VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI-590014

PROJECT ENTITLED

"CLASSICAL SNAKES GAME"

For the academic year 2016-2017
Submitted by:

MAYANK METHA D (1MV14CS054)

Project carried out at
Sir M. Visvesvaraya Institute of Technology
Bengaluru-562157.

Under the Guidance of

Mrs. Monika Rani H G
Asst.Professor, Dept.of.CSE
Sir. MVIT, BENGALURU.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Sir M. Visvesvaraya Institute of Technology
Hunasamaranahalli, Bengaluru-562157.

SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY BENGALURU-562157.

(Affiliated to Visvesvaraya Technological University, Belagavi)
Bengaluru-562157

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work entitled **"CLASSICAL SNAKES GAME"** is a bonafide work carried out by **MAYANK METHA D (1MV14CS054)**, in partial fulfillment for the award of Degree of **Bachelor of Engineering** in **Computer Science Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2016-2017 in **Computer Graphics and Visualization Laboratory**. The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the course of Bachelor of Engineering Degree.

Signature of the Guide

Mrs. Monika Rani H G
Asst. prof, Dept of CSE
Sir MVIT

Signature of the HOD

Prof. Dilip K Sen
HOD, Dept of CSE
Sir MVIT

External Examiner

Internal Examiner

ACKNOWLEDGEMENT

I am grateful to Prof. Dilip K Sen, Head of Department of CSE, Sir M Visvesvaraya Institute of Technology, Bengaluru for having encouraged me undertake this project.

I wish to express my deep sense of gratitude to our guide Mrs. Monika Rani H G, Asst. Prof. of Department of CSE, Sir M Visvesvaraya Institute of Technology, Bengaluru for her constant guidance and encouragement, without which it would have been impossible to meet the high standards set by the college

I like to thank all the other staff of the Department of CSE, Sir M Visvesvaraya Institute of Technology, Bengaluru for constantly pushing me to learn new concepts as quickly as possible.

I like to thank my parents, brother and relatives for letting me achieve my dreams by giving support and advices.

Lastly, I like to thank my friends for their valuable suggestions.

-Mayank Metha D

ABSTRACT

The “CLASSICAL SNAKE GAME” is a port of Nokia’s Snake II using OpenGL for popular OS like Windows 10, Apple MacOS and Linux. The game mechanics are as close to the native game. The game tries to mimic a retro 8-bit DOS game look and feel. Concept is simple food appears at random location. Snake is made to move over the food. The snake grows when it eats the food. Game finishes when snake eats itself or hits the wall.

INDEX

LIST OF FIGURES

Chapter 1

INTRODUCTION

The ancient Chinese proverb, “a picture is worth ten thousand words” became a cliché in our society. Graphics provides one of the most natural way of communication with the computer, since our highly developed 2D and 3D pattern recognition ability allows us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics thus permits extensive, high bandwidth user computer interaction. This significantly enhances our ability to understand data, to perceive trends and to visualize real and imaginary objects, indeed to create a “virtual world” that we can explore from arbitrary points and views. It makes communication more efficient, graphics makes possible higher quality and more precise results or products, greater productivity, and lower analysis and design cost.

OpenGL is a premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry’s most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

Chapter 2

LITERATURE SURVEY

2.1 EARLY GRAPHIC SYSTEMS

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.

2.2 OPENGL

OpenGL is a standard specification defining cross platform API for writing applications that produce 2D and 3D graphics. It contains multiple different function calls that help develop complex graphics with help of simple primitives. Developed by Silicon Graphics Inc. in 1992. Now its managed by the nonprofit technology consortium, the Khronos Group.

OpenGL has a set of library that help in various functions. They are GL, GLU and GLUT. OpenGL Library or GL provides a powerful yet primitive set of commands. OpenGL Utility Library or GLU contains several routines that help setting up matrices for specific viewing orientation, projections and surface rendering. OpenGL Utility Toolkit Library or GLUT contains routines that help in windowing functions and is system independent.

2.3 EXISTING SYSTEM

The existing graphics systems were the graphics header in C/C++. These graphics system are not system independent. Moreover, the underlying hardware knowledge is important for proper working of the code. Moreover, only 2D graphics were supported. Complex graphics concepts like camera position, shading, 3D graphics, material properties were absent.

2.4 PROPOSED SYSTEM

To achieve 3D graphics effects, OpenGL software was made. Moreover, system hardware independence and cross platform support OpenGL became famous. OpenGL is more streamlined than other graphics system APIs. The concept of building from primitives made it widely accepted by developers. It even supports animations, function driven events, callback functions. The transformation functions provide a more powerful ability to graphic coders to design their dreams digitally.

Chapter 3

SYSTEM REQUIREMENT SPECIFICATIONS

3.1 HARDWARE REQUIREMENTS

Minimum hardware specifications are

- Processor: Intel i3 4th Gen or better
- GPU: NVidia GT750M or better or Intel Iris Pro or better
- RAM: 1GB or more
- HDD: 60GB or more at 5400 RPM or better
- Keyboard: US English QWERTY Keyboard
- Mouse: Not Required
- Monitor: 800 x 600 or better

3.2 SOFTWARE REQUIREMENTS

Minimum software specifications are

- OS: Ubuntu 16.06(Linux 4.6) or MacOS/OSX 10.7 or Windows 10 or better
- Tools, IDE, Compilers:
    - freeglut3-dev, freeglut3, g++, C++11 for Linux
    - XCode 8.3, C++11 for MacOS/OSX
    - Visual Studio 2017, glut files, Visual C++ latest one for Windows
- Latest NVidia and/or Intel Drivers for GPU

Chapter 4

DESIGN

4.1 STATEMENT OF PROBLEM

Snake is the common name for a videogame concept where the player maneuvers a line which grows in length, with the line itself being a primary obstacle. The concept originated in the 1976 arcade game “Blockade” developed and published by Gremlin. After a variant was preloaded on Nokia mobile phones in 1998, there was a resurgence of interest in the snake concept.

4.2 OBJECTIVE OF THE PROBLEM

The player controls a dot, square or an object on a board plane. As it moves forward it leaves a trail behind, resembling a moving snake. The player attempts to eat items by running into them with the head of the snake. Each item eaten makes the snake longer, so controlling is progressively more difficult. The player losses when it runs into the wall or over itself.

As the items eaten by snake the score increases. After a certain score threshold, the snake moves faster making maneuvering more difficult. There are grid lines to help the player where and when to maneuver the snake. Score has no limit.

Chapter 5

IMPLEMENTATION

5.1 PLATFORM DEPENDENT HEADERS (PLATFORM.H)

```
#ifndef _PLATFORMS_H
#define _PLATFORMS_H
#ifdef __APPLE__
//APPLE macOS 10.7 upwards
#include <OpenGL/OpenGL.h>
#include <GLUT/glut.h>
#elif defined _WIN32
//WINDOWS 32bit only
#include<GL/glut.h>
#elif defined __linux__
//freeGlut on linux
#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>
#endif
#endif
```

5.2 SHARED HEADERS (GAME.H)

```
//define header
#ifndef _GAME_H
#define _GAME_H
//include headers
#include"platforms.h"
#include<stdio.h>
#include<stdlib.h>
#include<cstring>
#include<cmath>
#include<time.h>
//constants
#define ROWS 40 //yaxis
#define COLUMNS 40 //xaxis
#define FPS 12 //frame per seconds
#define MIN_FPS 5 //minimum FPS
//directions numbers
#define UP 1
#define RIGHT 2
#define DOWN 3
#define LEFT 4
//max length of snake
#define MAX_LEN 60;
//colors
typedef float colors[3];
const colors GREY = { 0.5,0.5,0.5 };
const colors WHITE = { 1.0,1.0,1.0 };
const colors RED = { 1.0,0.0,0.0 };
const colors GREEN = { 0.0,1.0,0.0 };
const colors DGREEN = { 0.0,0.8,0.0 };
const colors AQUAMARINE = { 0.498, 1.000, 0.831 };
const colors CYAN = { 0.0,1.0,1.0 };
const colors MAGENTA = { 1.0,0.0,1.0 };
const colors YELLOW = { 1.0,1.0,0.0 };
```

```
const colors DORANGE = { 1.000,0.549,0.000 };
//sharable functions
void initGame();
void initGrid(int, int);
void drawWall();
void drawGrid();
void drawSnake();
void drawFood();
void makeGameOver();
void random(int&, int&, int&);
void renderWelcome();
void renderGame();
void renderPause();
void renderExit();
void turnUp();
void turnDown();
void turnRight();
void turnLeft();
void autoPlay();
#endif
```

5.3 GAME ENGINE (GAME.CPP)

```
//header
#include"game.h"
//variables
int gridX, gridY; //grid lines
extern short sDir; //snake motion animation
extern bool sDirFlag; //snake direction keyboard flag
int snake_length;//snake length
int posX[60], posY[60]; //snake initial position
extern bool gameOver; //gameOver flag
extern bool gamePause; //gamePause flag
bool food; //place food or not
int foodX, foodY; // food position
int colorIndex; // food color
extern int score; //score
char text[50]; //score string
extern int fps; //animations frame rate
extern bool testFlag; //testing flag
extern bool gridFlag; //show hide grid flag
//init game variables
void initGame() {
   fps = MIN_FPS;
   score = 0;
   sDir = RIGHT;
   sDirFlag = true;
   snake_length = 5;
   for (int i = 0; i < snake_length; i++) {
      posX[i] = 20;
      posY[i] = 20 - i;
   }
   food = true;
   testFlag = false;
   gridFlag = true;
}
//initalize the grid
void initGrid(int rows, int columns) {
   gridX = columns;
   gridY = rows;
```

```
}
//draw cell outline
void drawCell(int x, int y, float thickness,const float* color) {
   glColor3fv(color);
   glLineWidth(thickness);
   glBegin(GL_LINE_LOOP);
   glVertex2d(x, y);
   glVertex2d(x + 1, y);
   glVertex2d(x + 1, y + 1);
   glVertex2d(x, y + 1);
   glEnd();
}
//draw grid when grid flag is active
void drawGrid() {
   if(gridFlag) {
      for (int x = 0; x < gridX; x++) {
         for (int y = 0; y < gridY; y++) {
            if (!gamePause && !gameOver && !(x == 0 || x == gridX - 1 || y == 0 || y == gridY - 1))
               drawCell(x, y, 1.0, GREY);
         }
      }
   }
}
//draw wall
void drawWall() {
   for (int x = 0; x < gridX; x++) {
      for (int y = 0; y < gridY; y++) {
         if (x == 0 || x == gridX - 1 || y == 0 || y == gridY - 1)
            drawCell(x, y, 3.0, RED);
      }
   }
}
//draw food
void drawFood() {
   //get new location
   if (food) {
      random(foodX, foodY, colorIndex);
      //is food at same loc with snake body then redraw at new location
      for (int i = 0; i < snake_length; i++) {
         for (int j = 0; j < snake_length; j++) {
            if (foodX == posX[i] && foodY == posY[j])
               drawFood();
         }
      }
   }
   food = false;
   //food color
   switch(colorIndex) {
      case 0:
         glColor3fv(AQUAMARINE);
         break;
      case 1:
         glColor3fv(MAGENTA);
         break;
      case 2:
         glColor3fv(CYAN);
         break;
      case 3:
         glColor3fv(YELLOW);
         break;
```

```
            case 4:
                glColor3fv(DORANGE);
                break;
        }
        //draw food rectangle
        glRectd(foodX, foodY, foodX + 1, foodY + 1);
        //testing only
        if (testFlag)
            autoPlay();
    }
    //3 random number generator
    void random(int &x, int &y, int &colorIndex) {
        //range
        int minX = 1;
        int maxX = gridX - 2;
        int minY = 1;
        int maxY = gridY - 2;
        x = minX + rand() % (maxX - minX);
        y = minY + rand() % (maxY - minY);
        colorIndex = rand()%5;
    }
    //draw snake
    void drawSnake() {
        if (!gamePause) {
            //body location without head
            for (int i = snake_length - 1; i > 0; i--) {
                posX[i] = posX[i - 1];
                posY[i] = posY[i - 1];
            }
            //snake direction change on key input and enable keyboard
            switch (sDir) {
                case UP:
                    posY[0]++;
                    sDirFlag = true;
                    break;
                case DOWN:
                    posY[0]--;
                    sDirFlag = true;
                    break;
                case RIGHT:
                    posX[0]++;
                    sDirFlag = true;
                    break;
                case LEFT:
                    posX[0]--;
                    sDirFlag = true;
                    break;
            }
        }
        //draw body segments
        for (int i = 0; i < snake_length; i++) {
            if(i % 2 == 0)
                glColor3fv(DGREEN);
            else
                glColor3fv(GREEN);
            glRectd(posX[i], posY[i], posX[i] + 1, posY[i] + 1);
        }
        if (!gamePause) {
            //collision detection with border
            if (posX[0] == 0 || posX[0] == gridX - 1 || posY[0] == 0 || posY[0] == gridY - 1) {
```

```
            glColor3fv(RED);
            glRectd(posX[0], posY[0], posX[0] + 1, posY[0] + 1);
            makeGameOver();
        }
        //collison with own body
        for (int i = 1; i < snake_length; i++) {
            if (posY[0] == posY[i] && posX[0] == posX[i]) {
                glColor3fv(RED);
                glRectd(posX[i], posY[i], posX[i] + 1, posY[i] + 1);
                makeGameOver();
            }
        }
        //collision with food
        if (posX[0] == foodX && posY[0] == foodY) {
            snake_length++;
            if (snake_length >= 60)
                snake_length = MAX_LEN;
            food = true;
            score++;
            if(score == 10)
                gridFlag = false;
        }
    }
}
//Testing AI for snake without self collision avoidance
void moveSnakeOnNoSelfCollision() {
    int xdiff = foodX - posX[0]; //distance between snake and food on xaxis
    int ydiff = foodY - posY[0]; //distance between snake and food on yaxis
    if (abs(xdiff) >= abs(ydiff)) {
        if (xdiff >= 0) {
            if (sDir != LEFT)
                turnRight();
            else {
                if (posX[0] != 1 || posY[0] != (gridY - 2))
                    turnUp();
                else if (posX[0] != (gridX - 2) || posY[0] != 1)
                    turnDown();
            }
        }
        else if (xdiff < 0) {
            if (sDir != RIGHT)
                turnLeft();
            else {
                if (posX[0] != 1 || posY[0] != (gridY - 2))
                    turnUp();
                else if (posX[0] != (gridX - 2) || posY[0] != 1)
                    turnDown();
            }
        }
    }
    else {
        if (ydiff >= 0) {
            if (sDir != DOWN)
                turnUp();
            else {
                if (posY[0] != 1 || posX[0] != (gridX - 2))
                    turnRight();
                else if (posY[0] != (gridY - 2) || posX[0] != 1)
                    turnLeft();
            }
```

```
        }
        else if (ydiff < 0) {
          if (sDir != UP)
            turnDown();
          else {
            if (posY[0] != 1 || posX[0] != (gridX - 2))
              turnRight();
            else if (posY[0] != (gridY - 2) || posX[0] != 1)
              turnLeft();
          }
        }
      }
    }
    //Testing AI for snake self collision avoidance
    void autoPlay() {
      unsigned int collidable = 0;
      const int up = 1;
      const int down = 2;
      const int right = 4;
      const int left = 8;
      for(int i = 0; i < (snake_length - 1); i++) {
        if((posX[0] + 1) == posX[i] && posY[0] == posY[i] && sDir!= LEFT)
          collidable |= right;
        if((posX[0] - 1) == posX[i] && posY[0] == posY[i] && sDir != RIGHT)
          collidable |= left;
        if((posY[0] + 1) == posY[i] && posX[0] == posX[i] && sDir != DOWN)
          collidable |= up;
        if((posY[0] - 1) == posY[i] && posX[0] == posX[i] && sDir != UP)
          collidable |= down;
      }
      switch (collidable) {
        case 0: //no collision possible
          moveSnakeOnNoSelfCollision();
          break;
        case 1: //collision on sDir = up
          if (sDir == UP) {
            if (posY[0] != (gridY - 2))
              turnRight();
            else if (posY[0] != 1)
              turnLeft();
          }
          break;
        case 2: //collision on sDir = down
          if (sDir == DOWN) {
            if (posY[0] != (gridY - 2))
              turnRight();
            else if (posY[0] != 1)
              turnLeft();
          }
          break;
        case 3: //collision on y-axis
          if (sDir == UP || sDir == DOWN) {
            if (posY[0] != (gridY - 2))
              turnRight();
            else if (posY[0] != 1)
              turnLeft();
          }
          break;
        case 4: //collision on sDir = right
          if (sDir == RIGHT) {
```

```
            if (posX[0] != 1)
                turnDown();
            else if (posX[0] != (gridX - 2))
                turnUp();
        }
        break;
    case 5: //collision on sDir = right and/or sDir = up
        if (sDir == RIGHT) {
            if (posX[0] != 1)
                turnDown();
        }
        else if (sDir == UP) {
            if (posY[0] != 1)
                turnLeft();
        }
        break;
    case 6: //collision on sDir = right and/or sDir = down
        if (sDir == RIGHT) {
            if (posX[0] != (gridX - 2))
                turnUp();
        }
        else if (sDir == DOWN) {
            if (posY[0] != 1)
                turnLeft();
        }
        break;
    case 7: //collision not possible on sDir = left
        if (sDir == UP || sDir == DOWN) {
            if (posY[0] != 1)
                turnLeft();
        }
        else if (sDir == RIGHT) {
            printf("Trapped\n");
        }
        break;
    case 8: //collision on sDir = left
        if (sDir == LEFT) {
            if (posX[0] != 1)
                turnDown();
            else if (posX[0] != (gridX - 2))
                turnUp();
        }
        break;
    case 9: //collision on sDir = left and/or sDir = up
        if (sDir == LEFT) {
            if (posX[0] != 1)
                turnDown();
        }
        else if (sDir == UP) {
            if (posY[0] != (gridY - 2))
                turnRight();
        }
        break;
    case 10: //collision on sDir = left and/or sDir = down
        if (sDir == LEFT) {
            if (posX[0] != (gridX - 2))
                turnUp();
        }
        else if (sDir == DOWN) {
            if (posY[0] != (gridY - 2))
```

```
                turnRight();
            }
            break;
        case 11: //collision not possible on sDir = right
            if (sDir == UP || sDir == DOWN) {
                if (posY[0] != (gridY - 2))
                    turnRight();
            }
            else if (sDir == LEFT) {
                printf("Trapped\n");
            }
            break;
        case 12: //collision on x-axis
            if (sDir == RIGHT || sDir == LEFT) {
                if (posX[0] != 1)
                    turnDown();
                else if (posX[0] != (gridX - 2))
                    turnUp();
            }
            break;
        case 13: //collision not possible on sDir = down
            if (sDir == RIGHT || sDir == LEFT) {
                if (posX[0] != 1)
                    turnDown();
            }
            else if (sDir == UP) {
                printf("Trapped\n");
            }
            break;
        case 14: //collision not possible on sDir = up
            if (sDir == RIGHT || sDir == LEFT) {
                if (posX[0] != (gridX - 2))
                    turnUp();
            }
            else if (sDir == DOWN) {
                printf("Trapped\n");
            }
            break;
        case 15: //collision avoidance not possible - ignore
            printf("Cornered\n");
            break;
    }
}
//game over engine
void makeGameOver() {
    gameOver = true;
    gamePause = true;
}
//stroke text generator
void drawStrings(char *str, float sx, float sy, float tx, float ty,float thickness) {
    glColor3fv(WHITE);
    glPushMatrix();
    glScalef(sx, sy,0);
    glTranslatef(tx, ty, 0);
    for (int i = 0; str[i] != '\0'; i++) {
        glLineWidth(thickness);
        glutStrokeCharacter(GLUT_STROKE_ROMAN, str[i]);
    }
    glPopMatrix();
}
```

```
//print score string
void getScoreString() {
   sprintf(text, "%s%d", (char*)"SCORE : ", score);
   drawStrings(text, 0.02, 0.02, 60, 70,2.0);
}
//render game screen
void renderGame() {
   drawGrid();
   drawWall();
   drawFood();
   drawSnake();
}
//render welcome screen
void renderWelcome() {
   drawStrings((char*)"CLASSICAL SNAKES", 0.03, 0.03, 30, 1200, 1.5);
   drawStrings((char*)"GRAPHICS PROJECT", 0.02, 0.02, 40, 1600, 1.5);
   drawStrings((char*)"PRESS S TO START GAME", 0.01, 0.01, 100, 2700, 1.2);
   drawStrings((char*)"PRESS P TO PAUSE GAME", 0.01, 0.01, 100, 2500, 1.2);
   drawStrings((char*)"PRESS Q TO QUIT GAME", 0.01, 0.01, 100, 2300, 1.2);
   drawStrings((char*)"PRESS G TO TOGGLE GRID LINES", 0.01, 0.01, 100, 2100, 1.2);
   drawStrings((char*)"USE ARROW KEYS OR W,S,A,D AS GAME CONTROLS", 0.01, 0.01, 100,
1900, 1.2);
   drawStrings((char*)"AUTHOR : MAYANK METHA D", 0.015, 0.015, 60, 800, 1.5);
   drawStrings((char*)"USN : 1MV14CS054", 0.015, 0.015, 60, 600, 1.5);
}
//render pause screen
void renderPause() {
   drawWall();
   drawFood();
   drawSnake();
   drawStrings((char*)"GAME PAUSED", 0.03, 0.03, 190, 1100, 1.5);
   drawStrings((char*)"PRESS P TO UNPAUSE", 0.015, 0.015, 300, 1600, 1.5);
   getScoreString();
}
//render exit screen
void renderExit() {
   drawWall();
   drawFood();
   drawSnake();
   drawStrings((char*)"GAME OVER", 0.03, 0.03, 220, 1100, 1.5);
   drawStrings((char*)"PRESS Q TO QUIT", 0.015, 0.015, 300, 1600, 1.5);
   drawStrings((char*)"PRESS N TO RESTART", 0.015, 0.015, 300, 1400, 1.5);
   getScoreString();
}
//turn up
void turnUp() {
   if (sDir != DOWN && sDirFlag == true) {
      sDir = UP;
      sDirFlag = false;
   }
}
//turn down
void turnDown() {
   if (sDir != UP && sDirFlag == true) {
      sDir = DOWN;
      sDirFlag = false;
   }
}
//turn right
void turnRight() {
```

```
    if (sDir != LEFT && sDirFlag == true) {
        sDir = RIGHT;
        sDirFlag = false;
    }
}
//turn left
void turnLeft() {
    if (sDir != RIGHT && sDirFlag == true) {
        sDir = LEFT;
        sDirFlag = false;
    }
}
```

5.4 THE MAIN FILE (MAIN.CPP)

```
//headers
#include"game.h"
// display axis
// y
// ^
// |
// |
//  ------->x
//variables
short sDir; // snake direction
bool gameOver = false; //game over flag
bool gamePause = false; //game pause flag
bool gameWelcome = false; //game before start flag
int score; // score of game
int fps; // frame rate of game
bool sDirFlag = true; // snake direction change input flag
bool testFlag = false; //for testing the game
bool gridFlag = true; //show hide grids
// initalze the program
void myInit() {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    initGrid(ROWS,COLUMNS);
    initGame();
    gameWelcome = true;
}
//timer function for animation fps
void myTimer(int) {
    glutPostRedisplay();
    //increase fps by one unit after score increaases by 5
    if ((score % 2) == 0 && score != 0) {
        fps = MIN_FPS + floor(score / 2);
        if (fps > FPS) fps = FPS;
    }
    glutTimerFunc(1000 / fps, myTimer, 0);
}
//display function
void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    //Welcome screen
    if (gameWelcome && !gamePause && !gameOver) {
        renderWelcome();
    }
    //Game screen
    else if (!gameWelcome && !gamePause && !gameOver) {
        renderGame();
```

```
    }
    //Pause Screen
    else if (!gameWelcome && gamePause && !gameOver) {
       renderPause();
    }
    //Exit Screen
    else if (gameOver && !gameWelcome) {
       renderExit();
    }
    glFlush();
    glutSwapBuffers();
}
//scaling & transform screen function
void myReshape(int w, int h) {
    if(w == h)
       glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    else if(w < h)
       glViewport(0.0, (h - w)/2, (GLsizei)w, (GLsizei)w);
    else
       glViewport((w - h)/2, 0.0, (GLsizei)h, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, COLUMNS, 0.0, ROWS, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
//keyboard special key
void mySpecialKeyboard(int key, int, int) {
    switch (key) {
       case GLUT_KEY_UP:
            if (!gameOver && !gamePause && !gameWelcome)
               turnUp();
        break;
       case GLUT_KEY_DOWN:
            if (!gameOver && !gamePause && !gameWelcome)
               turnDown();
          break;
       case GLUT_KEY_RIGHT:
            if (!gameOver && !gamePause && !gameWelcome)
               turnRight();
          break;
       case GLUT_KEY_LEFT:
            if (!gameOver && !gamePause && !gameWelcome)
               turnLeft();
          break;
    }
}
//keyboard alphanumeric
void myKeyboard(unsigned char key, int, int) {
    switch (key) {
       //restart after game over
       case 'n':
       case 'N':
          if (gameOver) {
             initGame();
             gameOver = !gameOver;
             gamePause = !gamePause;
             gameWelcome = true;
          }
          break;
```

```
        //pause
        case 'p':
        case 'P':
            if(!gameOver && !gameWelcome)
                gamePause = !gamePause;
            break;
        //exit
        case 'q':
        case 'Q':
            if (!gameOver && !gameWelcome)
                makeGameOver();
            else
                exit(0);
            break;
        //turn up
        case 'w':
        case 'W':
            if (!gameOver && !gamePause && !gameWelcome)
                turnUp();
            break;
        //start and turn down
        case 's':
        case 'S':
            if(gameWelcome)
                gameWelcome = false;
            else if (!gameOver && !gamePause && !gameWelcome)
                turnDown();
            break;
        //turn left
        case 'a':
        case 'A':
            if (!gameOver && !gamePause && !gameWelcome)
                    turnLeft();
        break;
        //turn right
        case 'd':
        case 'D':
            if (!gameOver && !gamePause && !gameWelcome)
                    turnRight();
        break;
        //test mode activation
        case 't':
        case 'T':
            testFlag = !testFlag;
            break;
        //show or hide grids
        case 'g':
        case 'G':
            gridFlag = !gridFlag;
            break;
    }
}
//main()
int main(int argc, char **argv) {
    srand((unsigned int)time(NULL));
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Classical Snakes");
    glutPositionWindow(500, 100);
```

```
    glutDisplayFunc(myDisplay);
    glutReshapeFunc(myReshape);
    glutTimerFunc(0,myTimer,0);
    glutSpecialFunc(mySpecialKeyboard);
    glutKeyboardFunc(myKeyboard);
    myInit();
    glutMainLoop();
    return 0;
}
```

Chapter 6

IMPORTANT FUNCTIONS

6.1 OPENGL FUNCTIONS

glutInit(): Initializes the OpenGL API

glutInitDisplayMode(): Set the display properties like buffer, colour, depth, etc.

glutInitWindowSize(): Set the screen resolution

glutDisplayFunc(): Calls the user defined function to display content on screen

glutInitReshapeFunc(): Resize the application window to scale

glutTimerFunc(): Controls the FPS of the application

glutKeyboardFunc(): Use alphanumeric keys of keyboard

glutSpecialFunc(): Use the arrow keys and other special keys

glutMainLoop(): Refreshes the display content and redisplays it

glClearColor(): Set background color

glutPostRedisplay(): Sets flags for next glutMainLoop() iteration

glClear(): Clear the flag bits

glFlush(): Display content as soon as it is rendered

glutSwapBuffers(): Swap buffer with another to animate objects

glViewport(): Set the window scale and position property when window resizes

glMatrixMode(): Use world coordinate or camera coordinate

glLoadIdentity(): See the Matrix in MatrixMode to identity Matrix

glOrtho(): Anchor the content as specific position on screen

glColor3fv(): Set color red green and blue values

glLineWidth(): Set thickness of lines

glVertex2d(): Set vertex position

glBegin(): Start drawing of primitives

glEnd(): End drawing of primitives

glRectd(): Alternative GL_POLYGON with 4 vertices

glPushMatrix(): Push the rendered scene onto a stack

glPopMatrix(): Pop the rendered scene from the stack

glScalef(): Scale transformation

glTranslatef(): Translate transformation

glutStrokeCharacter(): Draw alphanumeric characters using line strokes

6.2 USER DEFINED FUNCTIONS

initGame(): Initializes the game

initGrid(): Initialize the grid

drawCell(): Draw each cell of the grid

drawGrid(): Draw the grid lines

drawWall(): Draw the wall

drawFood(): Draw the food randomly

drawSnake(): Draw the snake

makeGameOver(): Set gameOver flags

drawStrings(): Draw the strings using Stroke characters

getScoreString(): Convert score value to string

renderGame(): Renders the game objects like grid, wall, food and snake

renderWelcome(): Renders the game welcome screen

renderPause(): Renders pause screen

renderExit(): Render exit screen

turnUp(): Moves the snake upward

turnDown(): Moves the snake downward

turnRight(): Moves the snake rightward

turnLeft(): Moves the snake leftward

myInit(): Initalizes the application on first run

myTimer(): glutTimerFunc() function parameter

myDisplay(): glutDisplayFunc() function parameter

myReshape(): glutReshapeFunc() function parameter

mySpecialKeyboard(): glutSpecialFunc() function parameter

myKeyboard(): glutKeyboardFunc() function parameter

Chapter 7

SCREENSHOTS

7.1 WELCOME SCREEN

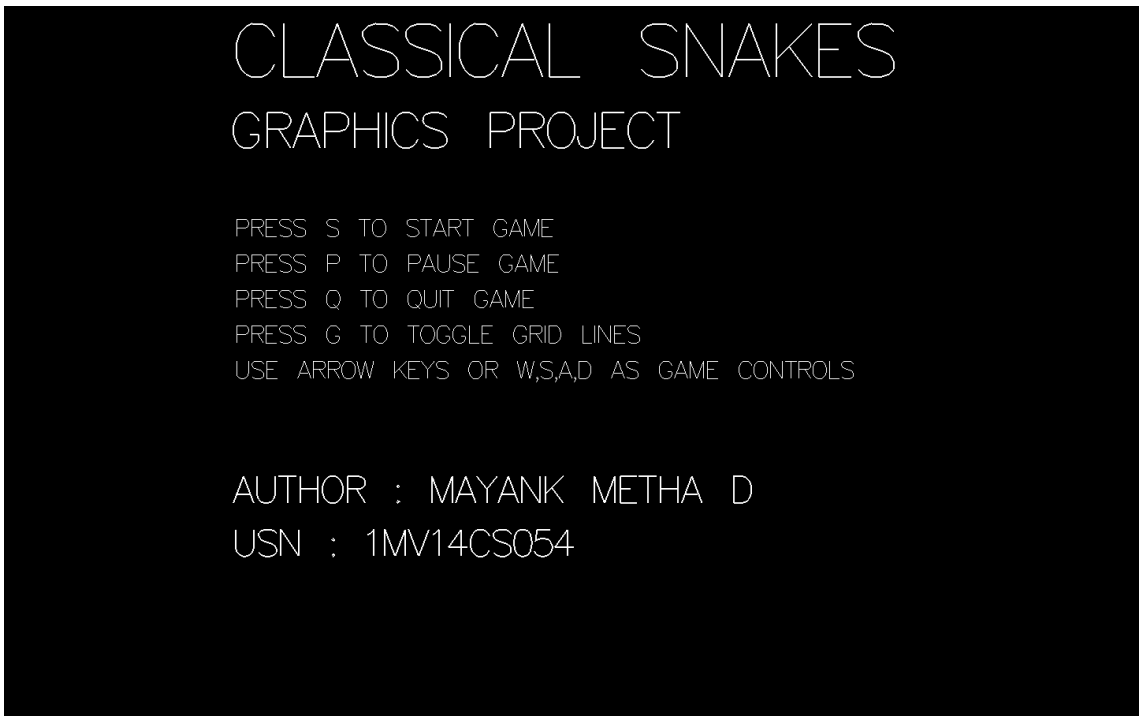

Figure 1: MAIN MENU

7.2 THE GAME

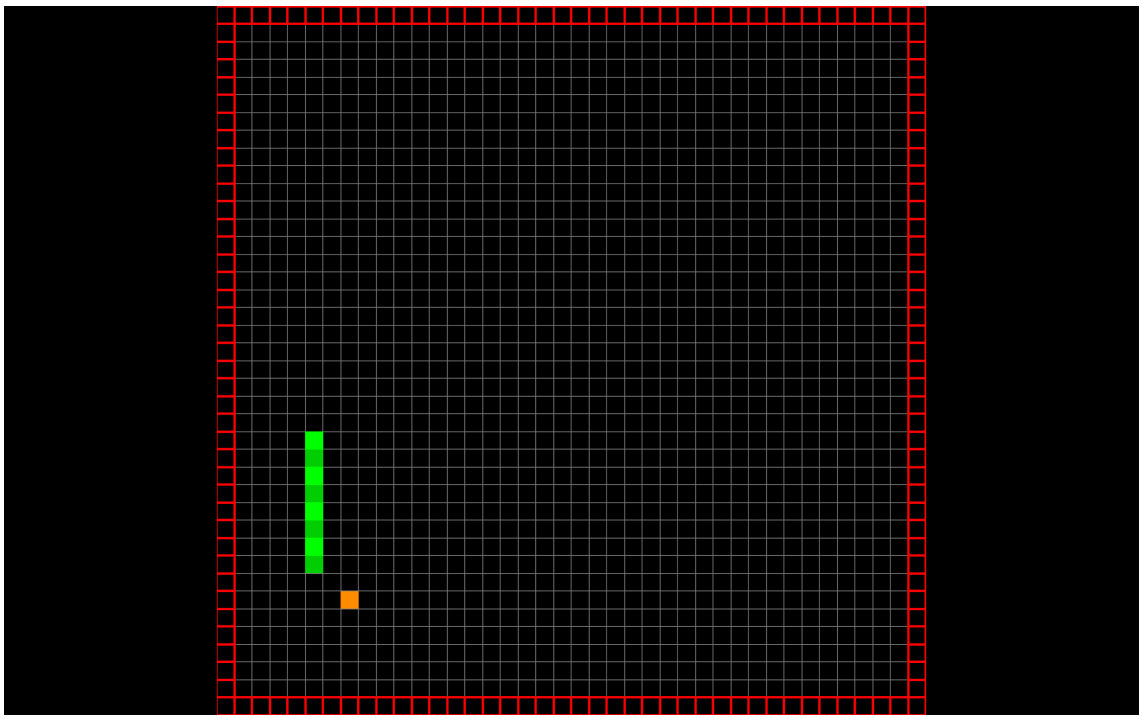

Figure 2: GAME PLAY WITH GRIDLINES

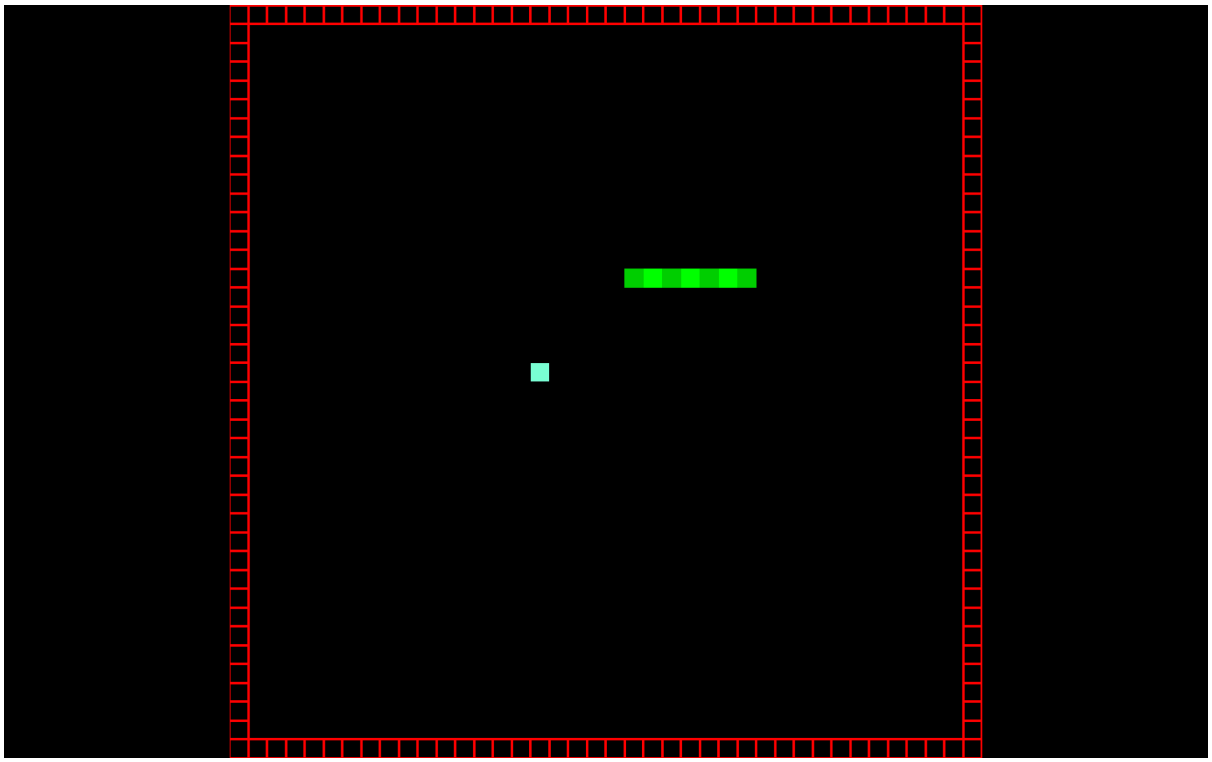

Figure 3: GAME PLAY WITHOUT GRIDLINES

7.3 PAUSE SCREEN

Figure 4: PAUSED GAME

7.4 GAME OVER AND EXIT SCREEN

Figure 5: SNAKE COLLIDES WITH ITSELF

Figure 6: SNAKE COLLIDES WITH WALL

Figure 7: QUITING USING KEYBOARD

Chapter 8

CONCLUSION

The Classical Snakes Game has been tested on Windows 10, Ubuntu 17.04 and MacOS Sierra (10.12.5). It was easy for porting it to popular computer based OS with different hardware like CPU, GPU, RAM, Keyboard, Display. The simple UI for the game gives it the Retro 8-bit game feel.

This project helps me learn how to use simple codes to build simple graphic content. These simple graphic contents can help render a complex set of graphical objects easily. It also helps me understand the mechanics of a game, handling various platforms and systems to help give the same user experience on all the platforms.

Chapter 9

FUTURE ENHANCEMENT

The following can be thought of future enhancement of the project

- Add random obstacles in the game
- Add multiplayer options over a local network
- Have a Score board Ranking and High Score System
- Adding a 3D effects
- Adding sound effects
- Support for mobile platforms like Android and iOS
- Add a AI using Deep Learning for testing the limits of the game

BIBLIOGRAPHY

1. BOOKS

- OpenGL Game Development by Robert Madesen
- OpenGL SuperBible by Graham Sellers and Richard S
- OpenGL Programming Guide by Dave Shreiner
- Interactive Computer Graphics by Edward Angel

2. WEBSITES

- https://learnopengl.com
- http://www.opengl-tutorial.org
- http://ogldev.atspace.co.uk
- http://www.tomdalling.com/blog/category/modern-opengl/