VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Belgaum – 590014, Karnataka State, India

Mini-Project Entitled

“Wrap-Around Minesweeper”

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

For the academic year 2016-2017

Submitted by:

Abhijith C **(1MV14CS004)**

Mini-Project Carried out at

Sir M. Visvesvaraya Institute of Technology

Bangalore - 562157

Under Guidance of

Mrs. Monika Rani H G

Asst. Professor

SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

HUNASAMARANAHALLI BENGALURU – 562157

SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belagavi)

Bangalore – 562157

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the mini-project work entitled **"WRAP-AROUND MINESWEEPER"** is a bonafide work carried by **ABHIJITH C (1MV14CS004)** in partial fulfillment for the degree of **Bachelor of Engineering** in **Computer Science and Engineering** of the **Visvesvaraya Technological University, Belgaum** during the **academic year 2016 – 2017** in **Computer Graphics and Visualization Laboratory.** The mini-project has been hereby approved as it satisfies the academic requirements in respect of the mini-project work prescribed for the course of Bachelor of Engineering Degree.

……………………………………	……………………………………
Signature of the guide	**Signature of the HOD**
Mrs. Monika Rani H G.	**Prof. Dilip K. Sen**
Asst. professor, Dept. of CSE	**HOD, Dept. of CSE**
Sir MVIT	**Sir MVIT**

Examiners:

…………………………………..	…………………………………..
External Examiner	Internal Examiner

ACKNOWLEDGEMENT

The successful completion of any work depends upon the cooperation and help of many people and not just those who directly execute the work. It is difficult to express in words our profound sense of gratitude to those who helped us, but we make a humble attempt to do so.

I wish to place on record my sincere thanks to **Prof Dilip K. Sen, Head of the Department**, Computer Science and Engineering for encouragement and support.

We express our sincere gratitude to our internal guide **Mrs. Monika Rani H G., Asst. Professor in Computer Science, and Engineering Department** for giving us the valuable suggestions regarding the mini-project and for her constant guidance and encouragement, without which it would have been impossible to meet the high standards set by the college.

We also thank the members of the faculty of CSE department and other members, whose suggestions helped us during this mini-project.

Our heartfelt thanks to all the above-mentioned people who have contributed in the accomplishment of this mini-project.

- ABHIJITH C

1MV14CS004

ABSTRACT

This mini-project demonstrates a simple game of **Mine-Sweeper**. Minesweeper is a single-player puzzle video game. The objective of the game is to clear a square board containing hidden "mines" without detonating any of them, with help from clues about the number of neighboring mines in each field. The game is played by revealing squares of the grid by clicking each square. If a square containing a mine is revealed, the player loses the game. If no mine is revealed, a digit is instead displayed in the square, Indicating how many adjacent squares contain mines; if no mines are adjacent, the square becomes blank. The player uses this information to deduce the contents of other squares, and may either safely reveal each square or mark the square as containing a mine by placing a flag.

Our implementation of this game has a small twist which we call "**Wrap-Around**" where clicking on tiles on the corners of the map opens up fields on the opposite corners of the game area, hence making the game more challenging.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

The term "Computer Graphics" includes almost everything on computers that is not text or sound. Today nearly all computers use some graphics and users expect to control their computer through icons and pictures rather than just by typing. Computer graphics is the field of visual computing, where one utilizes computers both to generate visual images synthetically and to integrate or alter visual and spatial information sampled from the real world. The term computer graphics has several meanings:

- The representation and manipulation of pictorial data by a computer.
- The various technologies used to create and manipulate such pictorial data.
- The sub-field of computer science which studies for digitally synthesizing and manipulating visual content.

This field can be divided into several areas: real-time 3D rendering (often used in video games), video capture and video creation rendering, special effects editing (often used for movies and television), image editing and modelling (often used for engineering and medical purposes).

1.2 History

Computer graphics started with pen plotter model. We had Cathode Ray Tube Display showing the graphics. Each line drawn was a result of intense calculation which was a huge overhead a few years back.

The phrase "Computer Graphics" was coined in 1960 by William fetter, a graphic designer for Boeing. The field of computer graphics developed with the emergence of computer graphics hardware. Early projects like the whirlwind and SAGE Projects introduced the CRT as a viable display and interaction interface and introduced the light pen as an input device.

In 1959, the TX-2 computer was developed at MIT's Lincoln Laboratory. The TX-2 integrated a number of new man-machine interfaces. A light pen could be used to draw sketches on the computer using Ivan Sutherland's revolutionary Sketchpad software. Using a light pen, Sketchpad allowed one to draw simple shapes on computer screen, save them and even recall them later.

Many of the most important early breakthroughs in computer graphics research occurred at the University of Utah in the 1970s. A student by the name of Edwin Catmull saw computers as the natural progression of animation and they wanted to be part of the evolution. He created an animation of his hand opening and closing. The first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden surface algorithm.

In the 1980s, artist and graphics designer began to see the personal computer, particularly the Commodore Amiga and Macintosh, as a serious design tool, one that could save time and draw more accurately than other methods. In the late 1980s, SGI computers were used to create some of the first fully computer-generated short films at Pixar.

1.3 Applications Of Computer Graphics

- Computational biology
- Computational physics
- Computer-aided design
- Computer simulation
- Digital art
- Graphic design
- Info graphics
- Information visualization
- Scientific visualization

CHAPTER 2

LITERATURE SURVEY

2.1 OpenGL (Open Graphics Library):

OpenGL has become a widely accepted standard for developing graphics application. OpenGL is easy to learn, and it possesses most of the characteristics of other popular graphics system. It is top-down approach. OpenGL[8] is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives.

OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms.

The interface between the application program and the graphics system can be specified through that set of function that resides in graphics library. The specification is called the APPLICATION PROGRAM INTERFACE (API). The application program sees only the API and is thus shielded from the details both the hardware and software implementation of graphics library. The software driver is responsible for interpreting the output of an API and converting these data to a form that is understood by the particular hardware.

Most of our applications will be designed to access openGL directly through functions in three libraries. Function in the main GL library have name that begin with the letter gl and stored in the library. The second is the openGL utility Library (GLU). This library uses only GL function but contains codes for creating common object and viewing. Rather than using a different library for each system we used available library called openGL utility toolkit (GLUT). It used as #include <glut.h>

A graphics editor is a computer program that allows users to compose and edit pictures interactively on the computer screen and save them in one of many popular “bitmap” or “raster” a format such as TIFF, JPEG, PNG and GIF.

Graphics Editors can normally be classified as:

- 2D Graphics Editors.
- 3D Graphics Editors.

A 3D Graphics Editor is used to draw 3D primitives Rectangles, Circle, polygons, etc. and alter those with operations like cut, copy, paste. These may also contain features like layers and object precision etc.

3D Graphics Editor should include the following features:

- Facilities: Cursor Movement, Editing picture objects.
- Good User Interface: GUI / Toolbars / Icon based User Interface.

Computer Graphics is concerned with all aspects of producing pictures or images using a computer. A particular graphics software system called OpenGL, which has become a widely accepted standard for developing graphics applications.

The applications of computer graphics in some of the major areas are as follows

1. Display of information.
2. Design.
3. Simulation and Animation.
4. User interfaces.

Our mini-project titled “WRAP-AROUND MINESWEEPER” uses OpenGL software interface and develops 2D images. This project uses certain techniques like Transformation, sounds, display list, etc.

2.2 Problem Section Statement

Computer graphics is no longer a rarity. It is an integral part of all computer user interfaces, and is indispensable for visualizing 2D, 3D and higher dimensional objects. Creating 3D objects, rotations and any other manipulations are laborious process with graphics implementation using text editor. OpenGL provides more features for developing 3D objects with few lines by built in functions.

The geometric objects are the building blocks of any individual. Thereby developing, manipulating, applying any transformation, rotation, scaling on them is the major task of any image development.

Thereby we have put our tiny effort to develop 2D objects and perform different operations on them by using OpenGL utilities.

2.3 Existing System

The existing system involves computer graphics. Computer graphics started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computer themselves. It includes the creation, storage and manipulation of models and images of objects.[3]

These models include physical, mathematical, engineering, architectural and so on Computer graphics today is largely interactive –the user controls the contents, structure and appearance of objects and their displayed images by using input devices, such as keyboard, mouse or touch-sensitive panel on the screen. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television.

2.4 Proposed System

In proposed system, the OpenGL is a graphic software system designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using.[2]

OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of *geometric primitives* - points, lines, and polygons.

2.5 Objectives Of The Project

- Developing a package using computer graphics with OpenGL.
- To show that implementation of Depth is easier with OpenGL.
- Implementing certain technical concept like Translation, motion, and use of Idle Function.
- How to use Buffer algorithms to remove hidden surfaces.

CHAPTER 3

SYSTEM REQUIREMENT SPECIFICATION

3.1 User Requirements

- Easy to understand and should be simple.
- The built-in functions should be utilized to the maximum extent.
- OpenGL library facilities should be used.

3.2 Hardware Requirements

- Intel Pentium CPU 2.6 GHz or AMD Athlon 64 (K8) 2.6 GHz or higher
- 1 GB RAM or more
- Mouse
- Keyboard 108 standard
- Monitor resolution 800x600

3.3 Software Requirements

- OpenGL Tools (FreeGLUT for Linux systems)
- OpenGL Extension Wrangler Library (GLEW)
- Mesa OpenGL Tools
- Linux 32-Bit Operating System
- G++ compiler for the Linux platform

IDE: Emacs (with C-Make enabled for Linux)

CHAPTER 4

DESIGN AND IMPLEMENTATION

To design the 'Wrap-around Minesweeper Game' using the glut library, we need to understand various concepts, components and utility functions that are essential to implement/integrate the required visual (and audio) effects. Hence by using the following functions we design our project.

4.1 Design

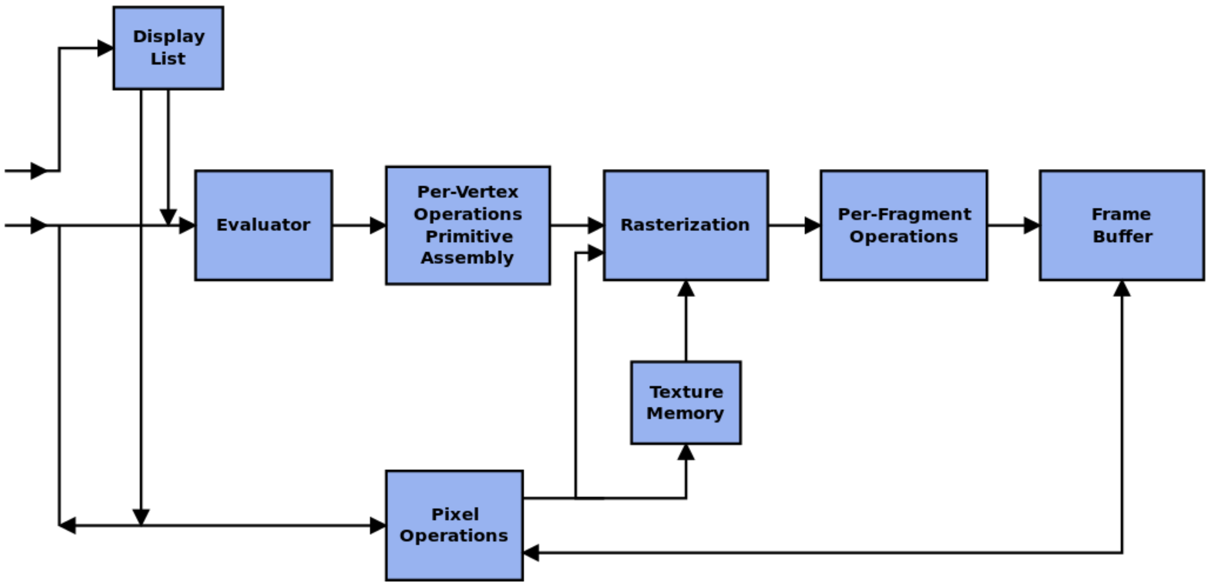

FIG 4.1 OPENGL PIPELINE

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware.[4]

The API is defined as a number of functions which may be called by the client program, alongside a number of named integer constants (for example, the constant GL_TEXTURE_2D, which corresponds to the decimal number 3553). Although the function definitions are superficially similar to those of the C programming language, they are language-independent. As such, OpenGL has many language bindings, some of the most noteworthy being the JavaScript binding WebGL (API, based on OpenGL ES 2.0, for 3D rendering from within a web browser); the C bindings WGL, GLX and CGL; the C binding provided by iOS; and the Java and C bindings provided by Android.

4.2 Header Files Used

#include <iostream>

iostream is the header in the C++ standard library that contains macro definitions, constants and declarations of functions and types used for various standard input and output operations.

#include <math.h>

math.h is a header file in the standard library of the C programming language designed for basic mathematical operation.

4.3 Simple Geometry

Void glBegin(glEnum mode): This function initiates a new primitive of type mode and starts the collection of vertices. Values of this mode include GL_POINTS, GL_LINE_STRIP and GL_QUADS.

Void glEnd(): Terminates a list of vertices.

glVertex2f(coordinates): This function defines the vertices of 2D figure with float as data type.

GlutInit(int argc, char *argv): Initialize GLUT. The arguments from main are passed in and can be used by the application.

glutCreateWindow(char*title): Create a Window on the display, the string title 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.

glutDisplayMode(unsigned int mode): Request 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(intwidth,int height): Specifies the initial height and width of the window in pixels.

glutInitWindowPosition(int x, int y): Specifies the initial position of top-left corner of the window in pixels.

glutDisplayFunc(void (*func)(void)): Registers the display function that is executed when the window needs to be redrawn.

glutPostRedisplay(): Requests that the display callback be executed after the current callback returns.

glutMainLoop(): Causes the program to enter an event processing loop. It should be the last statement in main.

4.4 Interaction

glutKeyboardfunc(void(*func)(unsigned char key,int x,int y)): Sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback.

glutMousefunc(void(*func)(int button, int state, int x,int y)): Sets the Mouse callback for the current window. When user clicks on something on the window, each click generates a state and a button value that is passed to this callback.

4.5 Transformations

glMatrixMode(GL_PROJECTION): Here the mode will be projection mode, specifies subsequent transformation matrix to an identity matrix.

glLoadIdentity(): Set the current transformation matrix to an identity stack corresponding to the current matrix mode.

4.6 Viewing.

glOrtho (left,right,bottom,top,near,far): It defines a 3 dimensional Orthographic view

4.7 Audio

system("play .. 2> /dev/null"): The system function calls the "play" LINUX command to play the specified audio file name mentioned in the parameters

4.8 Main Function

The main function must be called before any other GLUT/OpenGL calls. Execution of any program starts at main function. The main function gets argument c and argument v.

```
int main (int argc,char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
        glutInitWindowSize(Game::WIDTH * Drawer::CELL_WIDTH, Game::HEIGHT *
                                                        Drawer::CELL_HEIGHT);
        glutInitWindowPosition(100, 120);
        glutCreateWindow("Wrap-Around Minesweeper");
        glClearColor(0, 0, 0, 1);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0, Game::WIDTH * Drawer::CELL_WIDTH, Game::HEIGHT *
                                                Drawer::CELL_HEIGHT, 0, -1.0, 1.0);
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutKeyboardFunc(keyPressed);
        glutMouseFunc(mouse);
        glutMainLoop();
        return 0;
}
```

CHAPTER 5

SOURCE CODE

The entire project is divided into 3 files, main.cpp, gameMechanics.cpp and drawer.cpp. All these files must be linked together while compiling.

drawer.cpp → takes care of all the OpenGL based drawing functions implementations

gameMechanics.cpp → takes care of the rules that the game must follow during runtime.

1. drawer.cpp:

```
#include "drawer.hpp"
#include <GL/gl.h>
#include <GL/glut.h>
#include <math.h>
#include <iostream>
#include <unistd.h>
int boomOrNumberFlag = 1;
void Drawer::drawWelcomeScreen(int h, int w) //Draws Welcome Screen
{    renderBox(h, w, 3);
     glColor3f(0,0,0);
     glLineWidth(5.0);
     glColor3f(1.0f, 0.0f, 0.0f);
     drawStrokeText("Wrap-Around", (w/3)*CELL_WIDTH + 15, (h/3)*CELL_HEIGHT + 65, 0, 0.5, 0.35f);
     drawStrokeText("Minesweeper", (w/3)*CELL_WIDTH + 15, (h/3)*CELL_HEIGHT + 115, 0, 0.5, 0.35f);
     glColor3f(0.0f, 0.0f, 0.0f);
     drawStrokeText("By", (w/3)*CELL_WIDTH + 15, (h/3)*CELL_HEIGHT + 165, 0, 1.0, 0.25f);
     drawStrokeText("Akhil S", (w/3)*CELL_WIDTH + 15, (h/3)*CELL_HEIGHT + 205, 0, 0.8, 0.25f);
     drawStrokeText("Abhijith C", (w/3)*CELL_WIDTH + 15, (h/3)*CELL_HEIGHT + 245, 0, 0.8, 0.25f);
     glLineWidth(1.0);
     glutSwapBuffers();
}
void Drawer::drawMine(int x, int y) //Draw a Black Pentagon for a mine
{
       drawOpenedField(x, y); //Place a mine on the opened field
       glColor3f(0.0f, 0.0f, 0.0f);
       glBegin(GL_POLYGON);
       for (int i = 0; i < 5; ++i)
       {
              glVertex2f(x * CELL_WIDTH + CELL_WIDTH / 2 + 5.0f * cos(2 *3.1415926 * i / 5), y * CELL_HEIGHT
                                          + CELL_HEIGHT / 2 + 5.0f * sin(2 * 3.1415926 * i / 5));
       }
       glEnd();}
```

```
void Drawer::drawFlag(int x, int y)
{
        glColor3f(0.8f, 0.8f, 0.8f);
        glBegin(GL_QUADS);
        glVertex2f(x * CELL_WIDTH, y * CELL_HEIGHT);
        glVertex2f((x + 1) * CELL_WIDTH, y * CELL_HEIGHT);
        glVertex2f((x + 1) * CELL_WIDTH, (y + 1) * CELL_HEIGHT);
        glVertex2f(x * CELL_WIDTH, (y + 1) * CELL_HEIGHT);
        glEnd();
        glColor3f(0.0f, 0.0f, 0.0f);
        glBegin(GL_LINES);
        glVertex2f(x * CELL_WIDTH + CELL_WIDTH / 2, y * CELL_HEIGHT + 3);
        glVertex2f(x * CELL_WIDTH + CELL_WIDTH / 2, (y + 1) * CELL_HEIGHT - 3);
        glEnd();
        glColor3f(1.0f, 0.0f, 0.0f);
        glBegin(GL_TRIANGLES);
        glVertex2f(x * CELL_WIDTH + CELL_WIDTH / 2, y * CELL_HEIGHT + 3);
        glVertex2f(x * CELL_WIDTH + CELL_WIDTH / 2 - 3, y * CELL_HEIGHT + 3 + 3);
        glVertex2f(x * CELL_WIDTH + CELL_WIDTH / 2, y * CELL_HEIGHT + 3 + 3 + 3);
        glEnd();
}
void Drawer::drawClosedField(int x, int y)
{
        glColor3f(0.8f, 0.8f, 0.8f);
        glBegin(GL_QUADS);
        glVertex2f(x * CELL_WIDTH, y * CELL_HEIGHT);
        glVertex2f((x + 1) * CELL_WIDTH, y * CELL_HEIGHT);
        glVertex2f((x + 1) * CELL_WIDTH, (y + 1) * CELL_HEIGHT);
        glVertex2f(x * CELL_WIDTH, (y + 1) * CELL_HEIGHT);
        glEnd();
        glColor3f(1.0f, 1.0f, 1.0f);
        glBegin(GL_LINES);
        glVertex2f(x * CELL_WIDTH, y * CELL_HEIGHT);
        glVertex2f((x + 1) * CELL_WIDTH - 1, y * CELL_HEIGHT);
        glVertex2f(x * CELL_WIDTH, y * CELL_HEIGHT);
        glVertex2f(x * CELL_WIDTH, (y + 1) * CELL_HEIGHT - 1);
        glEnd();
        glColor3f(0.2f, 0.2f, 0.2f);
        glBegin(GL_LINES);
        glVertex2f((x + 1) * CELL_WIDTH - 1, y * CELL_HEIGHT);
        glVertex2f((x + 1) * CELL_WIDTH - 1, (y + 1) * CELL_HEIGHT - 1);
        glVertex2f(x * CELL_WIDTH, (y + 1) * CELL_HEIGHT - 1);
        glVertex2f((x + 1) * CELL_WIDTH - 1, (y + 1) * CELL_HEIGHT - 1);
        glEnd();
  }
  void Drawer::drawOpenedField(int x, int y, int neighbourMinesCount)
  {
            drawOpenedField(x, y);
        if (neighbourMinesCount > 0)
        {
              switch (neighbourMinesCount)
              {
                      case 1:
                           glColor3f(0.0f, 1.0f, 0.0f); //Green
                           break;
                      case 2:
                           glColor3f(0.0f, 0.0f, 1.0f); //Blue
                           break;
                      case 3:
                           glColor3f(1.0f, 0.0f, 0.0f); //Red
                           break;
                      case 4:
                           glColor3f(0.0f, 0.7f, 0.0f); //Celebi Green
                           break;
                      case 5:
                           glColor3f(0.5f, 0.4f, 0.0f); //Dragonite Orange
                           break;
                      case 6:
                           glColor3f(0.0f, 0.8f, 0.5f); //Grovyle Green
                           break;
                      case 7:
                           glColor3f(0.1f, 0.1f, 0.1f); //Onyx Grey
                           break;
                      case 8:
                           glColor3f(0.3f, 0.3f, 0.3f); //Geodude Grey
                           break;
              }
              glRasterPos2f(x * CELL_WIDTH + (15 - 9) / 2 + 1, (y + 1) * CELL_HEIGHT-1);
              glutBitmapCharacter(GLUT_BITMAP_9_BY_15, '0' +neighbourMinesCount);
        }
}
void Drawer::drawOpenedField(int x, int y)
{
        glColor3f(0.6f, 0.6f, 0.6f);
        glBegin(GL_QUADS);
        glVertex2f(x * CELL_WIDTH, y * CELL_HEIGHT);
        glVertex2f((x + 1) * CELL_WIDTH, y * CELL_HEIGHT);
        glVertex2f((x + 1) * CELL_WIDTH, (y + 1) * CELL_HEIGHT);
        glVertex2f(x * CELL_WIDTH, (y + 1) * CELL_HEIGHT);
        glEnd();
        //Dark Grey Borders for Depth
        glColor3f(0.2f, 0.2f, 0.2f);
        glBegin(GL_LINES);
        glVertex2f((x + 1) * CELL_WIDTH - 1, y * CELL_HEIGHT);
        glVertex2f((x + 1) * CELL_WIDTH - 1, (y + 1) * CELL_HEIGHT - 1);
        glVertex2f(x * CELL_WIDTH, (y + 1) * CELL_HEIGHT - 1);
        glVertex2f((x + 1) * CELL_WIDTH - 1, (y + 1) * CELL_HEIGHT - 1);
        glEnd();
}
void Drawer::drawStrokeText(const char*string, int x, int y, int z, int size, float scaleFactor)
{
          const char *c;
          glPushMatrix();
          glTranslatef(x, y,z);
          glScalef(scaleFactor,-scaleFactor,z);
          for (c=string; *c != '\0'; c++)
          {
                glutStrokeCharacter(GLUT_STROKE_ROMAN , *c);
                glutStrokeWidth(GLUT_STROKE_ROMAN, size);
          }
          glPopMatrix();
}
void Drawer::renderBox(int h, int w, int scale)
{
        glColor3f(0.8f, 0.8f, 0.8f);
        glBegin(GL_QUADS);
        glVertex2f((0.5*w/scale)*CELL_WIDTH, (0.5*h/scale)*CELL_HEIGHT);
        glVertex2f((2.5*w/scale)*CELL_WIDTH, (0.5*h/scale)*CELL_HEIGHT);
        glVertex2f((2.5*w/scale)*CELL_WIDTH, (2.5*h/scale)*CELL_HEIGHT);
        glVertex2f((0.5*w/scale)*CELL_WIDTH, (2.5*h/scale)*CELL_HEIGHT);
        glEnd();
        glColor3f(1.0f, 1.0f, 1.0f);
        glBegin(GL_LINES);
        glVertex2f((0.5*w/scale)*CELL_WIDTH, (0.5*h/scale)*CELL_HEIGHT);
        glVertex2f((2.5*w/scale)*CELL_WIDTH - 2, (0.5*h/scale)*CELL_HEIGHT);
        glVertex2f((0.5*w/scale)*CELL_WIDTH, (0.5*h/scale)*CELL_HEIGHT);
        glVertex2f((0.5*w/scale)*CELL_WIDTH, (2.5*h/scale)*CELL_HEIGHT - 2);
        glEnd();
        glColor3f(0.2f, 0.2f, 0.2f);
```

```
    glBegin(GL_LINES);
    glVertex2f((2.5*w/scale)*CELL_WIDTH - 2, (0.5*h/scale)*CELL_HEIGHT);
    glVertex2f((2.5*w/scale)*CELL_WIDTH - 2, (2.5*h/scale)*CELL_HEIGHT - 2);
    glVertex2f((0.5*w/scale)*CELL_WIDTH, (2.5*h/scale)*CELL_HEIGHT - 2);
    glVertex2f((2.5*w/scale)*CELL_WIDTH - 2, (2.5*h/scale)*CELL_HEIGHT - 2);
    glEnd();
}
void Drawer::gameOver(int h, int w)
{
    renderBox(h, w, 3);
    //Draw Game Over text on the box generated above
    glColor3f(0,0,0);
    glLineWidth(5.0);
    drawStrokeText("GAME OVER", (w/3)*CELL_WIDTH+5, (h/3)*CELL_HEIGHT + 60, 0, 0.5, 0.35f);
    glLineWidth(1.0);
    glLineWidth(3.0);
    drawStrokeText("Press R to restart", (w/3)*CELL_WIDTH + 5, (h/3)*CELL_HEIGHT + 100, 0, 0.2, 0.15f);
    drawStrokeText("Press Q to quit", (w/3)*CELL_WIDTH + 5, (h/3)*CELL_HEIGHT + 150, 0, 0.2, 0.15f);
    glLineWidth(1.0);
    glutSwapBuffers();
}
void Drawer::gameWon(int h, int w){
    renderBox(h, w, 3);
    //Draw Game Over text on the box generated above
    glColor3f(0,0,0);
    glLineWidth(5.0);
    drawStrokeText("GAME WON", (w/3)*CELL_WIDTH+5, (h/3)*CELL_HEIGHT + 60, 0, 0.5, 0.35f);
    glLineWidth(1.0);
    glLineWidth(3.0);

  drawStrokeText("Press R to restart", (w/3)*CELL_WIDTH + 5, (h/3)*CELL_HEIGHT + 100, 0, 0.2, 0.15f);
    drawStrokeText("Press Q to quit", (w/3)*CELL_WIDTH + 5, (h/3)*CELL_HEIGHT + 150, 0, 0.2, 0.15f);
    glLineWidth(1.0);
    glutSwapBuffers(); }
```

2. gameMechanics.cpp :

```
#include "gameMechanics.hpp"
#include "drawer.hpp"
#include <cstdlib>
#include <ctime>
#include <stdio.h>
#include <thread>
#include <unistd.h>
```

```
int initial = 0;
Drawer d;
int retFlag = 0, anotherFlag = 0;
int X, Y;
int winFlag = 0;
void boom()
{
    system("play boom.wav 2> /dev/null");
}
void win()
{
        system("play ToToDo.wav 2> /dev/null");
}
Game::Game()
{       gameOver = false;
        srand ( time(NULL) );
        for (int y = 0; y < HEIGHT; ++y)
          for (int x = 0; x < WIDTH; ++x)
          {
                field_[x][y].state = CLOSED;
                field_[x][y].hasMine = false;
          }
    for (int i = 0; i < 30; ++i)
    {
          int x, y;
          do
          {
                x = rand() % WIDTH;
                y = rand() % HEIGHT;
          }
          while (field_[x][y].hasMine);

          field_[x][y].hasMine = true;
    }
}
bool Game::gridSweeper(int x, int y)
{
        if (!(field_[x][y].hasMine || (field_[x][y].state == FLAG) ))
        {
                        int neighbourMinesCount = 0;
            for (int yy = y - 1; yy <= y + 1; ++yy){
                        for (int xx = x - 1; xx <= x + 1; ++xx)
                {
                    if ((xx == x && yy == y) || xx < 0 || xx >= WIDTH || yy < 0 || yy >= HEIGHT)
                        continue;
                    if (field_[xx][yy].hasMine)
                        ++neighbourMinesCount;
                }
            }
            field_[x][y].state = OPENED;
            d.drawOpenedField(x, y, neighbourMinesCount);
            return true;
    }
    else return false;
}
bool Game::gameWin()
{
      int count = 0;
      for(int x = 0; x < WIDTH; x++)
      {
                  for(int y = 0; y < HEIGHT; y++)
                  {
                      if(field_[x][y].hasMine && (field_[x][y].state == FLAG))
                          count++;
                  }
      }
      if(count == 30)
                  return true;
      else
                  return false;
}
void Game::draw()
{     int boomFlag = 0;
      if(gameWin())
      {
                  if(!winFlag){
                      std::thread t1(win);
                      t1.join();
                      winFlag = 1;
                  }
                  gameWonFlag = true;
                  gameOver = true;
      }
      if(!gameOver)
      {           for (int y = 0; y < HEIGHT; ++y)
              for (int x = 0; x < WIDTH; ++x)
```

```
                {
                        switch (field_[x][y].state)
                        {
                                case CLOSED:
                                d.drawClosedField(x, y); //Draw a closed field
                                      break;
                            case OPENED:
                                  if (!field_[x][y].hasMine)
                                {
                                    int neighbourMinesCount = 0;
                                    for (int yy = y - 1; yy <= y + 1; ++yy){
                                                for (int xx = x - 1; xx <= x + 1; ++xx)
                                                      {
                                                    if ((xx == x && yy == y) || xx < 0 || xx >= WIDTH
                                                                                              || yy < 0 || yy >= HEIGHT)
                                                                              continue;
                                                          if (field_[xx][yy].hasMine)
                                                                  ++neighbourMinesCount;
                                                          }
                                          }
                                                              d.drawOpenedField(x, y, neighbourMinesCount);

                                                      if(anotherFlag)
                        {
                                    int depth = 2;
                                    for(int k = 1; k <= depth;k++){
                                    if(X > 0 && !gridSweeper(X-k, Y)) break;
                                if(Y > 0 && !gridSweeper(X, Y-k)) break;
                                if(X < WIDTH && !gridSweeper(X+k, Y))
                                                      break;
                                    if(Y < HEIGHT && !gridSweeper(X, Y+k)
                                                      break;
                                    if(X > 0 && Y > 0 && !gridSweeper(X-k, Y-k))
                        break;
                                    if(X < WIDTH && Y < HEIGHT &&!
                                              gridSweeper(X+k, Y+k)) break;
                                    }
                              anotherFlag = 0;
                            }
              }
              else
              {
                        if(!boomFlag){
```

```
                        std::thread t1(boom);
                        t1.join();
                        boomFlag = 1;
                  }
                  for (int j = 0; j < HEIGHT; ++j)
                        for (int i = 0; i < WIDTH; ++i)
                              if (field_[i][j].hasMine ){
                                    field_[i][j].state = OPENED;
                                    d.drawMine(i, j);
                              }
                              gameOver = true;
         }
        break;
        case FLAG:
           d.drawFlag(x, y);
           break;
  }
}
if (!initial)
{
          initial = 1;
          Drawer d;
         d.drawWelcomeScreen((int)HEIGHT, (int)WIDTH);
        sleep(2);
}
}
else{
         for (int j = 0; j < HEIGHT; ++j)
              for (int i = 0; i < WIDTH; ++i)
              {
                        if (field_[i][j].hasMine && !gameWonFlag)
                                    {
                                    field_[i][j].state = OPENED;
                                    d.drawMine(i, j);
                                   }
                                   else if (field_[i][j].hasMine && gameWonFlag)
                                   {
                                    d.drawFlag(i, j);
                                   }
                                   else
                                   {
                                    d.drawClosedField(i, j);
                                   }
```

```
                                    }
                            sleep(1);
                            if(!gameWonFlag)
                            {
                                    d.gameOver((int)HEIGHT, (int)WIDTH);
                            }
                            else
                            {
                                    d.gameWon((int)HEIGHT, (int)WIDTH);
                            }
        }
}
void Game::markFlag(int x, int y)
{
        switch (field_[x][y].state)
    {
            case OPENED:
                break;
            case CLOSED:
                field_[x][y].state = FLAG;
                break;
            case FLAG:
                field_[x][y].state = CLOSED;
                break;
    }
}
void Game::open(int x, int y)
{
        if(!((field_[x][y].state == FLAG)||(field_[x][y].state == OPENED)))
        {
                            field_[x][y].state = OPENED;
                            anotherFlag = 1;
                            X = x;
                            Y = y;
        }
}
```

3. main.cpp :

```
#include "drawer.hpp"
#include "gameMechanics.hpp"
#include <GL/glut.h>
#include <unistd.h>
Game game;
void display()
{     glClear(GL_COLOR_BUFFER_BIT);
      game.draw();
      glutSwapBuffers();       }
void keyPressed (unsigned char key, int x, int y) {
      if (key == 'q') {
            exit(0);    }
      else if(key == 'r'){ game = Game(); }}
void reshape(int w, int h)
{     glutReshapeWindow(Game::WIDTH * Drawer::CELL_WIDTH, Game::HEIGHT * Drawer::CELL_HEIGHT); }
void mouse(int button, int state, int x, int y) // Mouse functions here..
{     if (state == GLUT_UP)
      {      switch (button)
            {    case GLUT_LEFT_BUTTON:    //Open that tile
                        game.open(x / Drawer::CELL_WIDTH, y / Drawer::CELL_HEIGHT);
                        break;
                  case GLUT_RIGHT_BUTTON: //Place a Flag here
                        game.markFlag(x/Drawer::CELL_WIDTH,y/ Drawer::CELL_HEIGHT);
                        break;
            }    glutPostRedisplay(); }    }
int main(int argc, char **argv)
{     glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
      glutInitWindowSize(Game::WIDTH * Drawer::CELL_WIDTH, Game::HEIGHT *Drawer::CELL_HEIGHT);
      glutInitWindowPosition(100, 120);
      glutCreateWindow("Wrap-Around MineSweeper");
      glClearColor(0, 0, 0, 1);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(0, Game::WIDTH * Drawer::CELL_WIDTH, Game::HEIGHT * Drawer::CELL_HEIGHT, 0, -1.0, 1.0);
      glutDisplayFunc(display);
      glutReshapeFunc(reshape);
      glutKeyboardFunc(keyPressed);
      glutMouseFunc(mouse); //Call the mouse function here
      glutMainLoop();          }
```

CHAPTER 6

SNAPSHOTS

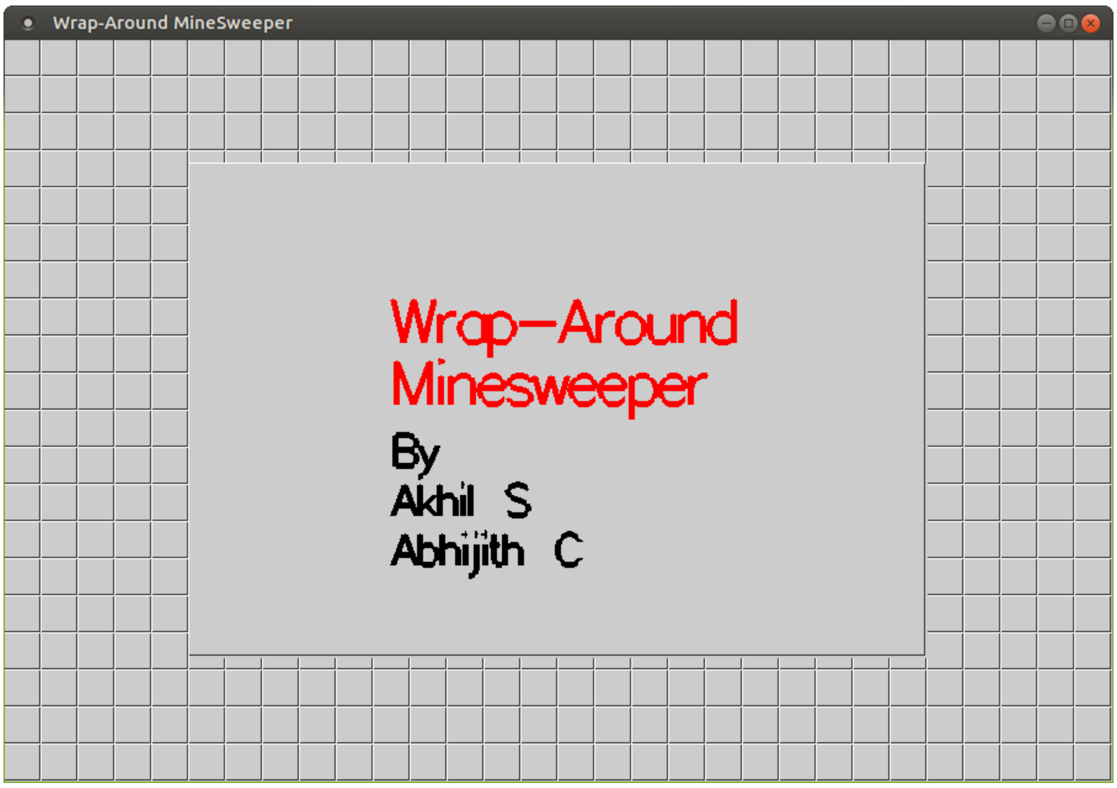

FIG 6.1 INITIAL SCREEN OF THE GAME

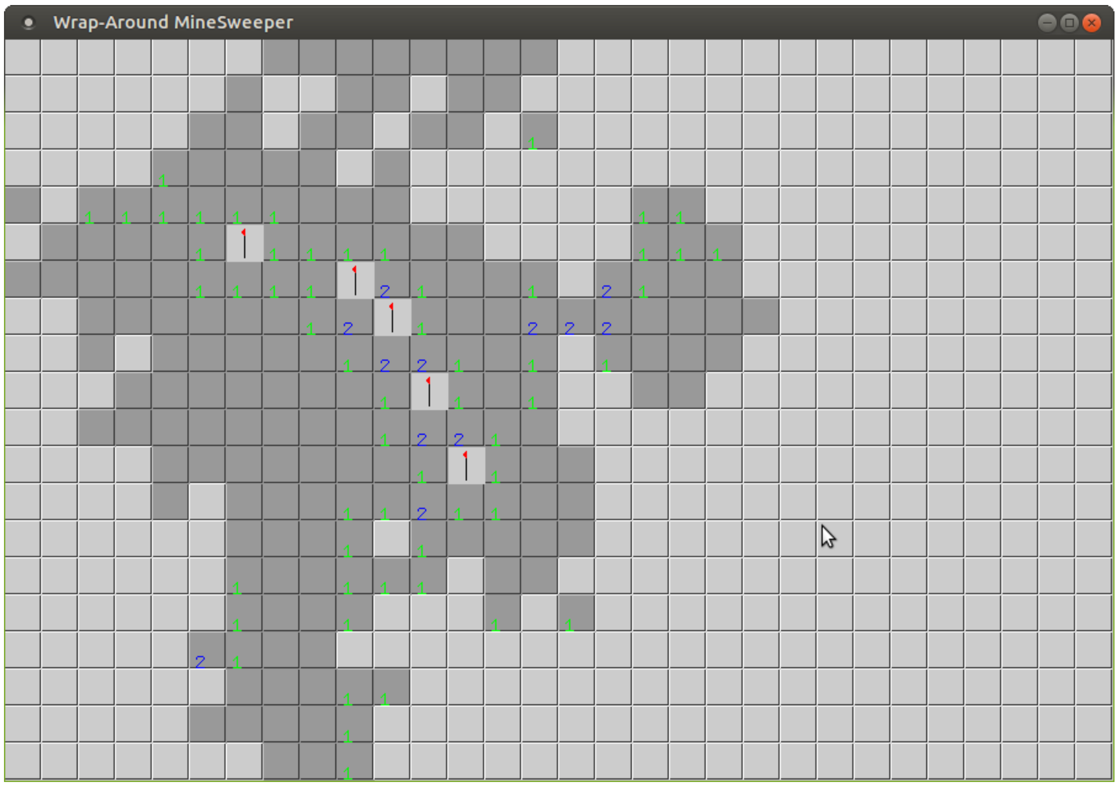

FIG 6.2 GAMEPLAY

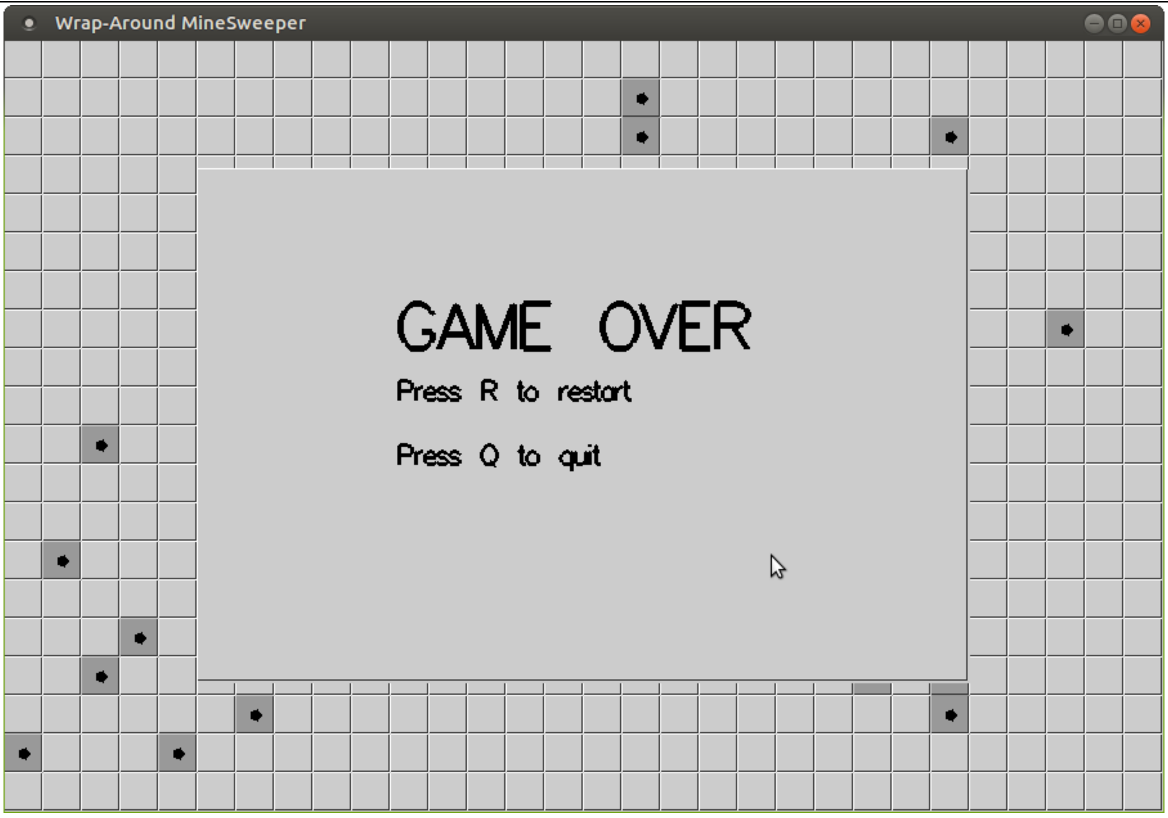

FIG 6.3 GAME-OVER SCREEN

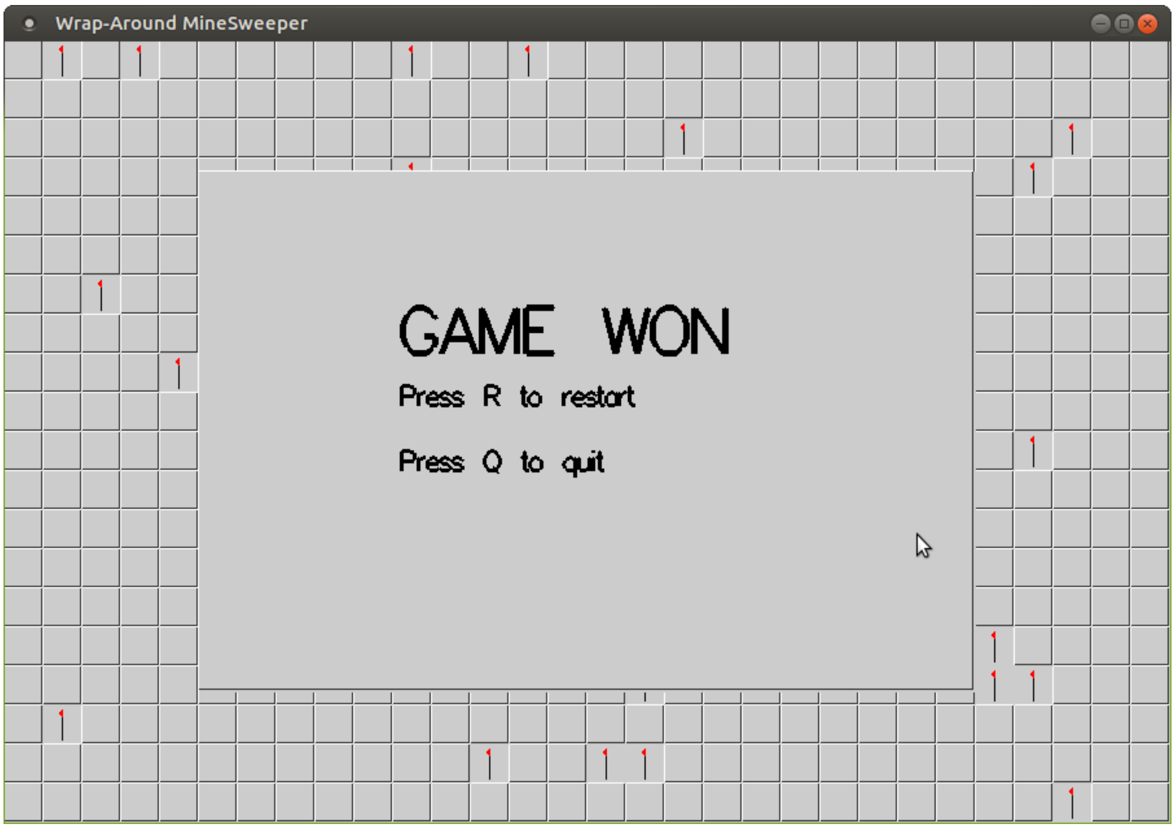

FIG 6.4 GAME-WON SCREEN

CHAPTER 7

CONCLUSION

An attempt has been made to develop an OpenGL graphics package, which meets necessary requirements of the users successfully. It enables us to learn about the basic concept in OPENGL graphics and graphics and know standard library graphics function and to explore some other function. OpenGL graphics is a huge library which consists of numerous functions.

The various shapes at lower level or to simulate any real thing animation etc. at high level. This project has given us an insight into the use of Computer graphics. As we had to use many built-in and user defined functions, we have managed to get a certain degree of familiarity with these functions and have now understood the power of these functions. We were able to comprehend the true nature of the most powerful tool graphics in OpenGL and have understood the reason why graphics is so powerful for programmers.

We can now converse with the certain degree of confidence about graphics in openGL. Finally, we have implemented this mini projection "Wrap-Around Minesweeper Game" using openGL package. We would like to end by saying that doing this graphics project has been a memorable experience in which we have learned a lot, although there is a scope for further improvement. We got to know a lot of different applications of OPENGL while doing this project.

FUTURE WORK

Scope of further improvement:

- Various lightening effects can be implemented show shadows.
- The game can include 3D graphics and support for material properties.
- Explosion effects can be shown for the bombs.
- Celebration effects by showing fireworks can be done for winning.

CHAPTER 8

BIBLIOGRAPHY

Books:

The books that helped us in implementing this project are as follows:

- **[1] OpenGL Game Development by Example**

  Robert Madsen and Stephen Madsen, Packt Publishing Limited, 2016

- **[2] Computer Graphics** (OpenGL Version)

  Donald Hearn and Pauline Baker, 2nd edition, Pearson Education, 2003

- **[3] Interactive Computer Graphics**

  Edward Angel, 5th edition, Addison Wesley, 2009

Reference Websites:

[4] freeglut.sourceforge.net/docs/api.php

[5] www.opengl.com

[6] www.learnopengl.com

[7] www.tutorialspoint.com

[8] https://en.wikipedia.org/wiki/OpenGL

Link to the Entire project : https://github.com/Akhilsudh/MineSweeper-CG-Project