Fun Maze

A Mini Project report submitted to

Visvesvaraya Technological University

in partial fulfilment of the requirements

for the award of degree of

Bachelor of Engineering

in

Computer Science and Engineering

Submitted by

Karthikraj M N 1KG16CS044

Under the Guidance of

Mrs. Sandhya.A.Kulkarni

Assistant.Professor

Department of Computer Science and Engineering

K.S. School of Engineering and Management

No. 15, Mallasandra, off Kanakapura Road, Bengaluru-560109

2018-19

K.S. School of Engineering and Management

No. 15, Mallasandra, off Kanakapura Road, Bengaluru-560109

Department of Computer Science and Engineering

Certificate

*This is to certify that the Mini Project Report entitled "**Fun Maze**" is a bonafide work carried out by*

Karthikraj M N 1KG16CS044

*in partial fulfilment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** of Visvesvaraya Technological University, Belagavi, during the year 2018-19. It is certified that all the suggestions indicated during internal assessment have been incorporated in the mini project report and satisfies the academic requirement in respect of mini project work prescribed for the degree.*

_____	_____
Mrs. Sandhya.A.Kulkarni	Prof Veena R.S
Assistant Professor,	Associate Prof and HOD,
Dept. of CSE,KSSEM	Dept. of CSE,KSSEM

University Examiners:

_____	_____
Name and Signature of Examiner-1	Name and Signature of Examiner-2

Acknowledgement

The successful completion of this technical seminar was made possible with the help of guidance received from faculty members. I would like to avail this opportunity to express my sincere thanks and gratitude to all of them.

I would like to express my deepest gratitude and sincere thanks to my guide **Mrs. Sandhya A Kulkarni**, Assistant Professor, Department of Computer Science and Engineering, KSSEM, Bengaluru, for her keen interest and encouragement.

I would like to express my deepest gratitude and sincere thanks to my guide **Mr. Preetham B**, Assistant Professor, Department of Computer Science and Engineering, KSSEM, Bengaluru, for his keen interest and encouragement.

I am grateful to thank **Prof Veena R.S**, Associate Professor & Head, Department of Computer Science and Engineering, KSSEM, Bengaluru, for her valuable suggestions and advice throughout work period.

I had a great pleasure in expressing our deep sense of gratitude to the **KAMMAVARI SANGHAM** for providing us with a great infrastructure and well-furnished labs.

I would like to thank all the staff members of Department of Computer Science and Engineering for their support and encouragement during the course of this technical seminar.

Definitely most, I would like to thank all of my parents, family members and friends for their support and encouragement.

Karthikraj M N 1KG16CS044

Abstract

The project titled ‘FUN MAZE’ is a Computer Graphics application. The application is developed using OpenGL(Open Graphics Library) on Windows, which is a standard specification defining a cross – language, cross – platform API for writing applications that produce 2D and 3D computer graphics. The four major areas of computer graphics i.e. display of information, design, simulation & animation and user interfaces are attempted in the application.

This application uses OpenGL functions to demonstrate the implementation of an interactive player. The end user can interact with the onscreen player using keyboard keys (left arrow,right arrow,up arrow,down arrow). The player can move in forward, backward, up and down motions.

Contents

LIST OF FIGURES

Chapter 1

INTRODUCTION

Computer graphics is concerned with all aspects of producing pictures or images using a computer. This field began almost 50 years ago, with the display of a few lines on a cathode ray tube. Now we can create images by computers that are indistinguishable from photographs of real objects.

1.1 Computer Graphics

Computer graphics is one of the most effective and most commonly used means of communication with the user. It displays the information in the form of graphics objects such as pictures, charts, graphs and diagrams instead of simple text. The pictures or graphics objects may vary from engineering drawings, business graphs and architectural structures to animated movies. All the functionalities required for the development and presentation of such an environment or interface to the user is provided by the graphics package.

There is numerous number of ways in which computer graphics has made user interaction fast, effective and fun. Graphics has enabled the designers to introduce the concept of windows that act as virtual graphics terminals, each of which is capable of running an independent application. The introduction of the mouse has made the selection of objects on the interface easy by the “Point and Click” facility and a lot more.

With the speedily increasing enhancements in the field of computer graphics one can simulate real world objects, create motion by using the different strategies introduced in 2D, 3D and 4D dynamics, one can produce independent frames of objects and convey real time motion to the user through the fast and smooth movement of frames, produce packages for scientific and engineering visualizations, in the field of medicine for the study of human behavior and a lot more. So, we can see that computer graphics has become an integral part of life today and will continue to enhance and ease the usage of computers further more in the near future.

The below figure 1.1 shows the graphics system.

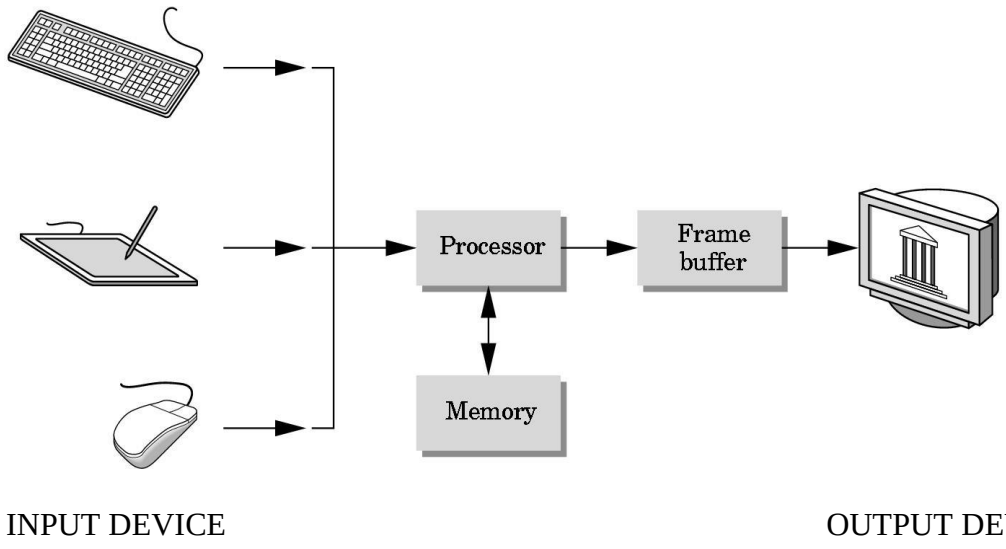

Figure.1.1 Graphics System

1.2 OpenGL

OpenGL was developed by Silicon Graphics Inc.(SGI) in 1992.OpenGL (Open Graphics Library) is a software interface to graphics hardware that produces 2D and 3D computer graphics. This interface consists of about 150 distinct commands that we use to specify the objects and operations needed to produce interactive 3D applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. With OpenGL, we can build up our desired model from a small set of geometric primitives-points, lines and polygon. The below figure 1.2 shows the graphics pipeline architecture.

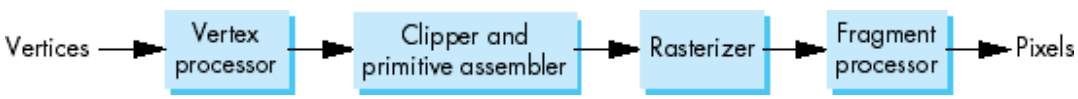

Figure.1.2 Graphics pipeline architecture

1.2.1 Purpose

OpenGL serves two main purposes:

- To hide the complexities of interfacing with different 3D accelerators, by presenting the programmer with a single, uniform API.
- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set.

1.2.2 Functionalities

Basic Functionalities in OpenGL include:

- Points, lines and polygons as basic primitives.
- A transform and lighting pipeline.
- Z-buffering.
- Texture mapping.
- Alpha blending.

1.2.3 OpenGL contributions

- It is very popular in the video games development industry where it competes with Direct3D (on Microsoft Windows).
- OpenGL is also used in CAD, virtual reality, and scientific visualization programs.
- OpenGL is very portable. It will run for nearly every platform in existence, and it will run well. It even runs on Windows NT 4.0 etc. The reason OpenGL runs for so many platforms is because of its Open Standard.
- OpenGL has a wide range of features, both in its core and through extensions. Its extension feature allows it to stay immediately current with new hardware features, despite the mess it can cause.
- Because the ARB is made up of a diverse group of companies, the features available in OpenGL represent a wide range of interests, and thus make it useful in many different applications.

1.2.4 Weakness

Though they are powerful, they do make code messy, very much so at times.

- They also make it confusing with any compiler that doesn't offer reference tracking (browse file). The worst part is, many newer extensions are completely card or vendor specific.
- Although useful for testing a graphics card's abilities, vendor-specific extensions are not frequently used by commercial applications.
- The function naming conventions can seem like overkill at times, since many IDEs have context sensitive help which can show you the parameters that are required, and in any case, if you know how to use a function, you should already know what parameters it takes. In addition, having 12 different names for a function may seem strange to a C++ programmer accustomed to function overloading (of course, since C doesn't support function overloading, there really isn't any way to get around it).

1.3 Procedural versus Descriptive

OpenGL provides you with fairly direct control over the fundamental operations of two-dimensional graphics. This includes specification of such parameters as transformation matrices, lighting equation coefficients, anti-aliasing methods, and pixel update operators. However, it doesn’t provide you with a means for describing or modeling complex geometric objects.

Thus, the OpenGL commands you issue specify how a certain result should be produced (what procedure should be followed) rather than what exactly that result should look like. That is, OpenGL is fundamentally procedural rather than descriptive. Because of this procedural nature, it helps to know how OpenGL works-the order in which it carries out its operations.

Execution Model

The model for interpretation of openGL commands is client-server. An application (the client) issues commands, which are interpreted and processed by OpenGL (the server).The server may or may not operate on the same computer as the client.

In this sense, OpenGL is network-transparent. A server can maintain several GL contexts, each of which is an encapsulated GL state. The required network protocol can be implemented by augmenting an already existing protocol (such as that of the X Window System) or by using an independent protocol. No OpenGL Commands are provided for obtaining user input.

1.4 Limitations

- OpenGL is case sensitive
- Line Color, Filled Faces and Fill Color not supported.
- Bump mapping is not supported.
- Shadow plane is not supported.
- Navigation Renderer is not supported.
- 3D measurement is not supported
- Streaming of individual 3D objectives is not supported

1.5 Literature Survey

- **Translation:**

Translation is an operation that displaces points by a fixed distance in a given direction.To specify a translation, we need only to specify a displacement vector d, because the transformed points are given by

P'= p +d

For all points up on the object. Translation has three of degrees of freedom because we can specify the three components of the displacement vector arbitrarily.

- **Rotation:**

Rotation is more difficult to specify than translation because we must specify more parameters. For example if we want to rotate a two dimensional point(x, y) in a frame with respect to origin by an angle theta to the position (x', y'), then we use a generalized formula:

X=$pcos\emptyset$

Y=p sin $\emptyset$

X'=$p\cos(\theta + \emptyset)$

Y'=p sin$(\theta + \emptyset)$

Rotation also has three degree of freedom, the two angles necessary to specify the orientation of the vector and the angle that specifies the amount of the rotation about the vector rotation

- **Scaling:**

Scaling is an affine non rigid body transformation by which we can make an object bigger or smaller. To specify a scaling we can specify the fixed point, a direction in which we wish to scale and scale factor. Scaling has six degrees of freedom because we can specify an arbitrary fixed point and three independent scaling factors.

Chapter 2

RESOURCE REQUIREMENTS

2.1 Hardware Requirements

The Hardware requirements are very minimal and the program can be run on most of the machines.

Processor	:	Intel CORE processor
Processor Speed	:	2.4 GHz
RAM	:	4 GB
Storage Space	:	40 GB
Monitor Resolution	:	1024*768 or 1336*768 or 1280*1024

2.2 Software Requirements

Operating System : Windows family

IDE : Microsoft Visual Studio 2019

OpenGL libraries

- **Header File**
    1. GL/glut.h
- **Object File Libraries**
    1. glu32.lib
    2. opengl32.lib
    3. glut32.lib
- **DLL files (Dynamic Link Libraries)**
    1. glu32.dll
    2. glut32.dll
    3. opengl32.dll
- **The programming is done in C**

Chapter 3

DESIGN

Design is the second stage in the software life cycle of any engineered product or system. Design is a creative process. A good design is the key to effective system. It may be defined as "the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization."

3.1 Description

This application uses OpenGL functions to demonstrate the implementation of an interactive player. The end user can interact with the onscreen player using keyboard keys. The player can move in forward, backward, up and down motions.

The user can use the following keys to interact with the player.

- right arrow – forward
- left arrow – backward
- down arrow – down
- up arrow – up

The figure 3.1 below shows the flow of the program. The main() function calls display() , SpecialKey() and mouse() functions. The display() function calls frontscreen(), startscreen(), wall(), instructions() and winscreen() functions which help in specific displaying. The keys used to interact with player are right arrow ,left arrow, down arrow and up arrow which comes under SpecialKey() function. The left click is used to swap between light mode and dark mode, this left click action comes under the mouse() function.

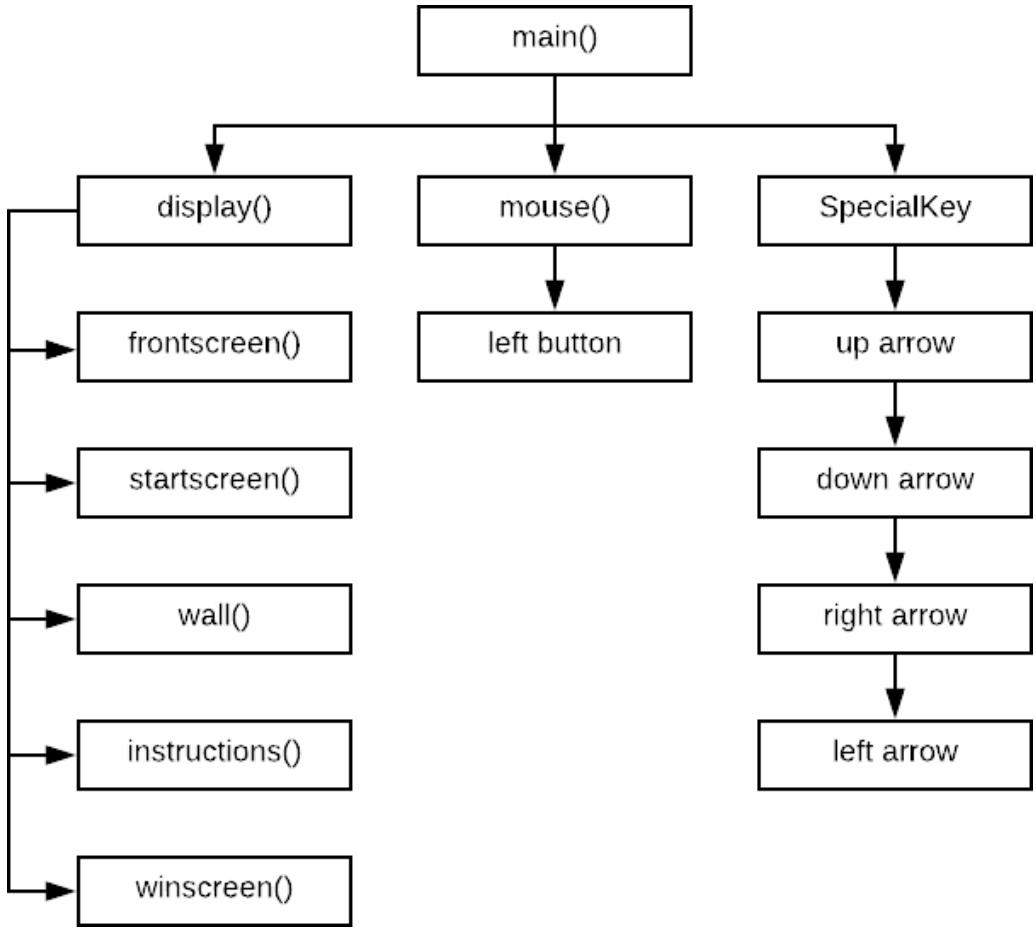

Figure:3.1 flow of the program

Chapter 4

IMPLEMENTATION

4.1 Header Files Used

- **#include<stdio.h>**

The standard input/output library requires the use of a header file called stdio.h. The include command is a directive that tells the compiler to use the information in the header file called stdio.h. The initials stdio stands for standard input output, and stdio.h file contains all the instructions the compiler needs to work with disk files and send information to the output device.

- **#include<GL/glut.h>**

GL is the fundamental openGL library. It provides functions that are permanent part of OpenGL. The functions start with characters ‘gl’.
GLUT, the GL utility tool kit supports developing and managing menus, and managing events. The functions start with characters ‘glut’.
GLU, the GL Utility Library provides high-level routines to handle certain matrix operations, drawing of quadric surfaces such as sphere and cylinders. The functions start with characters ‘glu’.

- **#include<math.h>**

math.h header file contains mathematical operations defined in it.

4.2 Description of the functions

- **main()**

The execution of the program starts from main().

- **glutInit()**

Initializes GLUT. The arguments from main are passed in and can be used by the application.

- **glutInitDisplayMode()**

Requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT_RGB,GLUT_INDEX) and buffering (GLUT_SINGLE,GLUT_DOUBLE).

- **glutInitWindowSize()**

Specifies the initial height and width of the window in pixels.

- **glutCreateWindow()**

Creates a window on the display. The string can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

- **glutDisplayFunc()**

Registers the display function that is executed when the window needs to be redrawn.

- **glutKeyboardFunc()**

Registers the keyboard callback function. The callback function returns the key and the position of the mouse relative to the top- left corner of window.

- **glutIdleFunc()**

Registers the display callback function that is executed whenever there are no other events to be handled.

- **glutMainLoop()**

Cause the program to enter an event-processing loop. It should be the last statement in main.

- **myinit()**

The function is defined to initialize the window parameters.

- **glClearColor()**

Sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating-point numbers between 0.0 and 1.0.

- **glMatrixMode()**

Specifies which matrix will be affected by subsequent transformations. mode can be GL_MODELVIEW,GL_PROJECTION or GL_TEXTURE.

- **glLoadIdentity()**

Sets the current transformation matrix to an identity matrix.

- **glOrtho()**

Defines an orthographic viewing volume with all parameters measured from the center of the projection plane.

- **display()**

The entire working of the program is graphically displayed on the screen by the contents defined in this function. This thus forms the heart of the program.
Here, all the conditions are checked the program instance must satisfy. Then suitably the instance and its current state are displayed.

- **glColor()**

Sets the present RGB colors. Valid types are byte (b), int (i), float (f), double (d), unsigned byte (ub), unsigned short (us) and unsigned int (ui). The maximum and minimum values of the floating point types are 1.0 and 0.0 respectively.

- **glutSwapBuffers()** Swaps the front and back buffers.

- **glPushMatrix()**

Pushes the current matrix stack down by one, duplicating the current matrix i.e., after a glPushMatrix call, the matrix on top of the stack is identical to one below it.

- **glPopMatrix()**

pops the current matrix on top of stack ,replacing the current matrix with one below it on the stack.

- **glFlush()**

Forces any buffered OpenGL commands to execute.

- **glutPostRedisplay()**

Requests that the display callback be executed after the current callback returns to 1. By checking for the above mentioned conditions, we display the queen suitably.

- **glutWireSphere()**

This function draws a wire frame sphere centered at origin. The “equatorial” great circle lies in the xy-plane.

- **glTranslatef()**

Alters the current matrix by a displacement of (x,y,z). Type is either GLfloat or GLdouble

Chapter 5

SOURCE CODE

```
#include<stdio.h>
#include<stdlib.h>
#include<GL/glut.h>
#include<math.h>
#include<string.h>
#include<time.h>
int x, y;
int i, count;
char t[2];
float px = 0.0, py = 175.0;
int flag, df = 10;
clock_t start, end;
void point()
{

glColor3f(0.0, 0.0, 1.0);
glBegin(GL_POINTS);
glVertex2f(px, py);
glEnd();
}
void point1()
{

glColor3f(.0, 1.0, 0.0);
glBegin(GL_POINTS);
glVertex2f(0.0, 175.0);
glEnd();
}
void point2()
{
```

```
glColor3f(1.0, 0.0, .0);
glBegin(GL_POINTS);
glVertex2f(176.0, 5.0);
glEnd();
}

void output(int x, int y, const char* string)
{
int len, i;

glRasterPos2f(x, y);
len = (int)strlen(string);
for (i = 0; i < len; i++)
{
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, string[i]);
}
}

void drawstring(int x, int y, const char* string, void* font)
{
int len, i;

glRasterPos2f(x, y);
len = (int)strlen(string);
for (i = 0; i < len; i++)
{
glutBitmapCharacter(font, string[i]);
}
}

void frontscreen(void)
{
```

```
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
glColor3f(1, 1, 1);
drawstring(120, 5, " Press ENTER to go To next screen",
GLUT_BITMAP_HELVETICA_18);
drawstring(-45, 5, "Maximize window for better view",
GLUT_BITMAP_HELVETICA_12);
glColor3f(1, 1, 1);
output(22, 160, " KS SCHOOL OF ENGINEERING AND MANAGEMENT");
glColor3f(1, 1, 1);
output(10.0, 150, "DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING");
glColor3f(1, 0, 1);
output(60, 130, "A Mini Project On:-");
glColor3f(0, 1, 0.5);
output(38, 120, "\"PATH FINDING USING OPENGL\"");
glColor3f(1, 0, 1);
output(40, 100, "By :");
glBegin(GL_LINES);
glVertex2f(40, 98);
glVertex2f(50, 98);
glEnd();
glColor3f(1, 0, 0);
output(40, 90, "ANIRUDH K N");
output(40, 80, "KARTHIKRAJ M N");
output(40, 80, "");
glColor3f(1, 0, 1);
output(40, 60, "Under the Guidence of :");
glBegin(GL_LINES);
glVertex2f(40, 58);
glVertex2f(98, 58);
glEnd();
```

```
glColor3f(1, 0, 0);
output(40, 50, "SANDEEP H");
glColor3f(1, 0, 0);
output(72, 50, "");
glColor3f(1, 0, 0);
output(70, 40, "Asst. Prof. Dept. of CSE");
glColor3f(1, 0, 0);
output(40, 30, "SANDHYA A  K");
glColor3f(1, 0, 0);
output(72, 30, "");

output(70, 20, "Asst. Prof. Dept. of CSE");
glFlush();

}

void winscreen()
{
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
glColor3f(0.0, 1.0, 0.0);
output(55, 120, "CONGRATS!!!");
glColor3f(1.0, 0.0, 1.0);
output(15, 100, "YOU HAVE SUCCEEDED IN FINDING OUT THE PATH");
output(35, 60, "* PRESS ESC TO GO TO MAIN MENU");
output(35, 45, "* PRESS 1 TO RESTART THE GAME");
glFlush();
}
void startscreen()
{

glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 1.0, 0.0);
```

```
output(25, 140, "WELCOME TO THE GAME FINDING THE PATH");
output(50, 100, "1.NEW GAME");
output(50, 80, "2.INSTRUCTIONS");
output(50, 60, "3.QUIT");
glFlush();
}

void instructions()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 0.0);
output(45, 140, "INSTRUCTIONS:");
glBegin(GL_LINES);
glVertex2f(45, 138);
glVertex2f(90, 138);
glEnd();
glColor3f(0, 1, 0);
output(-20, 120, "* TO MOVE THE POINT USE ARROW KEYS");
output(-20, 100, "* FIND THE WAY TO MOVE INTO THE MAZE AND GET OUT");
output(-20, 80, "* GREEN COLOURED POINT INDICATE THE POINT FROM
WHERE YOU HAVE TO START");
output(-20, 60, "* RED COLOURED POINT INDICATE THE POINT WHERE YOU
HAVE TO REACH");
output(-20, 40, "* YOU WILL HAVE TO HURRY AS YOU HAVE LIMITED TIME");
output(-20, 20, "* PRESS ESC TO GO TO MAIN MENU");
glFlush();
}

void idle()
{
if (df == 1)
{
end = clock();
```

```
count = (end - start) / CLOCKS_PER_SEC;
if (count == 3600)
{
df = 4;
}
else
if ((count < 3600) && ((px >= 175 && px <= 182) && (py >= 0 && py <= 6)))
{
df = 5;
}
}

glutPostRedisplay();
}

void wall(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2, GLfloat x3, GLfloat y3,
GLfloat x4, GLfloat y4)
{

glBegin(GL_POLYGON);
glVertex3f(x1, y1, 0);
glVertex3f(x2, y2, 0);
glVertex3f(x3, y3, 0);
glVertex3f(x4, y4, 0);
glEnd();

}

void SpecialKey(int key, int x, int y)
{
switch (key)
{
case GLUT_KEY_UP:
```

```
flag = 0;
if (py < 175)
if (!((px >= 168 && px <= 172) && (py >= 50 && py <= 50)))
if (!((px >= 138 && px <= 142) && (py >= 5 && py <= 5)))
if (!((px >= 148 && px <= 152) && (py >= 50 && py <= 50)))
if (!((px >= 132 && px <= 142) && (py >= 55 && py <= 55)))
if (!((px >= 138 && px <= 180) && (py >= 85 && py <= 85)))
if (!((px >= 108 && px <= 112) && (py >= 5 && py <= 5)))
if (!((px >= 88 && px <= 112) && (py >= 65 && py <= 65)))
if (!((px >= 108 && px <= 112) && (py >= 75 && py <= 75)))
if (!((px >= 112 && px <= 128) && (py >= 85 && py <= 85)))
if (!((px >= 168 && px <= 180) && (py >= 95 && py <= 95)))
if (!((px >= 98 && px <= 162) && (py >= 95 && py <= 95)))
if (!((px >= 128 && px <= 132) && (py >= 55 && py <= 55)))
if (!((px >= 92 && px <= 172) && (py >= 105 && py <= 105)))
if (!((px >= 88 && px <= 92) && (py >= 60 && py <= 60)))
if (!((px >= 78 && px <= 82) && (py >= 5 && py <= 5)))
if (!((px >= 68 && px <= 72) && (py >= 65 && py <= 65)))
if (!((px >= 22 && px <= 72) && (py >= 105 && py <= 105)))
if (!((px >= 28 && px <= 174) && (py >= 115 && py <= 115)))
if (!((px >= 146 && px <= 180) && (py >= 125 && py <= 135)))
if (!((px >= 72 && px <= 140) && (py >= 125 && py <= 125)))
if (!((px >= 28 && px <= 66) && (py >= 125 && py <= 125)))
if (!((px >= 143 && px <= 180) && (py >= 135 && py <= 135)))
if (!((px >= 18 && px <= 135) && (py >= 135 && py <= 135)))
if (!((px >= 18 && px <= 172) && (py >= 145 && py <= 145)))
if (!((px >= 8 && px <= 12) && (py >= 3 && py <= 158)))
if (!((px >= 8 && px <= 178) && (py >= 152 && py <= 165)))
if (!((px >= -4 && px <= 172) && (py >= 165 && py <= 172)))

py = py + 5;
glutPostRedisplay();
break;
```

```
case GLUT_KEY_DOWN:

flag = 0;
if (py > 5)
if (!((px >= 158 && px <= 162) && (py >= 85 && py <= 85)))
if (!((px >= 168 && px <= 172) && (py >= 50 && py <= 50)))
if (!((px >= 148 && px <= 152) && (py >= 50 && py <= 50)))
if (!((px >= 132 && px <= 142) && (py >= 65 && py <= 65)))
if (!((px >= 138 && px <= 180) && (py >= 95 && py <= 95)))
if (!((px >= 98 && px <= 102) && (py >= 65 && py <= 65)))
if (!((px >= 98 && px <= 102) && (py >= 95 && py <= 95)))
if (!((px >= 88 && px <= 112) && (py >= 75 && py <= 75)))
if (!((px >= 118 && px <= 122) && (py >= 85 && py <= 85)))
if (!((px >= 108 && px <= 112) && (py >= 95 && py <= 95)))
if (!((px >= 112 && px <= 128) && (py >= 95 && py <= 95)))
if (!((px >= 128 && px <= 132) && (py >= 55 && py <= 55)))
if (!((px >= 168 && px <= 180) && (py >= 105 && py <= 105)))
if (!((px >= 98 && px <= 162) && (py >= 105 && py <= 105)))
if (!((px >= 88 && px <= 92) && (py >= 60 && py <= 60)))
if (!((px >= 92 && px <= 172) && (py >= 115 && py <= 115)))
if (!((px >= 68 && px <= 72) && (py >= 65 && py <= 65)))
if (!((px >= 22 && px <= 72) && (py >= 115 && py <= 115)))
if (!((px >= 28 && px <= 174) && (py >= 125 && py <= 125)))
if (!((px >= 146 && px <= 180) && (py >= 135 && py <= 135)))
if (!((px >= 72 && px <= 140) && (py >= 135 && py <= 135)))
if (!((px >= 28 && px <= 66) && (py >= 135 && py <= 135)))
if (!((px >= 143 && px <= 180) && (py >= 145 && py <= 145)))
if (!((px >= 18 && px <= 135) && (py >= 145 && py <= 145)))
if (!((px >= 18 && px <= 172) && (py >= 155 && py <= 155)))
if (!((px >= 8 && px <= 178) && (py >= 165 && py <= 165)))
if (!((px >= -4 && px <= 172) && (py >= 168 && py <= 175)))
py = py - 5;
glutPostRedisplay();
```

```
break;

case GLUT_KEY_LEFT:
flag = 0;
if (px > 0)
if (!((px >= 5 && px <= 5) && (py >= 0 && py <= 170)))
if (!((px >= 165 && px <= 165) && (py >= 0 && py <= 82)))
if (!((px >= 175 && px <= 175) && (py >= 0 && py <= 47)))
if (!((px >= 175 && px <= 175) && (py >= 53 && py <= 88)))
if (!((px >= 145 && px <= 145) && (py >= 8 && py <= 58)))
if (!((px >= 155 && px <= 155) && (py >= 0 && py <= 47)))
if (!((px >= 155 && px <= 155) && (py >= 53 && py <= 88)))
if (!((px >= 145 && px <= 145) && (py >= 58 && py <= 88)))
if (!((px >= 115 && px <= 115) && (py >= 8 && py <= 68)))
if (!((px >= 105 && px <= 105) && (py >= 0 && py <= 62)))
if (!((px >= 105 && px <= 105) && (py >= 72 && py <= 92)))
if (!((px >= 115 && px <= 115) && (py >= 68 && py <= 72)))
if (!((px >= 125 && px <= 125) && (py >= 0 && py <= 82)))
if (!((px >= 115 && px <= 115) && (py >= 78 && py <= 92)))
if (!((px >= 135 && px <= 135) && (py >= 0 && py <= 52)))
if (!((px >= 165 && px <= 165) && (py >= 98 && py <= 102)))
if (!((px >= 135 && px <= 135) && (py >= 58 && py <= 108)))
if (!((px >= 95 && px <= 95) && (py >= 0 && py <= 57)))
if (!((px >= 175 && px <= 175) && (py >= 108 && py <= 112)))
if (!((px >= 95 && px <= 95) && (py >= 62 && py <= 118)))
if (!((px >= 85 && px <= 85) && (py >= 8 && py <= 118)))
if (!((px >= 75 && px <= 75) && (py >= 0 && py <= 62)))
if (!((px >= 75 && px <= 75) && (py >= 68 && py <= 112)))
if (!((px >= 50 && px <= 50) && (py >= 122 && py <= 128)))
if (!((px >= 175 && px <= 175) && (py >= 118 && py <= 122)))
if (!((px >= 145 && px <= 145) && (py >= 128 && py <= 132)))
if (!((px >= 70 && px <= 70) && (py >= 128 && py <= 132)))
if (!((px >= 140 && px <= 140) && (py >= 138 && py <= 142)))
```

```
if (!((px >= 175 && px <= 175) && (py >= 148 && py <= 152)))
if (!((px >= 25 && px <= 25) && (py >= 0 && py <= 152)))
if (!((px >= 15 && px <= 15) && (py >= 6 && py <= 158)))
if (!((px >= 175 && px <= 175) && (py >= 168 && py <= 172)))
px = px - 5;
glutPostRedisplay();
break;
case GLUT_KEY_RIGHT:
flag = 0;
if (px < 175)
if (!((px >= 155 && px <= 155) && (py >= 0 && py <= 82)))
if (!((px >= 165 && px <= 165) && (py >= 0 && py <= 47)))
if (!((px >= 165 && px <= 165) && (py >= 53 && py <= 88)))
if (!((px >= 135 && px <= 135) && (py >= 8 && py <= 58)))
if (!((px >= 145 && px <= 145) && (py >= 0 && py <= 47)))
if (!((px >= 145 && px <= 145) && (py >= 53 && py <= 88)))
if (!((px >= 135 && px <= 135) && (py >= 58 && py <= 92)))
if (!((px >= 105 && px <= 105) && (py >= 8 && py <= 68)))
if (!((px >= 95 && px <= 95) && (py >= 0 && py <= 62)))
if (!((px >= 95 && px <= 95) && (py >= 72 && py <= 92)))
if (!((px >= 115 && px <= 115) && (py >= 0 && py <= 82)))
if (!((px >= 105 && px <= 105) && (py >= 78 && py <= 92)))
if (!((px >= 125 && px <= 125) && (py >= 0 && py <= 52)))
if (!((px >= 165 && px <= 165) && (py >= 98 && py <= 102)))
if (!((px >= 95 && px <= 95) && (py >= 98 && py <= 102)))
if (!((px >= 125 && px <= 125) && (py >= 58 && py <= 108)))
if (!((px >= 85 && px <= 85) && (py >= 0 && py <= 57)))
if (!((px >= 85 && px <= 85) && (py >= 62 && py <= 118)))
if (!((px >= 75 && px <= 75) && (py >= 8 && py <= 118)))
if (!((px >= 65 && px <= 65) && (py >= 0 && py <= 62)))
if (!((px >= 65 && px <= 65) && (py >= 68 && py <= 108)))
if (!((px >= 30 && px <= 30) && (py >= 122 && py <= 128)))
if (!((px >= 25 && px <= 25) && (py >= 118 && py <= 122)))
```

```
if (!((px >= 143 && px <= 143) && (py >= 128 && py <= 132)))
if (!((px >= 69 && px <= 169) && (py >= 128 && py <= 132)))
if (!((px >= 25 && px <= 25) && (py >= 128 && py <= 132)))
if (!((px >= 140 && px <= 140) && (py >= 138 && py <= 142)))
//if (((px >= 180 && px <= 184) && (py >= 1 && py <= 5)))
if (!((px >= 15 && px <= 15) && (py >= 0 && py <= 152)))
if (!((px >= 5 && px <= 5) && (py >= 6 && py <= 158)))
if (!((px >= 5 && px <= 5) && (py >= 158 && py <= 162)))
px = px + 5;;
glutPostRedisplay();
break;
}
}
bool b=true;
void mouse(int btn, int state, int x, int y)
{
if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
b = !b;
}
void display()
{

glClear(GL_COLOR_BUFFER_BIT);
if (df == 10)
frontscreen();

else if (df == 0)
startscreen();
else if (df == 1)
{



//line();
```

```
if (b)
{
glColor3f(1.0, 1.0, 1.0);
glClearColor(0.0, 0.0, 0.0, 0.0);
}
if (!b)
{
glColor3f(0.0, 0.0, 0.0);
glClearColor(1.0, 1.0, 1.0, 0.0);

}

wall(-4, -4, 0, -4, 0, 168, -4, 168);
wall(-4, 178, -4, 184, 184, 184, 184, 178);
wall(180, 178, 184, 178, 184, 8, 180, 8);
wall(180, 0, 180, -4, -4, -4, 0, 0);
wall(-4, 172, 172, 172, 172, 168, -4, 168);
wall(8, 162, 180, 162, 180, 158, 8, 158);
wall(8, 158, 8, 8, 12, 8, 12, 158);
wall(18, 152, 18, 0, 22, 0, 22, 152);
wall(18, 152, 172, 152, 172, 148, 18, 148);
wall(18, 142, 18, 138, 137, 138, 137, 142 );
wall(143, 142, 143, 138, 180, 138, 180, 142);
wall(28, 132, 28, 128, 67, 128, 67, 132);
wall(73, 132, 73, 128, 142, 128, 142, 132);
wall(148, 132, 148, 128, 180, 128, 180, 132);
wall(28, 122, 28, 118, 172, 118, 172, 122);
wall(36, 128, 36, 122, 40, 122, 40, 128);
wall(22, 112, 22, 108, 72, 108, 72, 112);
wall(68, 108, 68, 68, 72 , 68, 72, 108);
wall(68, 62, 68, 0, 72, 0, 72, 62);
wall(78, 118, 78, 8, 82, 8, 82, 118);
wall(88, 118, 88, 63, 92, 63, 92, 118);
```

```
wall(92, 112, 92, 108, 172, 108, 172, 112);
wall(88, 57, 88, 0, 92, 0, 92, 57);
wall(132, 108, 132, 58, 128, 58, 128, 108);
wall(98, 102, 98, 98, 162, 98, 162, 102);
wall(168, 102, 168, 98, 180, 98, 180, 102);
wall(128, 52, 128, 0, 132, 0, 132, 52);
wall(112, 92, 112, 88, 128, 88, 128, 92);
wall(118, 82, 118, 0, 122, 0, 122, 82);
wall(108, 92, 108, 78, 112, 78, 112, 92);
wall(88, 72, 88, 68, 112, 68, 112, 72);
wall(98, 92, 98, 72, 102, 72, 102, 92);
wall(98, 0, 98, 62, 102, 62, 102, 0);
wall(108, 68, 108, 8, 112, 8, 112, 68);
wall(138, 92, 138, 88, 180, 88, 180, 92);
wall(138, 88, 138, 58, 142, 58, 142, 88);
wall(132, 62, 132, 58, 142, 58, 142, 62);
wall(138, 58, 138, 8, 142, 8, 142, 58);
wall(148, 47, 148, 0, 152, 0, 152, 47);
wall(148, 88, 148, 53, 152, 53, 152, 88);
wall(158, 82, 158, 0, 162, 0, 162, 82);
wall(168, 88, 168, 53, 172, 53, 172, 88);
wall(168, 47, 168, 0, 172, 0, 172, 47);
glutPostRedisplay();
output(-21, 172, "---->");
output(184, 3, "--- >");
glColor3f(1.0, 0.5, 0.0);
output(35, 80, "HOPE");
output(36, 70, "YOU ");
output(35, 60, "FIND ");
output(36, 50, "THE ");
output(35, 40, "WAY. ...");
output(240, 160, t);
glutPostRedisplay();
```

```
point();
point1();
point2();

}

else if (df == 2)
instructions();
else if (df == 3)
{
exit(1);
}
else if (df == 4)
{
//timeover();
}
else if (df == 5)
{
winscreen();
}

glFlush();

}
void keyboard(unsigned char key, int x, int y)
{

if (df == 10 && key == 13)
df = 0;

else if ((df == 0 || df == 4 || df == 5) && key == '1')
{
df = 1;
```

```
glutPostRedisplay();
}
else if (df == 0 && key == '2')
df = 2;
else if (df == 0 && key == '3')
df = 3;
else if (key == 27)
{
df = 0;
}
if ((key == '0' || key == '1') && (df == 4 || df == 1))
{
px = 0.0;
py = 175.0;
}
glutPostRedisplay();
}
void myinit()
{
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glPointSize(18.0);

glMatrixMode(GL_MODELVIEW);

glClearColor(0.0, 0.0, 0.0, 0.0);

// glClearColor(1.0, 1.0, 1.0, 0.0);
glFlush();
glutSwapBuffers();
glutPostRedisplay();
```

```
}

void myreshape(int w, int h)
{
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if (w <= h)
gluOrtho2D(45.0, 135.0, -2.0 * (GLfloat)h / (GLfloat)w, 180.0 * (GLfloat)h /
(GLfloat)w);
else
gluOrtho2D(-45.0 * (GLfloat)w / (GLfloat)h, 135.0 * (GLfloat)w / (GLfloat)h, -2.0,
180.0);
glMatrixMode(GL_MODELVIEW);
glutPostRedisplay();
}

int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(600, 600);
glutCreateWindow("Pathfinding game");
glutReshapeFunc(myreshape);
glutDisplayFunc(display);
glutMouseFunc(mouse);
glutIdleFunc(idle);
glutSpecialFunc(SpecialKey);
glutKeyboardFunc(keyboard);
myinit();
glutMainLoop();
return 0;
}
```

Chapter 6

TESTING

6.1 Unit Testing

Unit testing implies the first stage of dynamic testing process. According to software development expert Barry Boehm, a fault discovered and corrected in the unit testing phase is more than a hundred times cheaper than if it is done after delivery to the customer.

It involves analysis of the written code with the intention of eliminating errors. It also verifies that the codes are efficient and adheres to the adopted coding standards. Testing is usually white box. It is done using the Unit test design prepared during the module design phase. This may be carried out by software developers.

6.2 Integration Testing

In integration testing the separate modules will be tested together to expose faults in the interfaces and in the interaction between integrated components. Testing is usually black box as the code is not directly checked for errors.

6.3 System Testing

System testing will compare the system specifications against the actual system. After the integration test is completed, the next test level is the system test. System testing checks if the integrated product meets the specified requirements. Why is this still necessary after the component and integration tests? The reasons for this are as follows:

Reasons for system test

a) In the lower test levels, the testing was done against technical specifications, i.e., from the technical perspective of the software producer. The system test, though, looks at the system from the perspective of the customer and the future user. The testers validate whether the requirements are completely and appropriately met.

Example - The customer (who has ordered and paid for the system) and the user (who uses the system) can be different groups of people or organizations with their own specific interests and requirements of the system.

b) Many functions and system characteristics result from the interaction of all system components, consequently, they are only visible on the level of the entire system and can only be observed and tested there.

6.4 User Acceptance Testing

Acceptance testing is the phase of testing used to determine whether a system satisfies the requirements specified in the requirements analysis phase. The acceptance test design is derived from the requirements document. The acceptance test phase is the phase used by the customer to determine whether to accept the system or not.

The following description is unacceptable in and overview article Acceptance testing:

- To determine whether a system satisfies its acceptance criteria or not.
- To enable the customer to determine whether to accept the system or not.
- To test the software in the "real world" by the intended audience.

  Purpose of acceptance testing
- To verify the system or changes according to the original needs.

Chapter 7

RESULTS

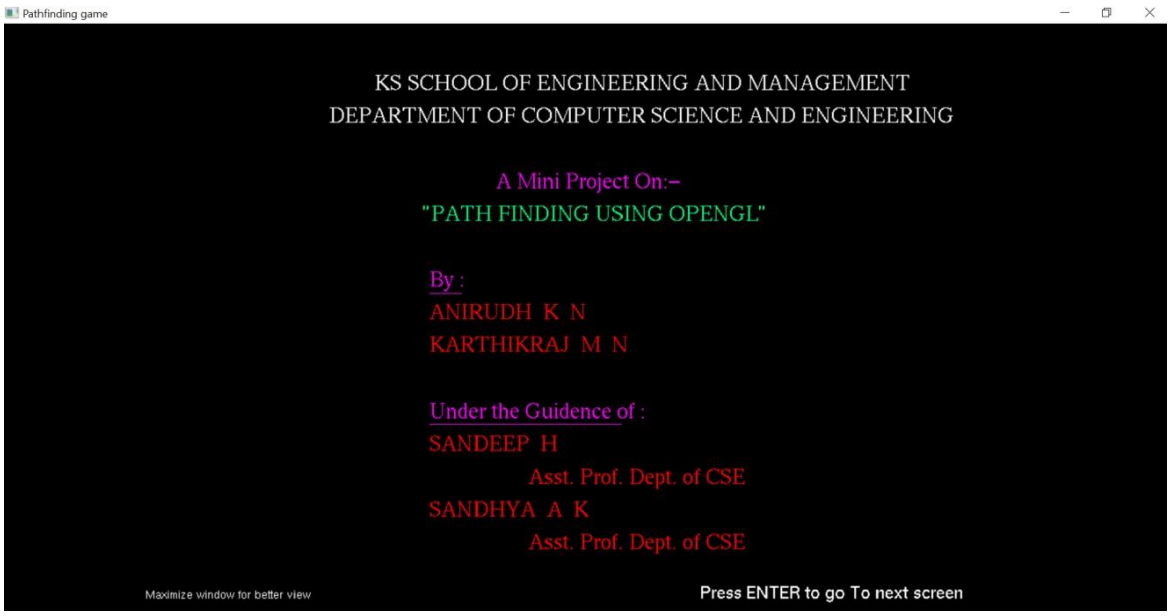

Figure:7.1 Opening Screen of the project

The above figure 7.1 shows the opening screen as the program starts to execute.

Figure:7.2 Welcome page

The above figure 7.2 shows the welcome page of the game with some options .

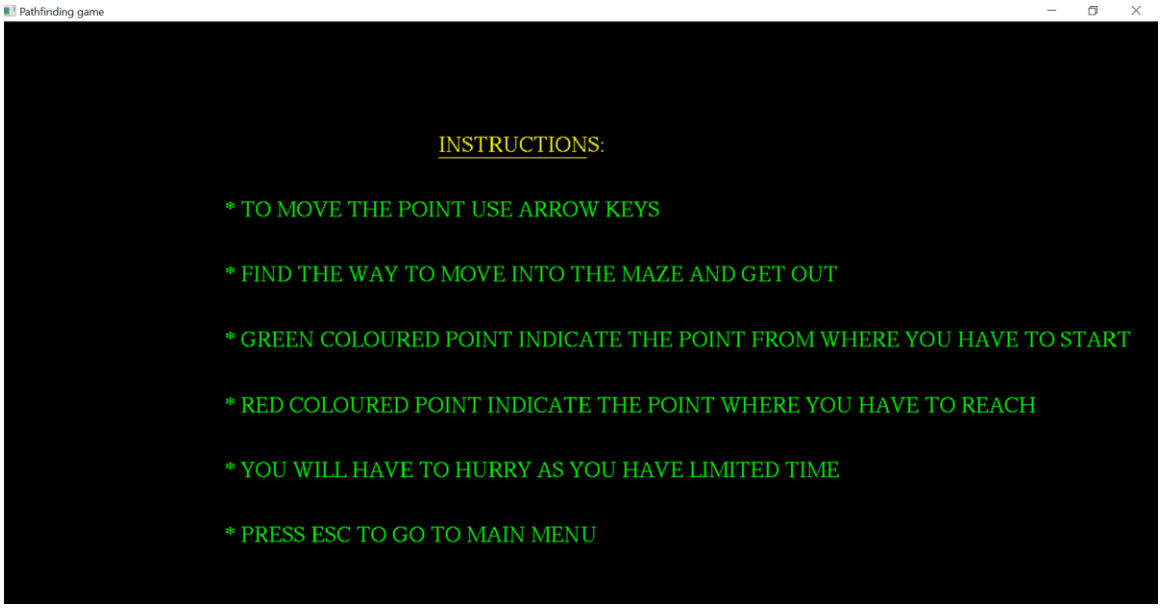

Figure:7.3 Instructions page

The above figure 7.3 shows the instructions to be followed while playing the game.

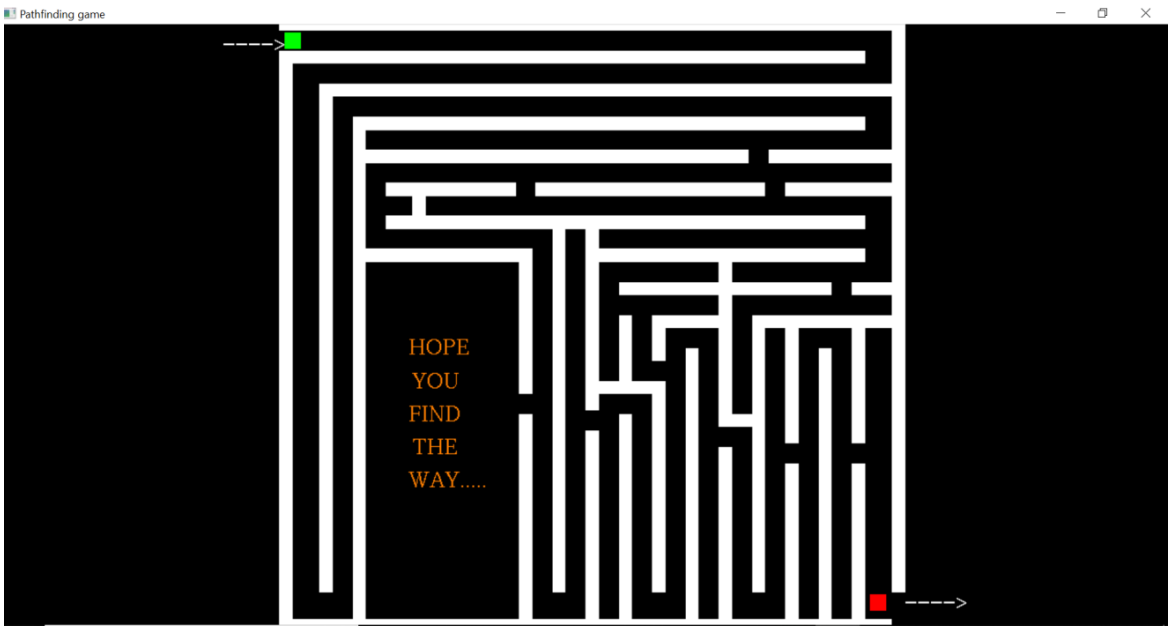

Figure:7.4 Initial position of the player

The above figure 7.4 shows the initial position of player before any key is pressed.

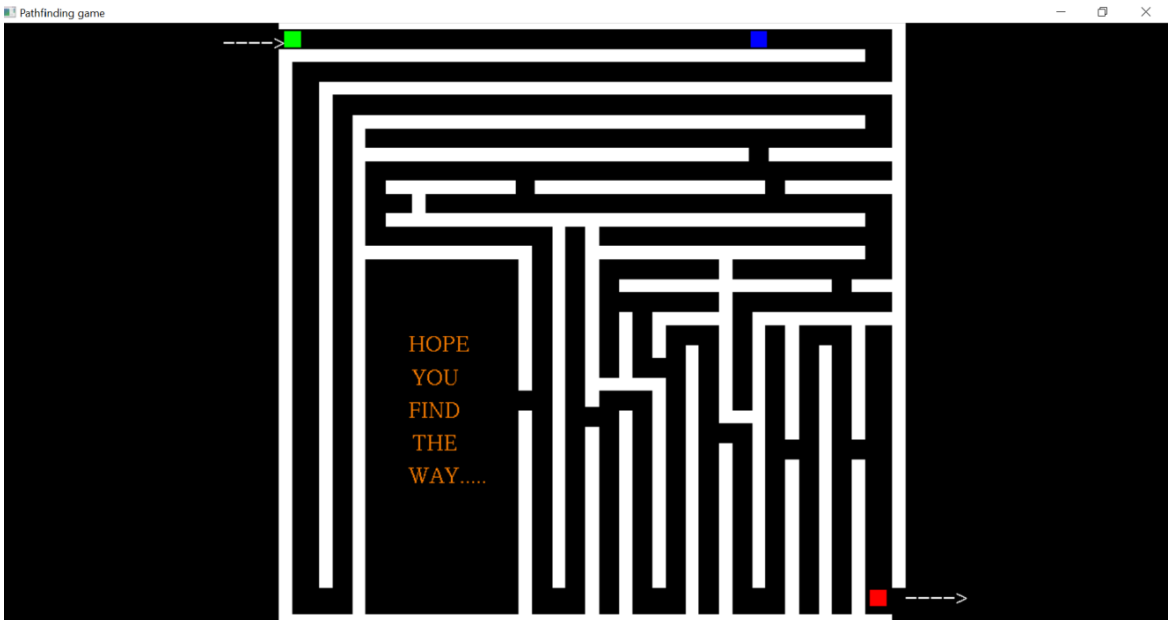

Figure 7.5:Position of player on pressing key-right arrow

The above figure 7.5 shows the position of player when ‘right arrow’ key is pressed. When this key is pressed, the player moves forward.

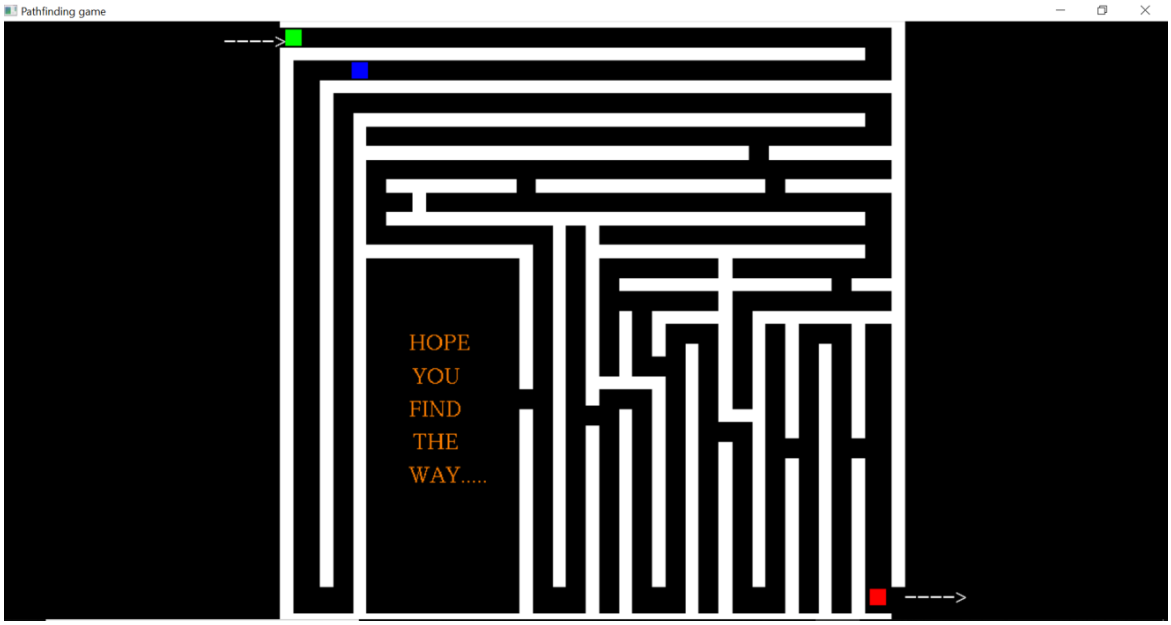

Figure:7.6 Position of player on pressing key-left arrow

The above figure 7.6 shows the position of player when ‘left arrow’ key is pressed. When this key is pressed, the player moves backward.

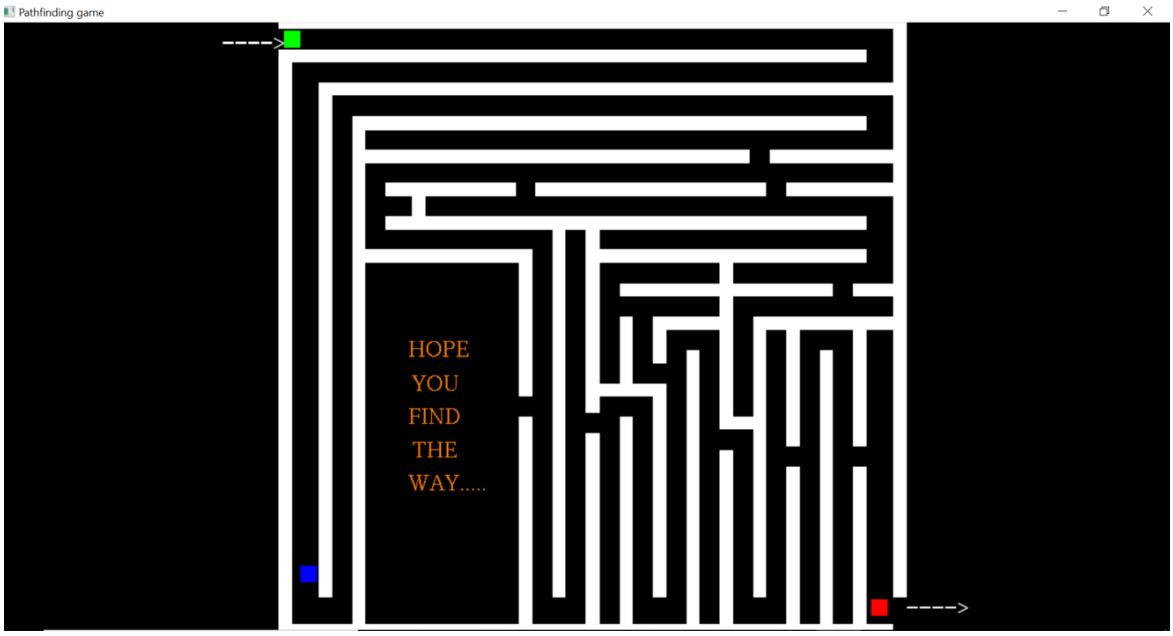

Figure:7.7 Position of player on pressing key-down arrow

The above figure 7.7 shows the position of player when ‘down arrow’ key is pressed. When this key is pressed, the player moves downward.

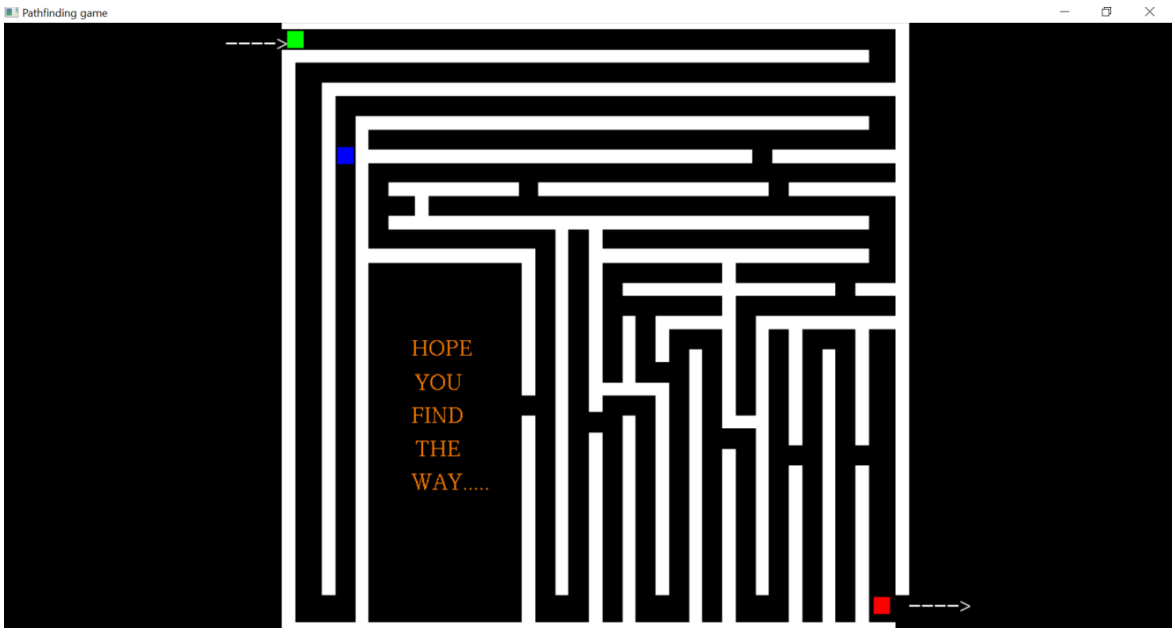

Figure :7.8 Position of player on pressing key-up arrow

The above figure 7.8 shows the position of player when ‘up arrow’ key is pressed. When this key is pressed, the player moves upwards

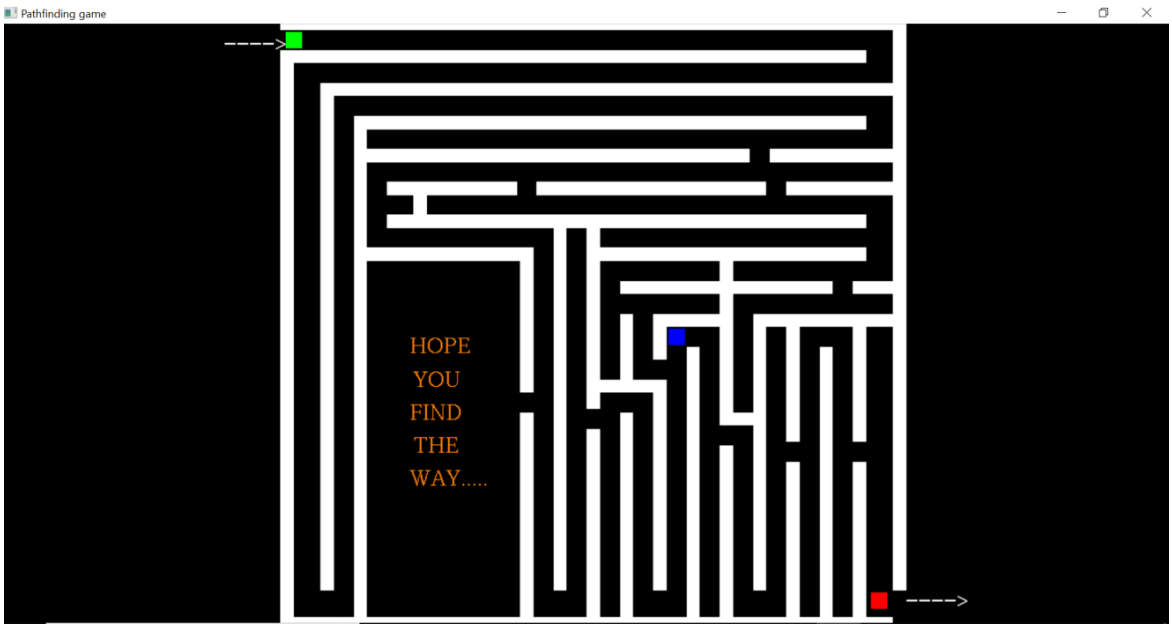

Figure :7.9 Position of player during the game progress

The above figure 7.8 shows the position of player when the game is in the progress .

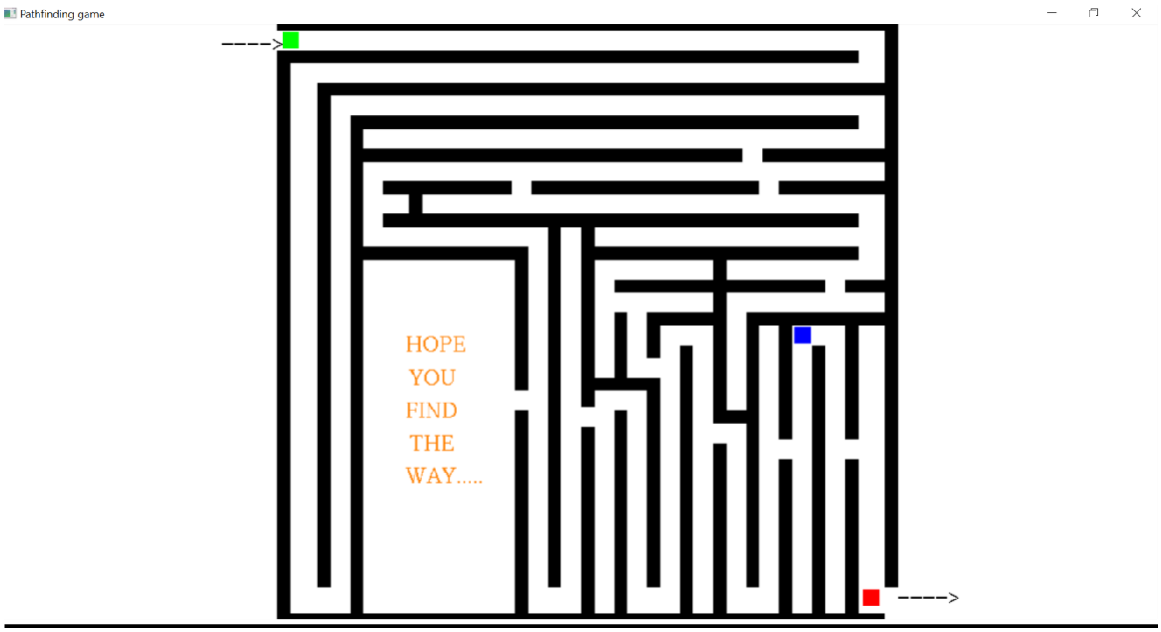

Figure :7.10 Game view during mouse action

The above figure 7.10 shows the the game scene when the left mouse button is clicked.

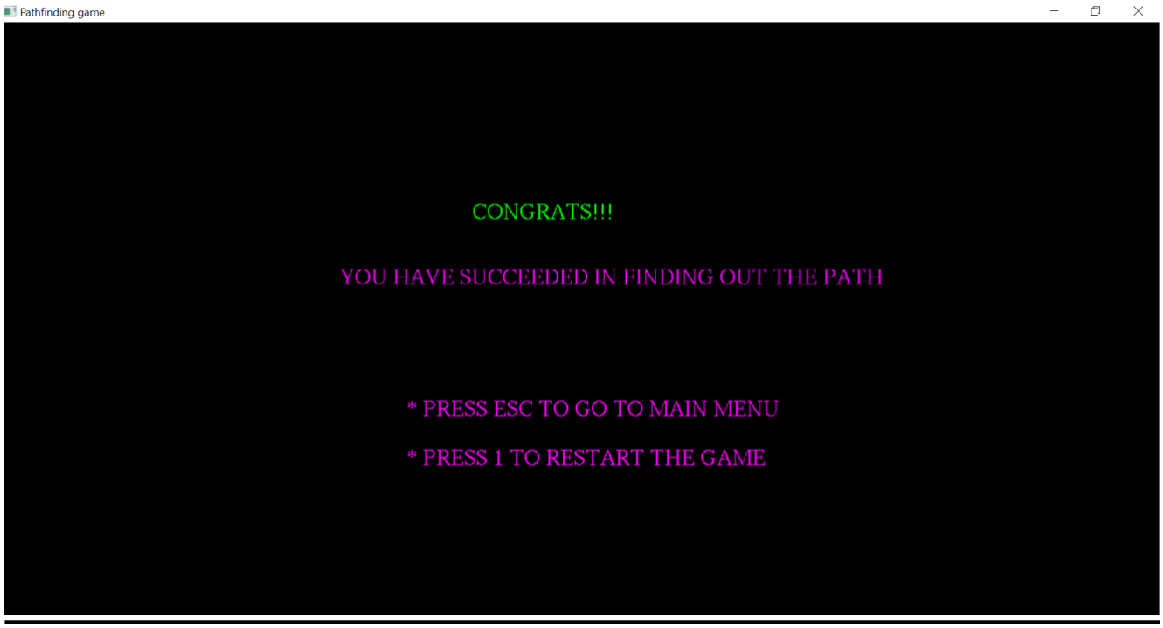

Figure :7.11 Completion screen

The above figure 7.11 shows the screen displayed when the player is finally reached the destination.

Chapter 8

CONCLUSION AND FUTURE ENHANCEMENT

Moving is implemented using OpenGL and its library functions. The program graphically displays the movement of the player using keyboard inputs. It takes the input from the user and moves the onscreen player accordingly. The primary objective of this project is to graphically show the movement of player. The onscreen player can be moved in up, down, forward and backward motions. The idea was to simplify the understanding of the concept.

The present project shows the movement of player in 2D. The further enhancements that can be made to this project are:

- Make the project visually appealing by giving the display a 3D perspective.
- Lighting and shading can be implemented.
- Modifying the maze according to the levels and difficulties.

BIBLIOGRAPHY

[1] Interactive Computer Graphics A Top-Down Approach with OpenGL, Edward Angel, 5th Edition, Addison- Wesley, 2008.

[2] http://www.cprogramming.com/tutorial/opengl_introduction.html

[3] www.opengl.org/resources/code/samples/redbook/