**CHAPTER 1**

**INTRODUCTION**

* 1. **Overview of computer graphics**

Computer Graphics become a powerful tool for the rapid and economical production of pictures. There is virtually no area in which Graphical displays cannot be used to some advantage so it is not surprising to find the use of CG so widespread.

Although early application in engineering & science had to rely on expensive and cumbersome equipment, advances in computer technology have made interactive computer graphics a practical tool. Today people find Computer Graphics in a diverse area such as science, engineering, medicine, business, industry, government, art, entertainment, education and training. Computer graphics as generalized tool for drawing and creating pictures and simulate the real world situations within a small computer window.

**1.2 History**

William fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeing. One of the first displays of computer animation was future world (1976), which included an animation of a human face and hand-produced by Carmel and Fred Parkle at the University of Utah.

There are several international conferences and journals where the most significant results in computer-graphics are published. Among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on Graphics journals.

**1.3 Applications of computer graphics**

Nowadays Computer Graphics used in almost all the areas ranges from science, engineering, medicine, business, industry, government, art, entertainment, education and training.

**1.3.1 CG in the field of CAD**

Computer Aided Design methods are routinely used in the design of buildings,

Automobiles, aircraft, watercraft, spacecraft computers, textiles and many other applications.

**1.3.2 CG in presentation Graphics**

Another major application area presentation graphics used to produce illustrations for reports or generate slides. Presentation graphics is commonly used to summarize financial, statistical, mathematical, scientific data for research reports and other types of reports.2D and 3D bar chart to illustrate some mathematical or statistical report.

**1.3.3 CG in computer Art**

CG methods are widely used in both fine art and commercial art applications. Artists use a variety of computer methods including special purpose hardware, artist’s paintbrush program (lumena), other pain packages, desktop packages, mathematics packages, animation packages that provide facility for designing object motion. Ex: cartoons design is an example of computer art which uses CG.

**1.3.4 Entertainment**

Computer graphics methods are now commonly used in making motion pictures, music, videos, games and sounds. Sometimes graphics objects are combined with the actors and live scenes.

**1.3.5 Education and Training**

Computer generated models of physical financial, economic system is often as education aids. For some training application special systems are designed. Ex: specialized system is simulator for practice sessions or training of ship captain, aircraft pilots and traffic control.

**1.3.6 Image Processing**

Although the methods used in CG image processing overlap, the 2 areas are concerned with fundamentally different operations. In CG a computer is used to create picture. Image processing on the other hand applies techniques to modify existing pictures such as photo scans, TV scans.

**1.4 User interface**

It is common for software packages to provide a graphical interface. A major component of a graphical interface is a window manager that allows a user to display multiple window area. Interface also displays menus, icons for fast selection and processing.

* 1. Problem statement

. Learning computer graphics as a part of academics and implementing it as a project on OpenGL made us to create a game i.e. Maze Runner game in OpenGL which included all the basic library functions.

* 1. Objectives of the project
* The main objective this project is to show how the “Maze Runner” game works with the help of built-in graphics library functions.
* Learning how to build a similar game like Mazes & More.
* Illustrating about the keyboard and mouse interaction.
* To provide entertainment to the user.

1.7 Organization of the report

This section deals with the Introduction and organization of the project report. Chapter 2 discusses the basic concepts OpenGL. Chapter 3 discusses the basic concept and working principle. Chapter 4 gives information about the design and implementation Chapter 5 include results and snapshot and. Chapter 6 gives the conclusion and future enhancement of the project.

**CHAPTER 2**

**INTRODUCTION TO OPENGL**

**2.1 Introduction**

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

Most of our application will be designed to access OpenGL directly through functions in three libraries. 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). The second is the OpenGL Utility Library (GLU). This library uses only GL functions but contains 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 begin with letters glu.

To interface with the window system and to get input from external devices into our programs, developer needs at least one more library. For each major window system there is a system-specific library that provides the “glu” between the window system and OpenGL. 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, here it uses a readily available library called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.

Fig 2.1 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.

# Fig 2.1 Library organization of OpenGL

OpenGL application

Program

GLU

GL

GLUT

GLX

Xlib, Xtk

Frame

Buffer

**2.2 OpenGL command syntax**

OpenGL commands use the prefix gland initial capital letters for each word making up the command name. Similarly, OpenGL defined constants begin with GL\_, use all capital letters, and use underscores to separate words (like GL\_TRUE).

Some extraneous letters are appended to some command names (for example, the 3f in glColor3f() and glVertex3f()). It's true that the Color part of the command name glColor3f() is enough to define the command as one that sets the current color. However, more than one such command has been defined so as to use different types of arguments. In particular, the 3 part of the suffix indicates that three arguments are given; another version of the Color command takes four arguments. The f part of the suffix indicates that the arguments are floating-point numbers. Having different formats allows OpenGL to accept the user's data.

**2.3 Pixel operation**

Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the Rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory.

There are special pixel copy operations to copy data in the frame buffer to other parts of the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer.

**2.4 Texture assembly**

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory.

**2.5 Rasterization**

Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support initializing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

Fragment

Processor

Rasterizer

Clipper and

Primitive Assembler

Vertex Processor

vertices Pixel

**Figure 2.2 Block diagram showing Rasterization**

**2.6 Immediate mode and display lists**

All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

#### Transforming to Window Coordinates

Before clip coordinates can be converted to window *coordinates*, they are normalized by dividing by the value of w to yield *normalized device coordinates*. After that, the viewport transformation applied to these normalized coordinates produces window coordinates. One can control the viewport, which determines the area of the on-screen window that displays an image, with glDepthRange() and glViewport().

#### Matrix Transformations

Vertices and normals are transformed by the modelview and projection matrices before they're used to produce an image in the frame buffer. You can use commands such as glMatrixMode(), glMultMatrix(), glRotate(), glTranslate(), and glScale() to compose the desired transformations, or you can directly specify matrices with glLoadMatrix() and glLoadIdentity(). Use glPushMatrix() and glPopMatrix() to save and restore modelview and projection matrices on their respective stacks.

The basic model for OpenGL command interpretation is immediate mode, in which a command is executed as soon as the server receives it; vertex processing, for example, may begin even before specification of the primitive of which it is a part has been completed. Immediate mode execution is well-suited to interactive applications in which primitives and modes are constantly altered. In OpenGL, the fine-grained control provided by immediate mode is taken as far as possible: even individual lighting parameters (the diffuse reflectance color of a material, for instance) and texture images are set with individual commands that have immediate effect. While immediate mode provides flexibility, its use can be inefficient if unchanging parameters or objects must be re-specified. To accommodate such situations, OpenGL provides display lists. A display list encapsulates a sequence of OpenGL commands (all but a handful of OpenGL commands may be placed in a display list), and is stored on the server. The display list is given a numeric name by the application when it is specified; the application need only name the display list to cause the server to effectively execute all the commands contained within the list. This mechanism provides a straightforward, effective means for an application to transmit a group of commands to the server just once even when those same commands must be executed many times.

**2.6.1 Display List Optimization**

Accumulating commands into a group for repeated execution presents possibilities for optimization. Consider, for example, specifying a texture image. Texture images are often large, requiring a large, and therefore possibly slow, data transfer from client to server (or from the server to its graphics subsystem) whenever the image is re-specified. For this reason, some graphics subsystems are equipped with sufficient storage to hold several texture images simultaneously. If the texture image definition is placed in a display list, then the server may be able to load that image just once when it is specified. When the display list is invoked (or re-invoked), the server simply indicates to the graphics subsystem that it should use the texture image already present in its memory, thus avoiding the overhead of re-specifying the entire image. Examples like this one indicate that display list optimization is required to achieve the best performance. In the case of texture image loading, the server is expected to recognize that a display list contains texture image information and to use that information appropriately. This expectation places a burden on the OpenGL implementer to make sure that special display list cases are treated as efficiently as possible. It also places a burden on the application writer to know to use display lists in cases where doing so could improve performance. Another possibility would have been to introduce special commands for functions that can be poor performers in immediate mode. But such specialization would clutter the API and blur the clear distinction between immediate mode and display lists.

**2.7 Advantages of using 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.

**CHAPTER 3**

**BASIC CONCEPT AND WORKING PRINCIPLE**

“MAZE RUNNER” game is designed completely using an Open GL functions. This project uses simple function “GL\_POINTS”, which creates point object and “GL\_LINES” which creates lines forming the walls of this game. The game is so simple that when the game starts the box will be at the starting position. As soon as the game starts the timer begins. The player playing this game has to use the arrow keys to move the box.The timer will be running. The player has to find the right path to complete the game in the given time. When the player loses to find the right path in the given time he looses the game, player can quit the game or can restart the game if he wishes to play again.

**3.1 Working Principle of the game**

Box will be at “START” when the new game starts and as the game proceeds the player has to reach “END” to win the game. Player has to press arrow keys to move the box in x axis and y axis. As soon as the game starts timer goes on decrementing. Box should reach the destination before timer gets over or else the game ends. If player wants to exit the game in middle,press the “ESCAPE” key.

**3.2 Rules**

* Only one player is involved in the game.
* The player can use the arrow keys from the keyboard to move the box from starting point to the ending point.
* The player is given only one chance to win the game.
* If the player finds the right path in the given amount of time he wins the game.
* If the player fails to find the right path in given time he looses the game. If he wants to try again he should restart the game.

**CHAPTER 4**

**DESIGN AND IMPLEMENTATION**

**4.1 Design**

The whole program has been implemented in C language. The bottom line of the design is keyboard interaction and some minor functions are used to print the text on the screen.

**4.2 Implementation**

The OpenGL provides very powerful translation facilities which relive the programmers by allowing them to concentrate on their jobs rather than focusing on how to implement these operations

**4.2.1 Built in functions**

* **void glutInitDisplayMode(unsigned int mode)**

*mode* – specifies the display mode. Use the *mode* to specify the colour mode

and the number of type of buffers.

* **void glutInitWindowSize(int width, int height)**

*Width* - The width of the window.

*Height* - The height of the window.

* **void glutWindowPosition(int x, int y)**

*x* -Window X location in pixels.

*y* -Window Y location in pixels.

glutWindowPosition set the initial window position and size respectively.

* **int glutCreateWindow(char \*name);**

*name* - ASCII character string for use as window name.

glutCreateWindowcreates a top-level window.

* **void glutDisplayFunc(void (\*func)(void));**

*func* - The new display callback function.

glutDisplayFunc sets the display callback for the current window*.*

* **void glutKeyboardFunc(void (\*func)(char key,int width,int height));**

*func--* The new keyboard callback function.

glutKeyboardFunc sets the keyboard callback for the *current window*.

* **void glutMainLoop(void);**

glutMainLoop enters the GLUT event processing loop.

* **void glFlush()**

forces any buffered openGL commands to execute.

**,**

* **void glClear(GL\_COLOR\_BUFFER\_BIT)**

To make the screen solid and black.

* **void glutIdleFunc(void(\*func)(void))**

A glutIdleFunc sets the global idle callback to func,so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received .

**4.2.2 User defined function**

* **void display()**

This is the routine which is used to display front page on the screen. It is

passed as an argument to glutDisplayFunc ().

* **void init()**

To initialize OpenGL. It sets orthogonal viewing volume.

* **void idle()**

This function helps to decrement the time each second as soon as the game starts and to move to the winning or to the timeout screen.

* **void box()**

This function helps to create a point object representing box which has to be moved .

* **void frontscreen(void)**

This function helps to display the front screen of the game.

* **void instruction(void)**

This function helps to display the instructions to be followed while playing the game.

* **void startscreen(void)**

This function helps to display the start screen where the actual game is being played.

* **void timeover(void)**

This function helps to display a message that the player could not finish the game in the given time and if he wishes to play again he can restart that game again.

* **void winscreen(void)**

This function helps to display the winning screen congratulating the player for successfully completing the game.

* **void keyboard(unsigned char, int, int)**

Keyboard interface function to enter into next screen.

**4.3 System requirements**

**4.3.1 Minimum Software Requirements**

1. Operating system: Windows XP/ Windows 7/ Windows 8

2. Compiler Used: Visual C++ 6.0

3. Language Used: C language

**4.3.2 Minimum Hardware Requirements**

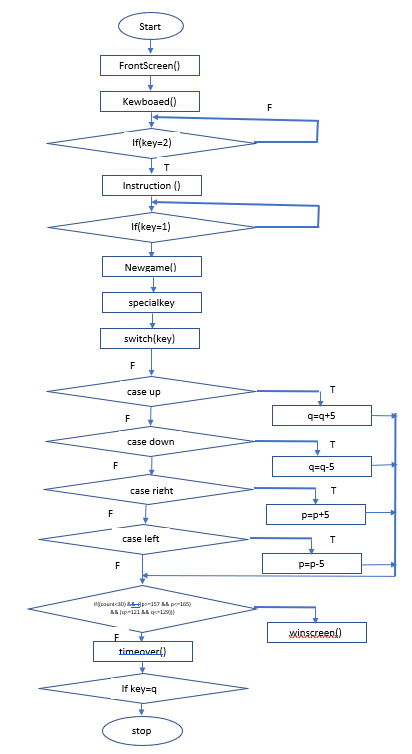
1. Main processor: PENTIUM III

2. Processor Speed: 800 MHz

3. RAM Size: 128 MB DDR

**4.4 Flow Chart**

The following diagram shows the flow chart of the project. It shows the flow of control throughout the program and the game. This is very helpful for the player to continue the game. By seeing this flow chart player can easily analyze the game.



keyboard()

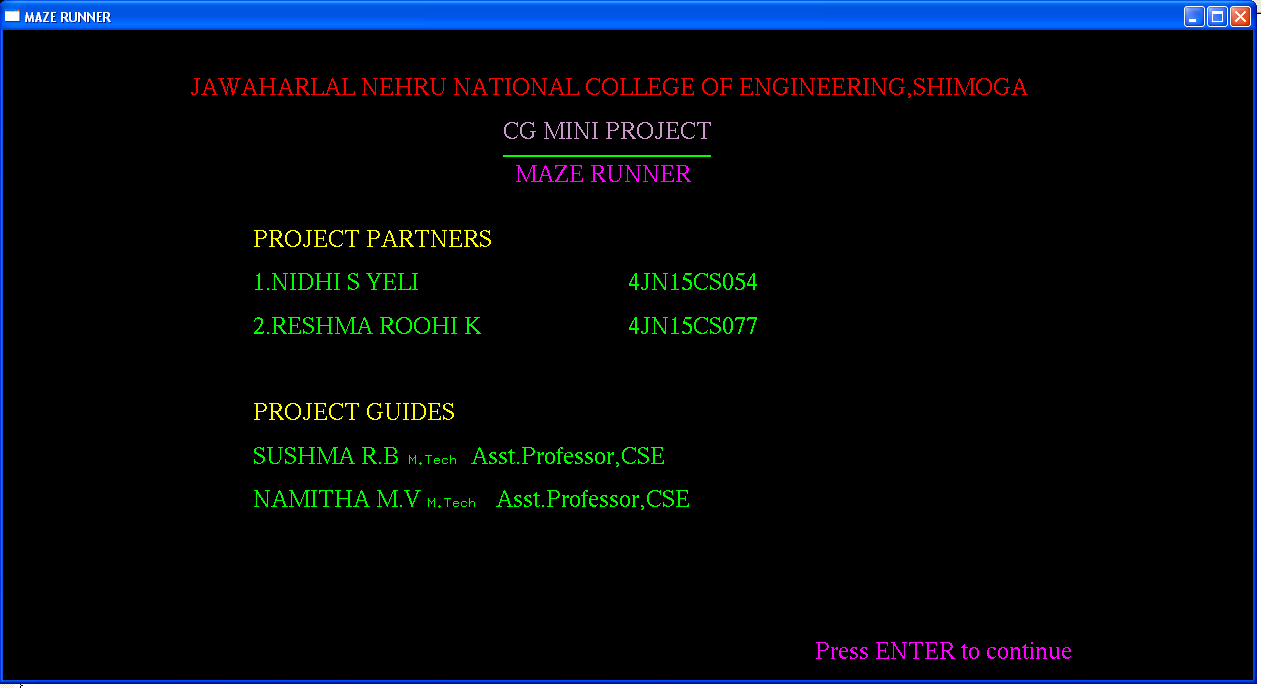
winscreen()

**Fig.4.1 Flow chart of Maze Runner.**

The above flow chart gives the step by step flow of execution about the game.

**CHAPTER 5**

**RESULTS AND SNAPSHOTS**

****

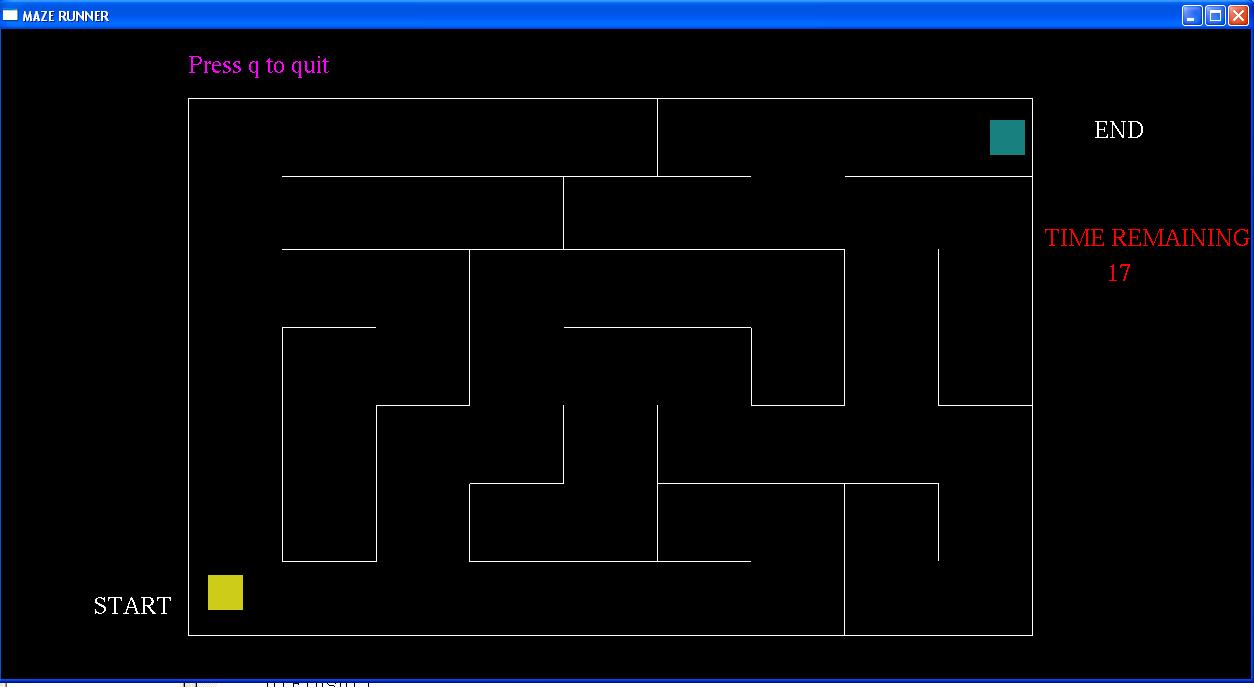
**Fig 5.1: Snapshot showing Front Sheet of the project**

The above snapshot shows front sheet of the project

****

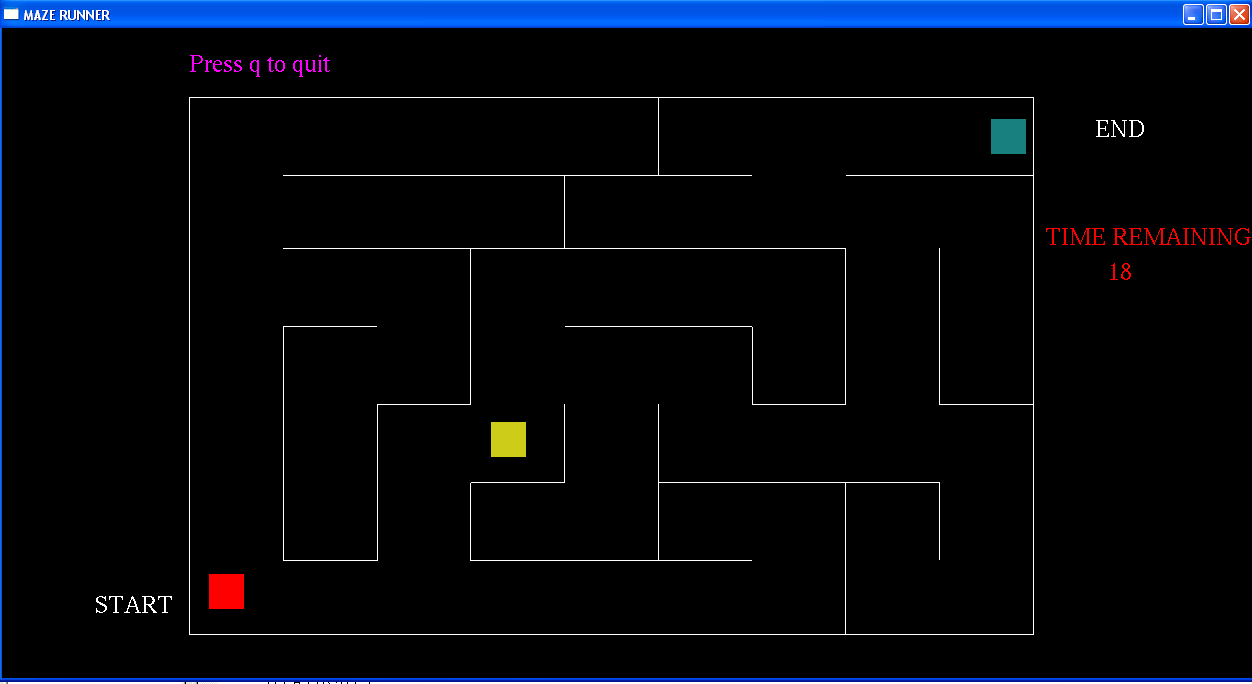
**Fig 5.2: Snapshot showing display options available**

The above snapshot showing three options play,instruction and exit.

****

**Fig 5.4: Initial position of box in the game**

The above snapshot shows the initial position of the game, where box is in the start position, pressing arrow keys will make box to move further.

****

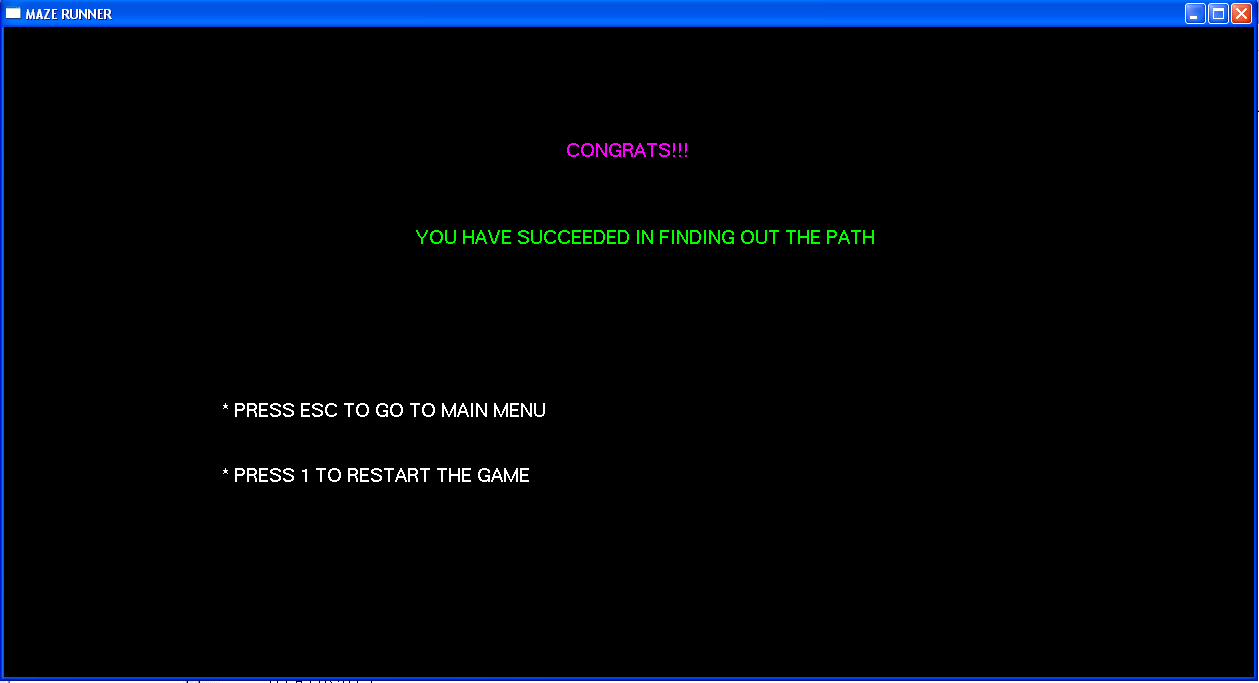
**Fig 5.5: Movement of the box**

The above screen shot shows the movement of the box.

****

**Fig 5.6: Game Over**

The above screen shot shows the options after game over

****

**Fig 5.7: Win**

The above screen shot shows the winning screen congratulating the player for completing the game successfully.

**CHAPTER 6**

**CONCLUSION AND FUTURE ENHANCEMENT**

The aim of this project is to show how to move the box from start to end .As the name itself indicates that, user has to solve the maze. The code has been written in OpenGL. The code contains the function that necessary to accomplish all the tasks required for this project. It works well with the windows platform. It can be further improved to provide better facilities such as shading and lighting effects and can be developed using 3D effect.

**REFERENCES**

[1] Edward Angel**,** Interactive Computer Graphics,5th Edition, Addison-Wesley, 2008

[2] F.S Hill Jr: Computer Graphics Using OpenGL,3rd Edition.

[3] Donald Hearn and Pauline Baker: Computer Graphics- OpenGL Version,3rd Edition.

[4] <http://www.opengl.org/documentation/books>.