REPORT

"Perplexity"
A Simple Maze Game

BY: AJAY V RAIKAR

ABSTRACT

"PERPLEXITY" is a 2D Maze game. The game is created using OpenGL where the player tries to solve the maze. The objective of the game is to navigate through the maze and complete the game within a minute then he wins the game either he loses the game.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OPENGL

OpenGL is the abbreviation for Open Graphics Library. It is a software interface for graphics hardware. This interface consists of several hundred functions that allow you, a graphics programmer, to specify the objects and operations needed to produce high-quality color images of two-dimensional and three-dimensional objects. Many of these functions are actually simple variations of each other, so in reality there are about 120 substantially different functions. The main purpose of OpenGL is to render two-dimensional and three-dimensional objects into the frame buffer. These objects are defined as sequences of vertices (that define geometric objects) or pixels (that define images).OpenGL performs several processes on this data to convert it to pixels to form the final desired image in the frame buffer.

1.2 HISTORY

As a result, SGI released the **OpenGL** standard In the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. Software developers wrote custom interfaces and drivers for each piece of hardware. This was expensive and resulted in much duplication of effort.

By the early 1990s, Silicon Graphics (SGI) was a leader in 3D graphics for workstations. Their IRIS GL API was considered the state of the art and became the de facto industry standard, overshadowing the open standards-based PHIGS. This was because IRIS GL was considered easier to use, and because it supported immediate mode rendering. By contrast, PHIGS was considered difficult to use and outdated in terms of functionality.

SGI's competitors (including Sun Microsystems, Hewlett-Packard and IBM) were also able to bring to market 3D hardware, supported by extensions made to the PHIGS standard. This in turn caused SGI market share to weaken as more 3D graphics hardware suppliers entered the

market. In an effort to influence the market, SGI decided to turn the Iris GL API into an open standard.

SGI considered that the Iris GL API itself wasn't suitable for opening due to licensing and patent issues. Also, the Iris GL had API functions that were not relevant to 3D graphics. For example, it included a windowing, keyboard and mouse API, in part because it was developed before the X Window System and Sun's NEWS systems were developed.

In addition, SGI had a large number of software customers; by changing to the OpenGL API they planned to keep their customers locked onto SGI (and IBM) hardware for a few years while market support for OpenGL matured. Meanwhile, SGI would continue to try to maintain their customers tied to SGI hardware by developing the advanced and proprietary Iris Inventor and Iris Performer programming APIs.

1.3 FEATURES OF OPENGL

- **Industry standard**

  An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.

- **Stable**

  OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

- **Reliable and portable**

  All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.

- **Evolving**

  Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

- **Scalable**

  OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

- **Easy to use**

  OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.

- **Well-documented**

  Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

1.4 BASIC OPENGL OPERATION

The following diagram illustrates how OpenGL processes data. As shown, commands enter from the left and proceed through a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during various processing stages.

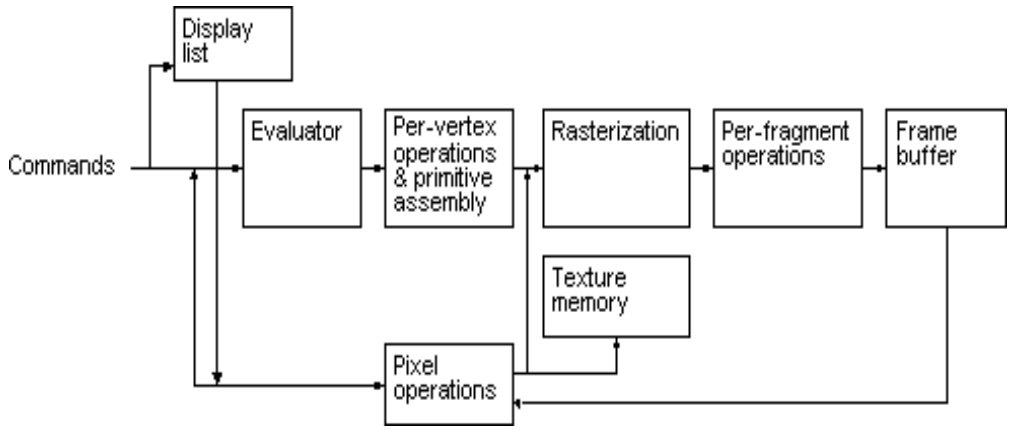

Fig: 1.4 OpenGL Block Diagram

The processing stages in basic OpenGL operation are as follows:

- **Display list**

  Rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing later.

- **Evaluator**

  The evaluator stage of processing provides an efficient way to approximate curve and surface geometry by evaluating polynomial commands of input values.

- **Per-vertex operations and primitive assembly**

  OpenGL processes geometric primitives - points, line segments, and polygons all of which are described by vertices. Vertices are transformed, and primitives are clipped to the viewport in preparation for rasterization.

- **Rasterization**

  The rasterization stage produces a series of frame-buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each so produced is fed into the last stage, per-fragment operations.

- **Per-fragment operations**

  These are the final operations performed on the data before it is stored as pixels in the frame buffer Per-fragment operations include conditional updates to the frame buffer based on incoming and previously stored z values (for z buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

- **Pixel operation**

  Input data can be in the form of pixels rather than vertices. Such data which might describe an image for texture mapping skips the first stage of processing and instead processed as pixels in the pixel operation stage.

- **Texture memory**

  The result of pixel operation stage is either stored as texture memory for use in rasterization stage or rasterised and resulting fragment merged into the frame buffer just as they were generated from the geometric data.

1.5 THE OPENGL INTERFACE

Most of our applications will be designed to access OpenGL directly through functions in three libraries. They are

- **GL – Graphics Library**

  Functions in the main GL (or OpenGL in Windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in Windows).

- **GLU – Graphics Utility Library**

  This library uses only GL functions but contain code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begins with the letters glu.

- **GLUT – OpenGL Utility Toolkit**

  To interface with the window system and to get input from external devices into our programs we need at least one more library. For the X window System, this library is called GLX, for Windows, it is wgl, and for the Macintosh, it is agl. Rather than using a different library for each system, we use a readily available library called the OpenGL Utility Toolkit (GLUT) , which provides minimum functionality that should be expected in any modern windowing system.

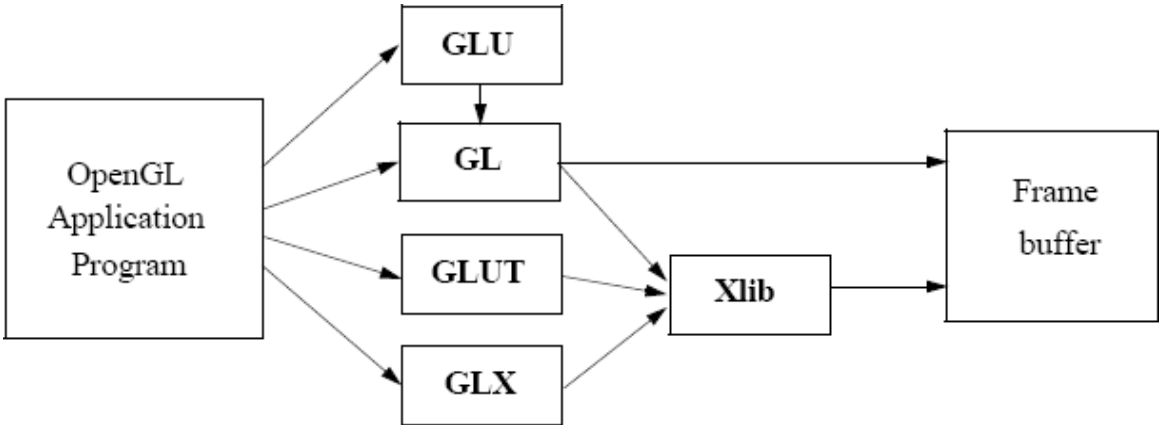

Fig: 1.5 Library Organization

The above figure shows the organization of the libraries for an X Window System environment.

In most implementations, one of the include lines

#include<GL/glut.h>

or

#include<GLUT/glut.h>

is sufficient to read in glut.h, gl.h and glu.h.

1.6 GRAPHICS FUNCTIONS

Our basic model of a graphics package is a black box, a term that engineers use to denote a system whose properties are described only by its inputs and outputs; we may know nothing about its internal workings.

OpenGL functions can be classified into seven major groups:

- **Primitive function**

  The primitive functions define the low-level objects or atomic entities that our system can display. Depending on the API, the primitives can include points, lines, polygons, pixels, text, and various types of curves and surfaces.

- **Attribute functions**

  If primitives are the what of an API – the primitive objects that can be displayed- then attributes are the how. That is, the attributes govern the way the primitive appears on the display. Attribute functions allow us to perform operations ranging from choosing the color with which we display a line segment, to picking a pattern with which to fill inside of a polygon.

- **Viewing functions**

  The viewing functions allow us to specify various views, although APIs differ in the degree of flexibility they provide in choosing a view.

- **Transformation functions**

  One of the characteristics of a good API is that it provides the user with a set of transformations functions such as rotation, translation and scaling.

- **Input functions**

  For interactive applications, an API must provide a set of input functions, to allow users to deal with the diverse forms of input that characterize modern graphics systems. We need functions to deal with devices such as keyboards, mice and data tablets.

- **Control functions**

  These functions enable us to communicate with the window system, to initialize our programs, and to deal with any errors that take place during the execution of our programs.

- **Query functions**

  If we are to write device independent programs, we should expect the implementation of the API to take care of the differences between devices, such as how many colors are supported or the size of the display. Such information of the particular implementation should be provides through a set of query functions.

1.7 DATA TYPES

OpenGL supports different data types. A list of data types supported by OpenGL is given in the following table.

Sl no.	Suffix	Data type	C type	OpenGL type
1.	B	8 bit int	signed int	GLbyte
2.	S	1 bit int	Short	GLshort
3.	I	32 bit int	Long	GLint , GLsizei
4.	F	32 bit float	Float	GLfloat ,GLclampf
5.	D	64 bit float	Double	GLdouble, GLclampd
6.	Ub	8 bit unsigned	unsigned char	GLubyte, GLboolean
7.	Us	16 bit unsigned	unsigned short	GLushort
8.	Ui	32 bit unsigned	unsigned int	GLuint, GLenum, GLbitfield

Table: 1.1 DATA TYPES LIST

1.8 OBJECTIVES

The objectives of this study are summarized below:

- To develop a Open GL software called “PERPLEXITIY”.
- To build the environment for the player to improve his quick thinking/accuracy.
- To build the basic platform of problem solving for the player.
- To progress the thinking ability of the player to solve the game.
- To navigate through the maze and complete the game within a minute then he wins the game either he loses the game.

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 SOFTWARE REQUIREMENTS

1. Operating System : Microsoft Windows XP, Microsoft Windows 7
2. Compiler used: VC++ 6.0 compiler
3. Language used: Visual C++

2.2 HARDWARE REQUIREMENTS

1. Processor: Intel® Core™ i3-32 bit
2. Processor Speed: 2.9 GHz
3. RAM Size: 8GB DDR3
4. Graphics – 2GB
5. Cache Memory: 2MB

CHAPTER 3

SYSTEM DESIGN

3.1 INITIALIZATION

- Initialize to interact with the Windows.
- Initialize the display mode that is double buffer and RGB color system.
- Initialize window position and window size.

Initialize and create the window to display the output.

3.2 DISPLAY

- Introduction page of "PREPLEXITY"
- Menus are created and depending on the value returned by menus.
- Suitable operations are performed.
- The operations performed are:
  - New Game
  - Instructions
  - Quit

3.3 FLOW CHART

When we run the program, home window appears. On clicking 'Enter' button Main window is opened. In main window list of options like New Game, Instructions & Quit appears. By selecting any of these options we can perform the specified operation in the game.

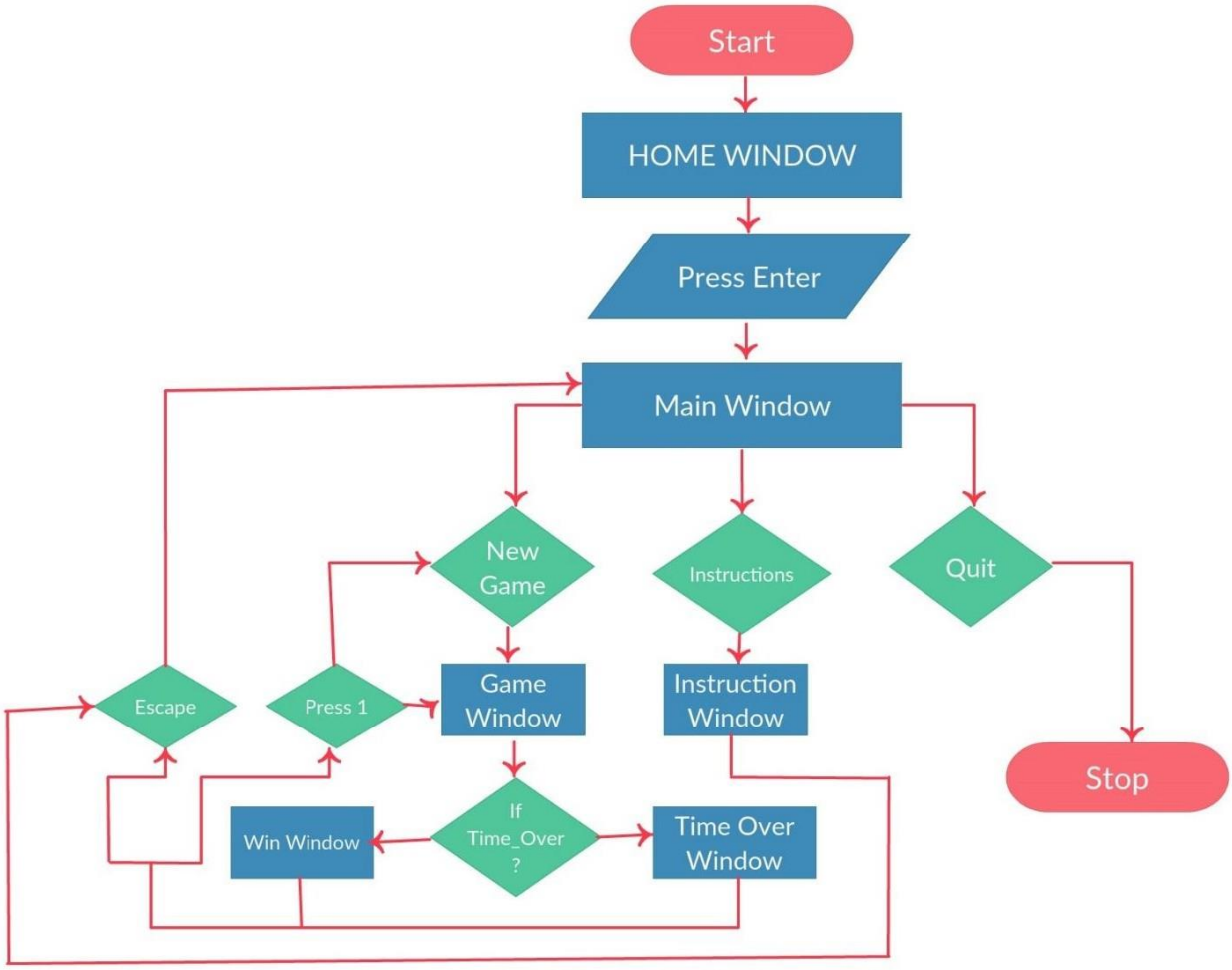

Fig: 3.3 Flow Chart

CHAPTER 4

IMPLEMENTATION

4.1 OVERVIEW

This project is a demonstration of "**Maze Game**". We have taken the help of built in functions present in the header file. To provide functionality to our project we have written sub functions. These functions provide us the efficient way to design the project. In this chapter we are describing the functionality of our project using these functions.

Keyboard interactions are provided where, when a Enter button is pressed, menu displays and we can select options from menu displayed.

4.2 USER INTERFACE

The Project which we have done uses OpenGL functions and is implemented using C. Our Project is to demonstrate **MAZE GAME**. User can perform operations using keyboard.

Keyboard interaction

- Firstly, after compiling we get a Home Page.
- Then we click the Enter button to display the Main window here we get three options in which user has to specify his choices:
  - New Game: To start the new game.
  - Instructions: It Guides the user how to play the game.
  - Exit: Quits the Game.
- As the player clicks 1 i.e. To open the new game.
- Now in game the player uses the arrow key to complete the game.
- Regardless of a win or a lose the player is redirected to pop-up page, where again he has to specify his choice.

4.3 STRUCTURE

- void point();
- void point1();
- void point2();
- void output(int x,int y,char *string);
- void draw_string(int x,int y,char *string);
- void frontscreen(void);
- void winscreen();
- void startscreen();
- void instructions();
- void timeover();
- void idle();
- void wall();
- void specialkey(int key,int x,int y);
- void display();
- void keyboard(unsigned char key,int x,int y);
- void myinit();
- void myreshape(int w,int h);
- int main(int argc,char** argv);

4.4 ANALYSIS

FUNCTIONS

A function is a block of code that has a name and it has a property that it is reusable that is it can be executed from as many different points in a c program as required.

The partial code of various function that have been used in the program are:

4.4.1 myinit

```
void myinit()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glPointSize(18.0);
        glMatrixMode(GL_MODELVIEW);
        glClearColor(0.0,0.0,0.0,0.0);
}
```

This function is used to initialize the graphics window.glMatrixMode(GL_PROJECTION), glLoadIdentity() are used to project the output on to the graphics window.

4.4.2 Display

```
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        if(df==10)
                frontscreen();
```

```
else if(df==0)
        startscreen();
else if(df==1){
        output(-21,172,"---->");
        output(-21,163,"<----");
        glColor3f(0.0,0.0,1.0);
        output(185,160,"TIME REMAINING : ");
        drawstring(190,130,"HURRY UP",GLUT_BITMAP_HELVETICA_18);
        glColor3f(1,0,0);
        drawstring(190,140,"Time is running out",GLUT_BITMAP_HELVETICA_18);
        sprintf(t,"%d",60-count);
        output(240,160,t);
        glutPostRedisplay();
        point();
        point1();
        point2();
        //line();
        glColor3f(1.0,1.0,1.0);
        wall(-4,-4,0,-4,0,162,-4,162);
        ……
        ……
        wall(8,162,8,158,0,158,0,162);
        glutPostRedisplay();
}
```

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

If df==10, i.e., it will call the frontscreen(), else if df==0 then startscreen() is called, Now the game has been started with the timer of 60sec displaying the MAZE to be solved by the player

4.4.3 Wall

```
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();
}
```

This function is used to display the Wall forming the Maze.

4.4.4 Point

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

This function is used to create a color point in the game to identify the start & end point. In the game starting point is green & end point is red, and the player's color point is blue which he uses to play the game

4.4.5 Frontscreen

```
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(72,30,"(B.E.)",GLUT_BITMAP_HELVETICA_12);
        output(70,20,"Lecturer,Dept. of CSE");
        glFlush();
}
```

This is the function which helps in opening the Home page of the game. This page is linked to all other pages described before. After clicking enter in this page the Main page is opened.

4.4.6 Idle

```
void idle()
{
	if(df==1)
	{
		end=clock();
		count=(end-start)/CLOCKS_PER_SEC;
		if(count==60)
			df=4;
		else if((count<60) && ((px>=0 && px<=4) && (py>=162 && py<=168)))
			df=5;
	}
	glutPostRedisplay();
}
```

This function is the major criteria of this game as it sets a timer for the player which limits the player to finish his game within 60sec else he loses the game.

CHAPTER 5

SNAPSHOTS

1.After running the program

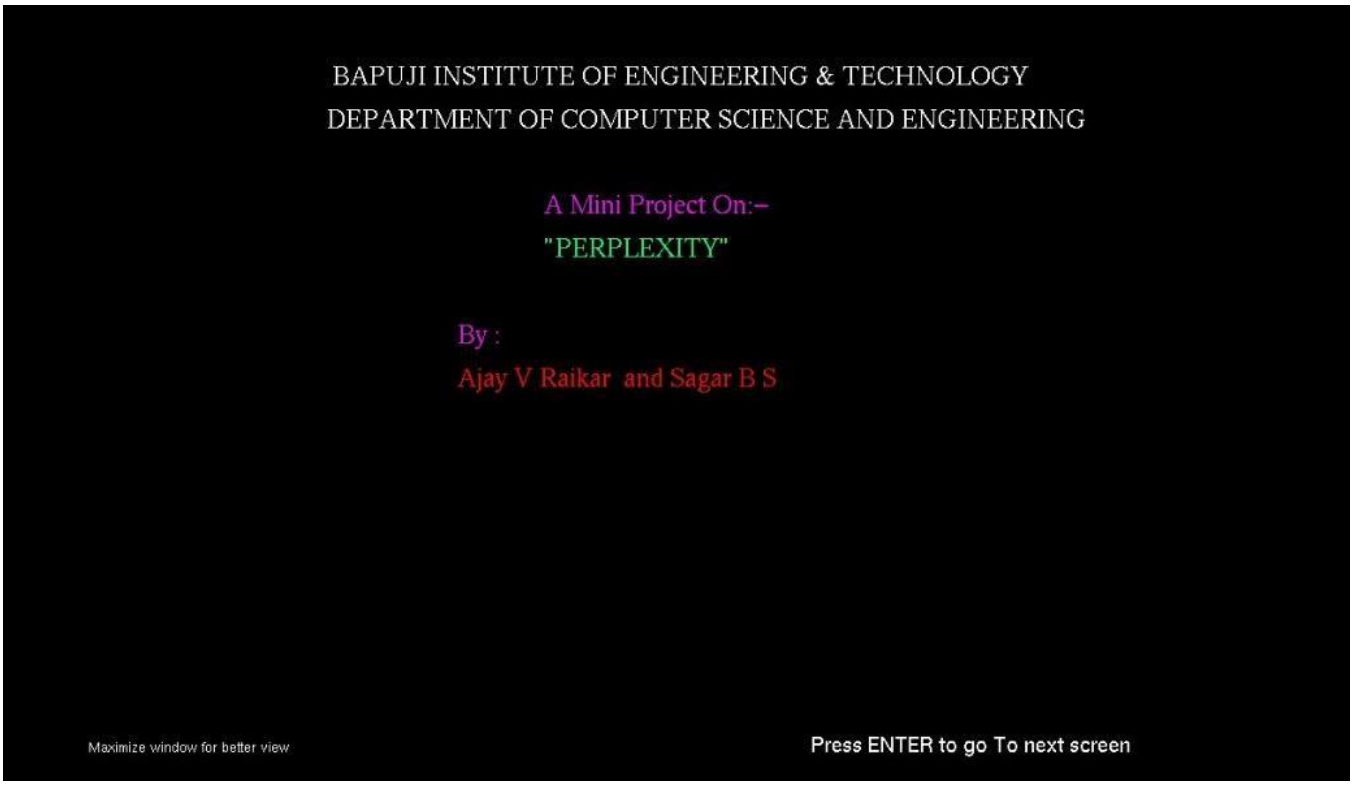

Fig 5.1: Home page

The above snapshot shows the screen displayed when the program gets Executed.

2. Game Menu

Fig 5.2: Main Window

The above snapshot shows the Game Menu 1st one to start the game , 2nd option guides the player how to use the game & 3rd option Exits the game.

3.The Game

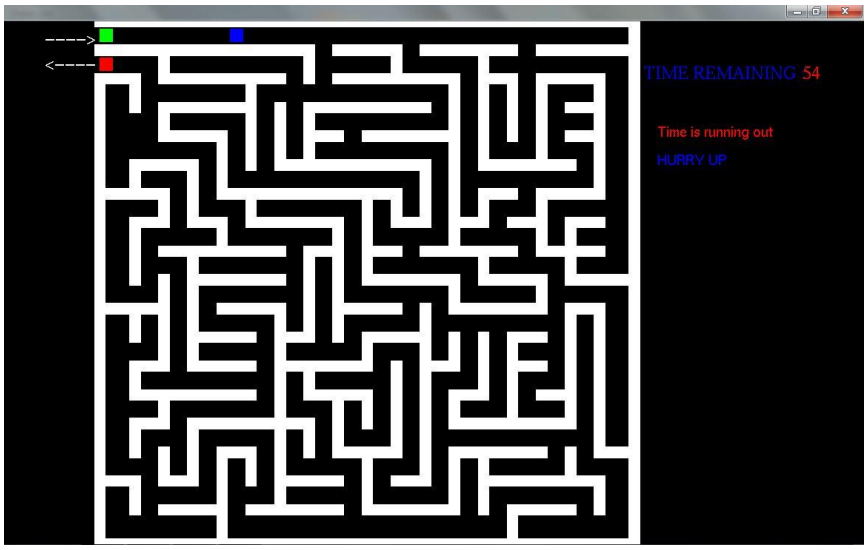

Fig 5.3: Game Page

The above snapshot our Maze Game with green mark as the starting point & red mark is the end point & the player uses the blue block.

4. Win Screen

Fig 5.4: Win Page.

The above snapshot shows the win screen as the player has finished the game within 60 Sec.

5. Game Over (Lost!!)

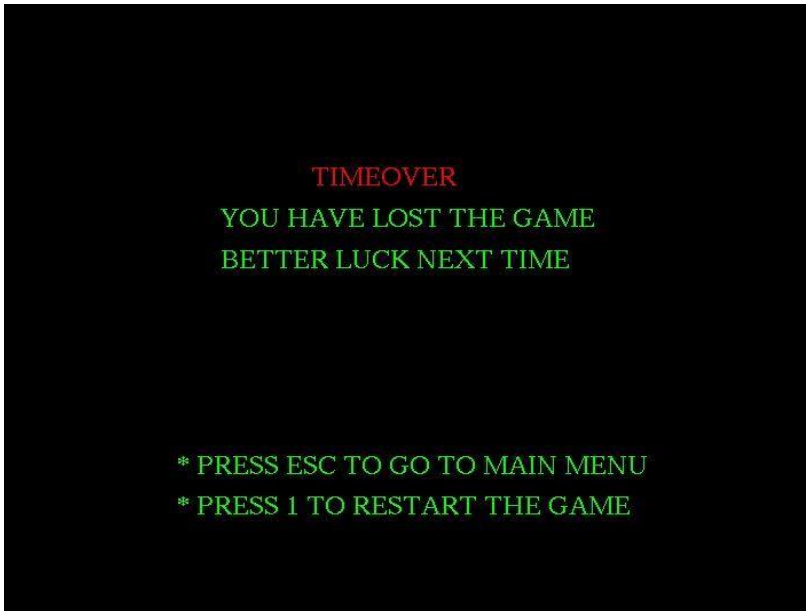

Fig 5.5: Game Over Page

The above snapshot shows the lost screen as the player has not finished the game within 60 Sec.

6. Instructions

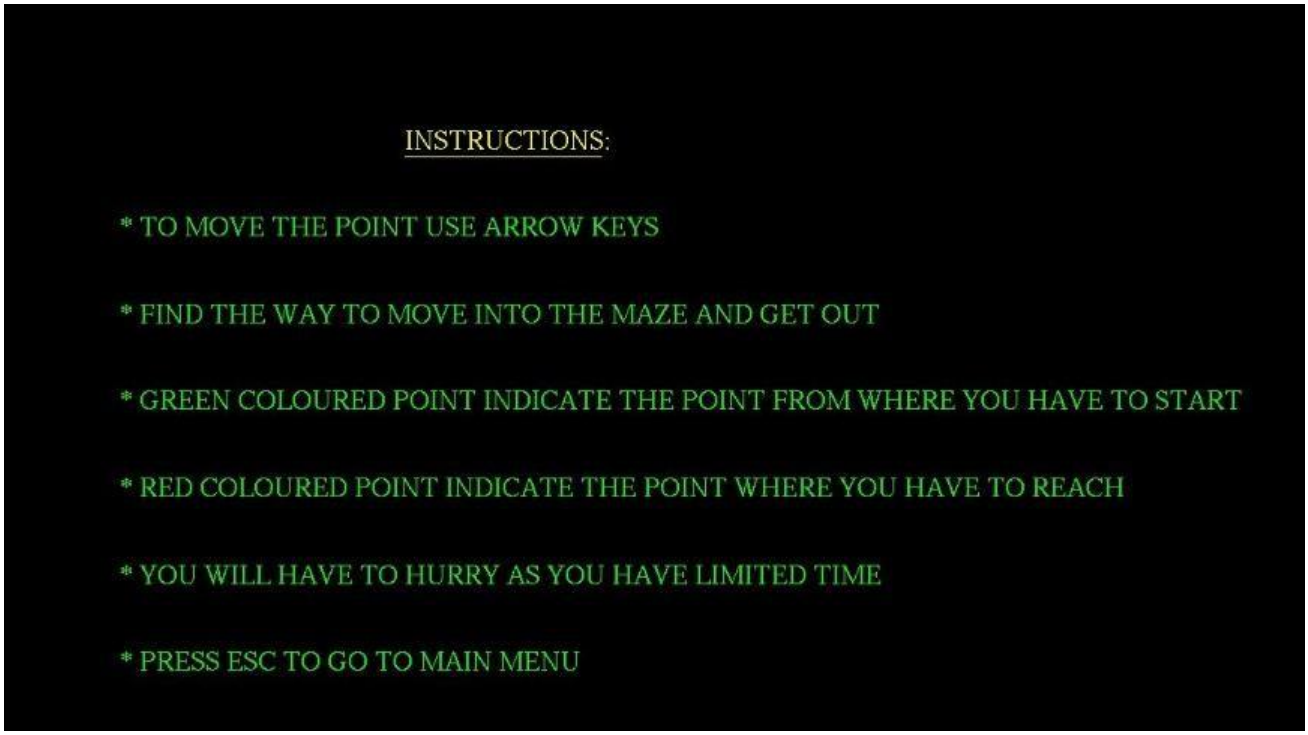

Fig 5.6: Instruction Page

The above snapshot shows the instruction to the player , how he has to play game

CONCLUSION

PERPLEXITY is designed and implemented using a graphics software system called **OpenGL** which has became a widely accepted standard for developing graphic application. Using OpenGL functions user can create geometrical objects and can use **translation, rotation, scaling** with respect to the co-ordinate system. The development of this project has enabled us to improve accuracy, problem solving skills while providing a fun and interactive experience to the player.

BIBLIOGRAPHY

Books:

1. Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version,3rd / 4th Edition, Pearson Education,2011

2. Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL, 5th edition. Pearson Education, 2008

3. James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer graphics with OpenGL: pearson education

4. Xiang, Plastock : Computer Graphics , sham's outline series, 2nd edition, TMG.

List of websites:

1. http://www.opengl.org/
2. http://www.academictutorials.com/graphics/graphics-flood-fill.asp
3. http://www.glprogramming.com/
4. https://www.opengl.org/discussion_boards/showthread.php/167379-Making-a-maze-using-arrays

APPENDIX

- **GL:** Graphics Library
- **GLU:** Graphics Utility Library
- **GLUT:** OpenGL Utility Tool kit
- **glutInitDisplayMode():** It sets the initial display mode.
- **glutDisplayFunc():** Sets the display callback for the *current window*.
- **glutInitWindowPosition():** Initializes GLUT and specifies command-line options for window system in use.
- **glColor():**Set the current color.
- **2D:** Two dimentional.
- **glLoadidentity():**This replaces the current matrix with the identity matrix.
- **glutPostRedisplay():** This marks the current window as needing to be redisplayed.
- **API:** Application Programming Interface.
- **GLX:** OpenGL Extension to the X Window System.