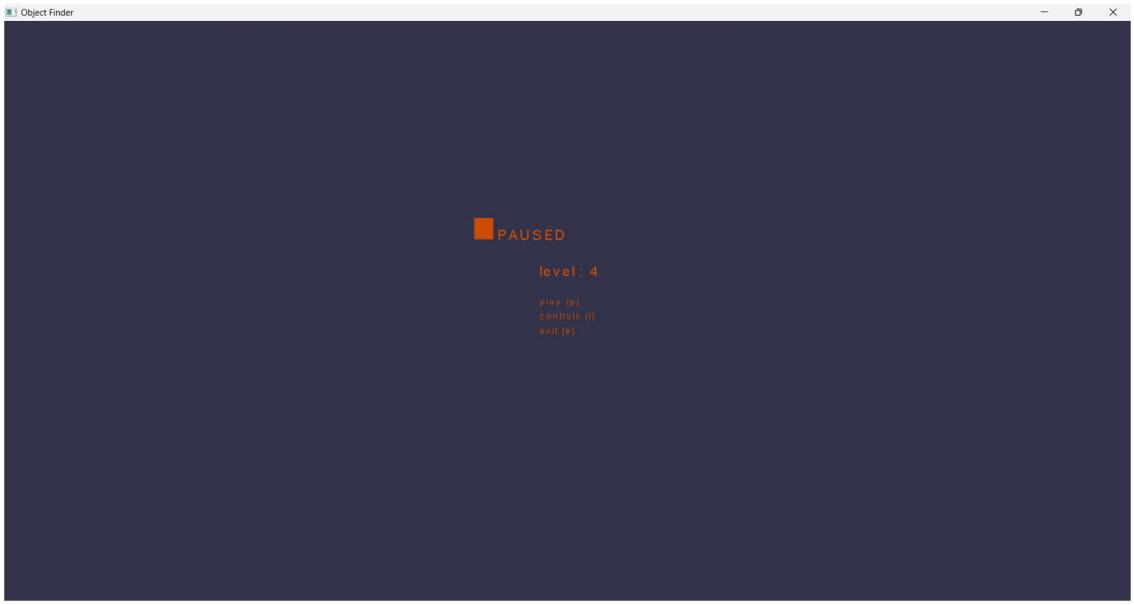

Fig 4.3: Paused Menu

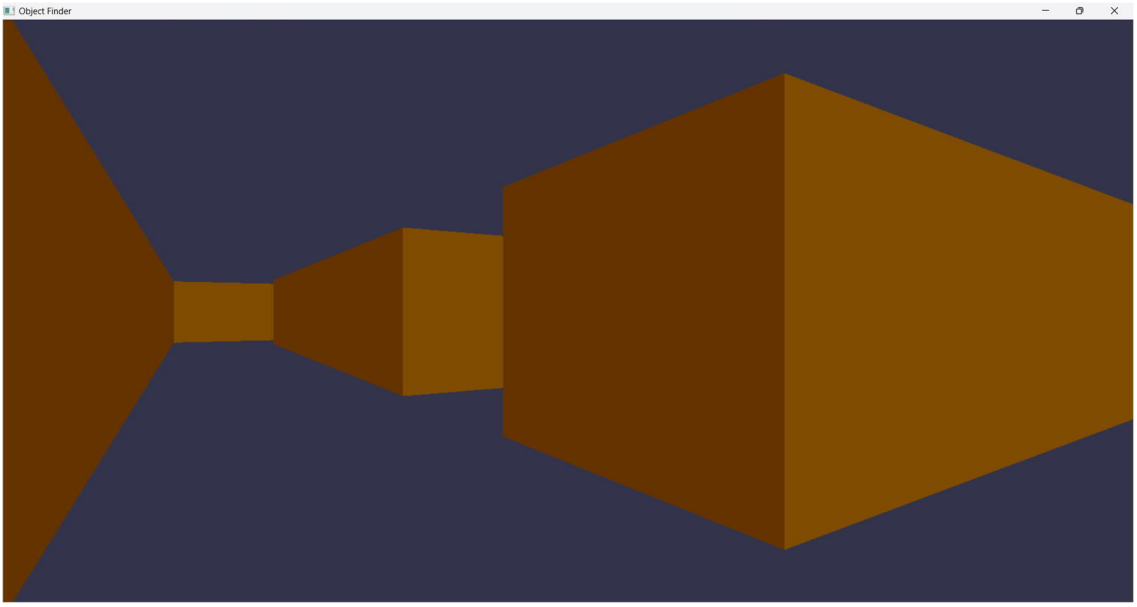

Fig 4.4: Game World

Fig 4.5: Controls Menu

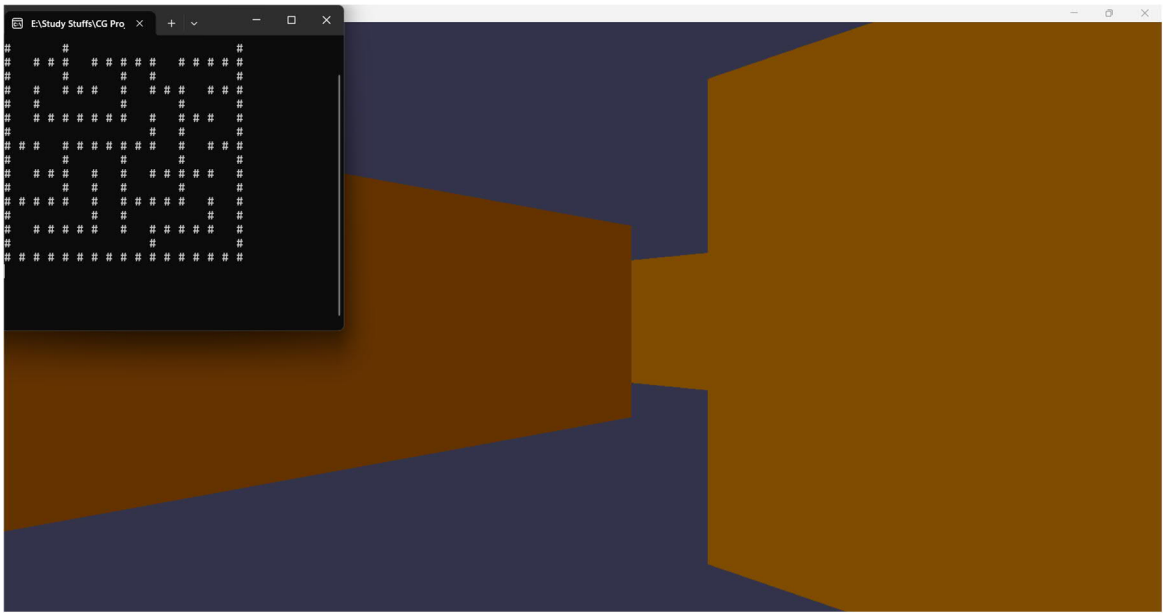

Fig 4.6: Maze Generation and Mapping

CHAPTER 5

CONCLUSION AND RECOMMENDATION

The "Object Finder" project endeavors to create an interactive 3D maze experience that amalgamates cutting-edge rendering techniques with algorithmic maze generation. This project serves as a testament to the potential of computer graphics in both gaming and education, offering an immersive environment where users can navigate procedurally generated mazes while honing their problem-solving skills. By harnessing the power of OpenGL for real-time rendering and employing the Depth-First Search algorithm for maze generation, the project showcases the seamless integration of technology and creativity.

Based on experience after playing this game, the following recommendation and suggestion was made:

1. **Educational Integration:** It is possible to deepen educational elements within the game, offering interactive tutorials and supplementary materials to elucidate computer graphics and game development principles.
2. **Enhanced Interactivity:** The gameplay can be expanded with additional challenges, puzzles, and dynamic obstacles, fostering strategic thinking and problem-solving skills for increased engagement.
3. **Optimization for Performance:** Rendering processes can be made more streamlined and resource usage can be minimized to ensure smooth gameplay across different platforms and devices, with adjustable graphics settings for customization.
4. **Responsive Controls:** The controls seem a little laggy and it makes multiple input impossible creating a situation when player can’t perform two operations at once. It can be addressed by prioritizing player input to ensure prompt and accurate responses, even during complex maneuvers, for a seamless gaming experience.

CHAPTER 6

REFERENCES

- Draw Color Cube Using Camera & Perspective Projection | *CG Lab Program – 4* | *OpenGL Programming*, https://www.youtube.com/watch?v=elw1WOziruk&t=1331s
- J. Kilgard, Mark. "The OpenGL Utility Toolkit (GLUT) Programming Interface." *OpenGL*, https://www.opengl.org/resources/libraries/glut/glut-3.spec.pdf
- E. Lee and P. Varaiya, "State Machines." Available: https://ptolemy.berkeley.edu/projects/chess/eecs124/reading/LeeAndVaraiya3_4.pdf