Jaypee Institute of Information Technology
Department of Computer Science & I.T.

PROJECT REPORT

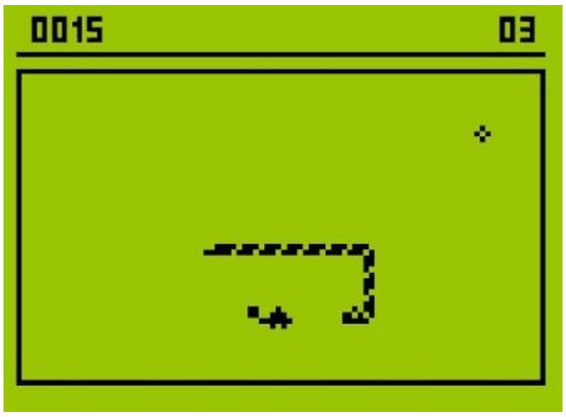
Report to highlight the working of SLYTHERON – THE SNAKE GAME

Submitted by:

Siddhant N Trivedi	**17104025**
Mayank Singh	**17104029**
Aman Verma	**17104006**
Siddharth Aggarwal	**17104031**

Introduction to the Project

Slytheron - the Snake Game



How can we forget the first game we all played when the all new Nokia Mobile came into existence!!!Yes, the snake game in which we found thrill by making the snake long enough and beating the highest score of our elder cousins….

And since, this being our first project, so we thought of reviving the same experience of an interesting snake-eater game through the implementation of a graphics library i.e. **OpenGL and Linked lists**.

About OpenGL: OpenGL is a cross platform and cross language API that helps in rendering real time 2D and 3D graphics in our applications. It houses a large variety of functions and extensions that ease the work of graphics development and rendering. Even many popular game development user-interface libraries like **SDL, Allegro, SFML, Qt and FLTK**. The first library that was made solely to display and OpenGL window was **OpenGL Utility Toolkit(GLUT)**. In our project, we have used **FreeGLUT** library of OpenGL which offers more functionality than GLUT like keyboard and mouse inputs etc.

Game Features:

Our Game “SLYTHERON” consists of two windows-

1. Our initial window which displays the title of our game in the window screen, and on pressing SPACE key it starts the game play and on pressing ESCAPE key it ends the game.
2. Our Second Game window which consists of title in the title bar with Live Score Updates aligned to it, then it contains different objects such as Snake itself, Rectangular Needled Hurdles, and Food which is drawn on random position of x co-ordinates and y co-ordinates in accordance with the random1() function which we have implemented in our game. Snake Game consists of few different collision conditions, which are:
    a) When the snake collides with itself, the Game Over condition becomes true, which then displays the updated score in a dialog box created with the help of MessageBox() function.
    b) There are two different rectangular hurdles at the top and bottom of the game window, each of the rectangular hurdles consist of 6-pointed needled structure, on touching them the Snake dies, and once again the score is displayed.
    c) Apart from displaying the score, we have also implemented a Leader board for our game, which displays the names of different players with their respective scores in a Sorted fashion.
    d) Our Game also has a Pause Mode, when we press ALT key on our keyboard, our game pauses.

Thus, we have tried our best to provide an amazing user experience of gameplay while collecting food for the snake and simultaneously getting to the top of the leaderboard. So, it will surely be a great fun filled time for any player to try it access his reflexes.

DATA STRUCTURES USED

A game without a leader board becomes morose and boring and so, in this project, we have used Linked List as our data structure for storing as well as sorting and arranging the scores of various players in descending order which as a result, helps us to display the Leaderboard of the game.

To keep storing the name and score of participant of all gameplays, we have supplemented our data structures with File Handling.

The data flow is as follows:

1. The score of present gameplay keeps getting stored in a global variable score.
2. As the game gets over, the linked list is filled up with previous stored data, if any from the file.
3. Then, the score and player id with the prefix MARK is appended to the linked list.
4. Now, the linked list is sorted in descending order and after this, the list elements are sorted in descending order and then type casted to String datatype so that they can be printed on the MessageBox which is a windows system function included in the GLUT.h header file
5. The sorted linked list elements are written again to the file.
6. Now, the whole sorted data (according to score) is concatenated to the single string file and then printed on the Message box screen.

Thus, Linked lists due to its versatility and easy mutabilty serves as one of the best choices of data structures for us and serves our cause to the best extent. Thus, we have tried to put it to the best of use.

DIVISION OF WORK AMONG THE GROUP MEMBERS.

GROUP MEMBERS	CONTRIBUTIONS
SIDDHANT N TRIVEDI	I. GAME WINDOW:- Creating the game window with OpenGL. II. SNAKE:- Designing snake and the arena using the same. III. LEADERBOARD:- Displaying the leaderboard after the is game over using Linked lists.
MAYANK SINGH	I. GAME WINDOW:- Creating the game window with OpenGL. II. SNAKE:- Designing snake and the arena using the same. III. HURDLES:- Hurdles and Death Pins to make game more interesting . IV. LIVE SCORE UPDATER :- Displaying the live score on the top of the game window.
AMAN VERMA	I. OPENING STARTING WINDOW:- Creating the opening startup window with OpenGL libraries. II. FUNCTIONALITY OF OPENING WINDOW:- Connecting the opening window with the game window. III. GAME WINDOW:- Creating the game window with OpenGL.
SIDDHARTH AGGARWAL	I. SNAKE:-Building the snake with OpenGL. II. COLOR SCHEME:-Selecting the suitable color scheme and features for different objects and windows. III. ADDED DOCUMENTATION TO CODE:- Added comments to the codebase of the project.

Important Functions used in the Project

Our Header file **“game.h”** is used across multiple files in our project, it contains the prototypes of the different functions which are used in **“game.c”, “lboard.c”** & **“main.c”** which handles the drawing of different objects, showing the leader board of all the players, and implementing the main windows through the GL libraries respectively along with many other different features, the definition of function prototypes which are used in these files are explained below:

1. *main.c :*

a. glutdisplay() and display_callback():

“glutDisplayFunc” sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the “display_callback” for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and (if no overlay display callback is registered) the layer in use is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback

When a window is created, no display callback exists for the window. It is the responsibility of the programmer to install a display callback for the window before the window is shown. A display callback must be registered for any window that is shown. If a window becomes displayed without a display callback being registered, a fatal error occurs.

b. glutReshape and reshape_callback():

“glutReshapeFunc” sets the reshape callback for the current window. The “reshape_callback” is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width and height parameters of the callback specify the new window size in pixels. Before the callback, the current window is set to the window that has been reshaped.

If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply call glViewport(0,0,width,height) on the normal plane.

If an overlay is established for the window, a single reshape callback is generated. It is the callback's responsibility to update both the normal plane and overlay for the window**.**

c. gluttimer() and timer_callback():

“glutTimerFunc” registers the “timer_callback” function to be triggered in at least milliseconds or frames per seconds. The value parameter to the timer callback will be the value of the value parameter to “glutTimerFunc”. Multiple timer callbacks at same or differing times may be registered simultaneously.

The number of milliseconds is a lower bound on the time before the callback is generated. GLUT attempts to deliver the timer callback as soon as possible after the expiration of the callback's time interval

d. glutkeyboard() and keyboard_callback():

“glutKeyboardFunc” 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. The “keyboard_callback” parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. Passing NULL to “glutKeyboardFunc” disables the generation of keyboard callbacks.

e. void printtext():

It basically sets the position of the pointer and using glRasterPos2i(x,y) and print the text there using the loop.

f. void processnormalkeys():

It processes normal keys by taking 3 parameters.Ist is the ascii code of the key pressed and the other 2 for the position.

g. void display(void):

It clears the color buffer of the screen ,changes the color of the screen and calls the printtext() function for printing the the text.

h. initmain():

“initMain” set up the grid of desired rows and columns and sets the background color of the window. It also helps to set the opacity of the background i.e. the Alpha value.

i. initgamewindow()

“initGameWindow” set up the game grid of the desired rows and columns and sets the background colour of the game window. It also helps to set the opacity of the background i.e. the Alpha value.

2. ***game.c :***

The different global variable used in the implemented functions in **“game.c”** are as follows:

a. **extern int score;** -

the **int score** variable was already defined in **“main.c”** and we added an extra keyword **extern** before int score in **“game.c”**, so that we can use the same score variable across different files.

b. **extern bool gameOver;** -

similiary, we defined a boolean type of variable gameOver with extern keyword in “game.c” so as to use it across different files, the gameOver variable is helping us in defining collision and end of the game.

c. **int gridX, gridY, foodX, foodY, snake_length = 5;** -

gridX , gridY variable are used to draw the square grid pattern which we have used in our game file, we initialize these variables in initGrid(int x,int y) function, with rows and columns , both of these in our case are equal to 40.

foodX, foodY are the variables which are used to draw food at random positions on the screen, to achieve this we have used random1(int *x,int *y) function,

snake_length is the variable which we have used to control the length of the snake which can’t be more than 60, the initial length of the snake will be equal to 5,

d. **int posX[60]={20,20,20,20,20},posY[60]={20,19,18,17,16};** -

we have defined these two arrays, which are initiated with initial co-ordinates of our snake, these two arrays are used to define the snake’s body, if the snake catches food, the snakes length will be increased, and to simulate the movement of snake we use these arrays.

e. **bool food = true;**

   if the food variable is equal to true, the random1() function will be called to draw the food and once we are done with drawing the food, the food variable will be set to false, and finally when the food is collected again the food variable will be set to true.

f. **short sDirection= RIGHT;**

   this variable defines the initial direction of the snake, that is equal to right, RIGHT is the keyword defined in the glut library and also equal to right arrow key.

The functions which are implemented in “game.c” are:

a. **void unit(int x,int y)**

: this function takes arguments of int x and int y , which here are the co-ordinates at which the unit has to be drawn, this function draws the units of the grid mainly with the help of glut library functions, glBegin(GL_LINE_LOOP); glVertex2f(x,y); & glEnd(); and connects them through GL_LINE_LOOP.

b. **void hurdle()**

: this function implements drawing of the two rectangular, needled hurdles, where the two for loops are used to draw top and bottom rectangular region with the help of glRect(x1,x2,y1,y2) and the multiple GL_LINE_LOOP (s) are used to draw the needles at each of the top and bottom rectangular hurdle.

c. **void drawFood()**

: this function is used to draw the food at random positions on the screen by the help of food variable, foodX and foodY variable , random1(int *x, int *y) function and glRect(x1,x2,y1,y2) function from the glut library, also it is taken care that the food is not drawn on and around the edge of rectangular hurdles.

d. **void drawSnake()**

: this function is used to draw the snake and keep swapping the different unit parts of the snake with the help swapping technique as defined in the loop, we also have enabled our snake to cross the boundary edges of the screen and we have taken care that once the snake crosses any boundary it reappears in the opposite boundary heading in the same direction as it was doing previously, until any arrow keys are pressed by the gamer, in the drawSnake() function we have also implemented all the different collision conditions, those with itself, the hurdles in different directions, also we have defined the food collection condition in the drawSnake() function, taking care that when the snake collects food it’s length and score increases.

e. **void random1(int *x,int *y)**

: this function is accepting two pointer type variables, that in our case will be the foodX and foodY co-ordinates, which are passed through reference so that value assigned by the random function is reflected directly to the global variables foodX and foodY.

f. **void initGrid(int x,int y)**

: this functions initializes the two variable gridX and gridY by accepting rows and columns respectively from the main function that in our case will be equal to 40 x 40 , that is 40 rows and 40 columns.

g. **void drawGrid()**

: this function leverages unit(int x, int y) function , it is using two for loops to draw the unit squares to complete the grid at each and every x and y co-ordinates.

3. *lboard.txt*

a. **struct Node**

:declares the structure for the creation of Linked List with the elements being score and name.

b. **void fileread()**

:copies the previously stored data from the file into the newly created linked list for including the scores of previous gameplays in the Linked lists.

c. **vocinsertlist()**

:function to insert the data extracted from the file into the linked list.

d. **void sortlist()**

:function to sort the linked list in descending order.

e. **void writetofile()**

:function to write the newly created linked list into the file.

f. **void boardprint()**

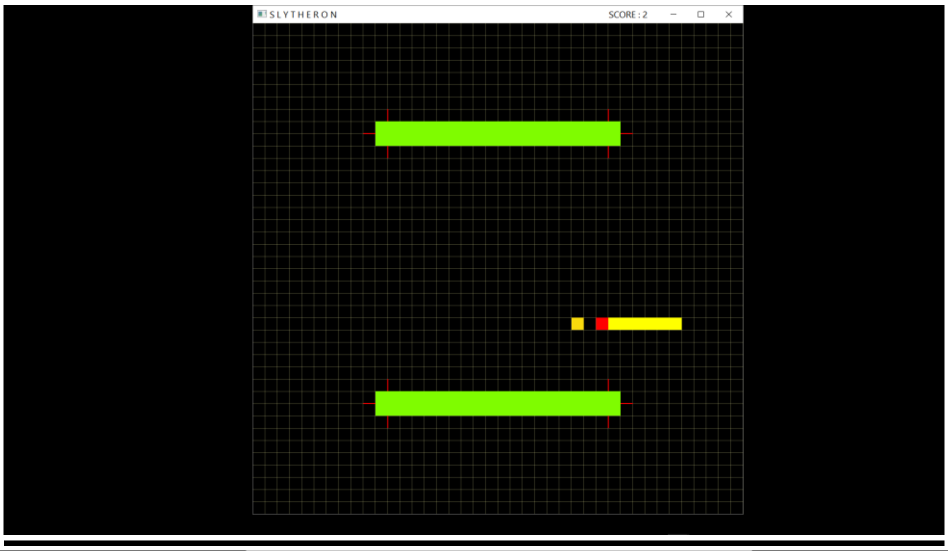
:the main function which calls all other functions i.e. fileread(), insertlist() , sortlist() and writetofile(). Having got the sorted linked list, it concanates the scores and names to a single character array and prints it using the MessageBox() function.

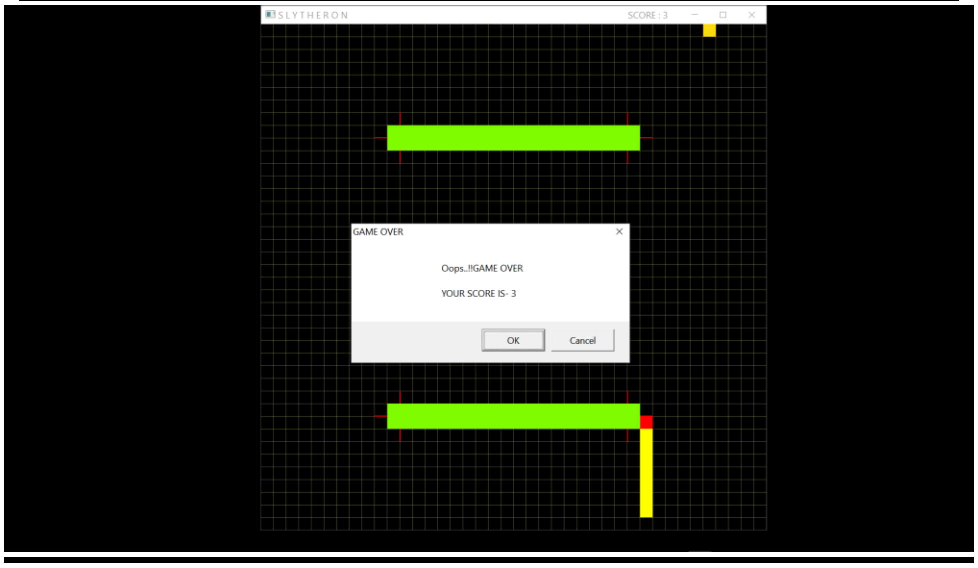
g. **MessageBox()**

:the windows function included in the <GLUT.h> header file which helps to display a message box with some functional buttons in Windows OS.It accepts only String arguments and then prints the whole string on the MessageBox window.

SCREENSHOTS OF THE OUTPUT

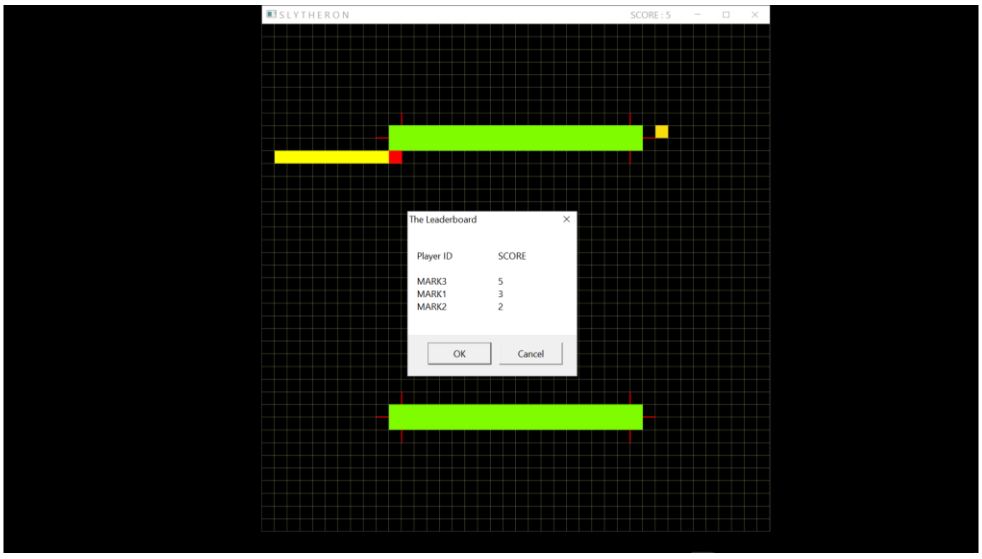
The background of the screen while running the game are as follows:





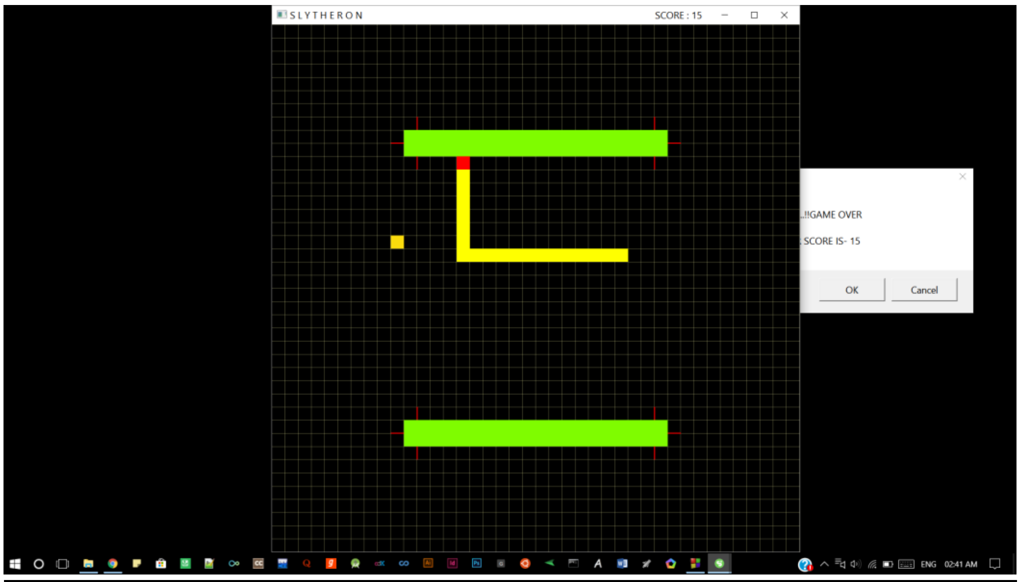
Oops..!!GAME OVER

YOUR SCORE IS- 3



The Leaderboard

Player ID	SCORE
MARK3	5
MARK1	3
MARK2	2



..!!GAME OVER

SCORE IS- 15

<u>THANK YOU….</u>