CS118: Programming Fundamental
Project: Snake Game

Jawad Hassan, Amna Irum, Atifa Sarwar

Deadline: December 13, 2018 before 23h30

Attention:

- Make sure that you read and understand each and every instruction. If you have any questions or comments you are encouraged to discuss your problems with your colleagues (and instructors).
- Plagiarism is strongly forbidden and will be very strongly punished. If we find that you have copied from someone else or someone else has copied from you (with or without your knowledge) both of you will be punished. You will be awarded (straight zero in the project which can eventually result in your failure) and appropriate action as recommended by the Disciplinary Committee (DC can even award a straight F in the subject) will be taken.
- Try to understand and do the project yourself even if you are not able to complete the project so that you will be awarded according to your effort.
- Divide and conquer: since you have around **13** days so you are recommended to divide the complete task in manageable subtasks. We recommend to complete the drawing and design (i.e. number of functions and their order of calling) phase as quickly as possible and then focus on the intelligence phase.
- Before writing even one line of code, you must design your final project. This process will require you to break down and outline your program into functionality of your program and pseudocode important methods. After designing your program you will find that writing the program is a much simpler process.
- Imagination Powers: Use your imaginative powers to make this interesting and appealing as you can think of. An excellent solution can get you bonus marks.

Game Description:

Most of you have played a very interesting game “Snake” on your old Nokia phones (Black & White). Now it is your time to create it with more interesting colors and features. When the game is started a snake is controlled by up, down, left and right keys to eat food which appears on random locations. By eating food snake’s length increases one unit and player’s score increases by 5 points. Food disappears after 15 seconds and appears again on some other random location. The game also has hurdles placed at the random locations. The snake dies if it collides with itself or the hurdle.

1. Create board (hint: a 2D array) and snake. Snake first block (mouth) must be a circle and have at least 2 squares as the tail.
2. Move snake
3. Show player score on progress bar at top of the game.
4. Display five foods at a particular time at random locations on a board by following the following rules
    a. Not two food items are on same row.

b. Not two food items are on the same columns.
c. Not two food are along the same diagonal (primary and secondary).
d. When the snake eats the food, the food item again appear by following the same conditions described above.
e. The length of snake increases when it eats food.
f. Each food disappears after 15 seconds of its placement.

5. Introduce the hurdles on board.
    a. You can have at least one hurdle and at most three.
    b. The snake should avoid the hurdles while navigating to eat food item.
    c. The hurdles disappear after 30 seconds and reappear again at different locations on the board.
    d. New hurdles should be placed such as they do not collide with already placed food items and snake.
    e. Hurdles can take “L-shape”, “U-shape”,”Z-shape”, and horizontal or vertical lines of variable lengths.
    f. You can increase or decrease the difficulty level using hurdles.
6. Game must terminate when snake collide with itself or with hurdles by showing a message game over and your score is “XYZ”. If score is highest show a message “you have broken all previous records and now highest score is xyz”. After this, start the game once again.
7. Display power food after each minute and it should disappear after 15 seconds. If player eats a power food its score is increased by 20.
8. Save highest score and players history in files and display corresponding information when user selects an option from them (figure 2).

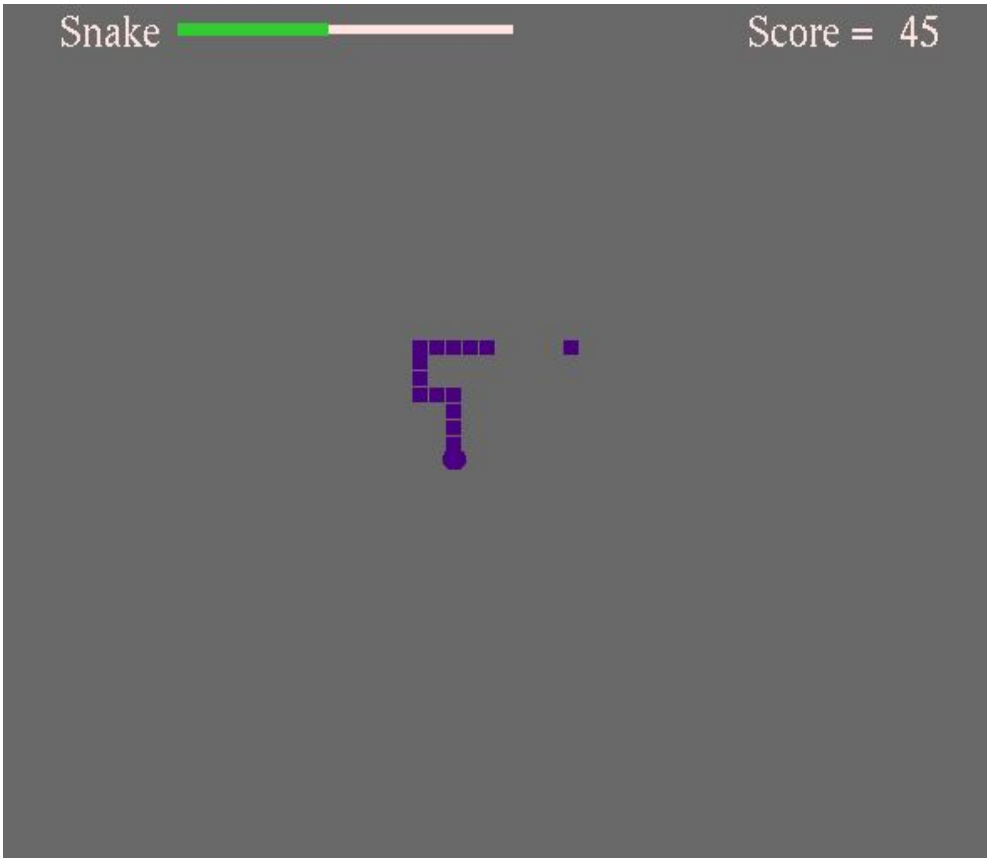

Figure 1: Snake game with snake and food.

BONUS MARKS:

When we open snake game we see a screen like below (figure 2) which shows different options: Start Game, Resume Game, Change Level, See High Score and See Previous History of Game. When user select “Start Game” option first level of game started. You can also see high score and history of game from given options.

Figure 2: Shows different options to start snake game

How to Start:

In this project you will be provided code of some functions to draw circle, rectangle, print string and code to control keys (up, down, left and right). You are not required to understand the given functions implementation, you only have to call these functions by passing appropriate arguments. For example if you want to draw a circle you will call a function “DrawCircle(int starting_x_axis, int starting_y_axis, float radius , float color_of_circle)” and it will draw a circle depending upon the arguments. Same for remaining functions you will only call them and they will do your job. Details of these functions are given in code you can see them from there.

Instructions

We provide complete skeleton with all the basic drawing functions (can be located in util.h and util.cpp) needed in project with detailed instructions and documentation. In other words, all you need to know for building the game is provided. Your main task will be to understand the main

design of game and implement it. However, before proceeding with code writing you will need to install some required libraries.

1.1 Installing libraries on Linux (Ubuntu)

You can install libraries either from the Ubuntu software center or from command line. We recommend command line and provide the file “install-libraries.sh” to automate the complete installation procedure. To install libraries:

1. Simply run the terminal and go to directory which contains the file downloaded file “install-libraries.sh”.

2. Run the command $: bash install-libraries.sh
3. Provide the password and wait for the libraries to be installed. If you get an error that libglew1.6-dev cannot be found, try installing an older version, such as libglew1.5-dev by issuing following on command line $: sudo apt-get install libglew1.5-dev
4. If you have any other flavour of Linux. You can follow similar procedure to install “OpenGL” libraries.

1.2 Compiling and Executing

To compile the game (skeleton) each time you will be using “g++”. However to automate the compilation and linking process we use a program “make”. Make takes as an input a file containing the names of files to compile and libraries to link.
This file is named as “Makefile” in the game folder and contains the detail of all the libraries that game uses and need to linked. So each time you need to compile and link your program (game) you will be simply calling the “make” utility in the game directory on the terminal to perform the compilation and linking.$: make
That’s it if there are no errors you will have your game executable (on running you will see three shapes on your screen). Otherwise try to remove the pointed syntax errors and repeat the make procedure.
You only have to modify the code in "game-release.cpp" file and particularly in "void Display()" function which is called by graphics library automatically. After each modification you need to run $: make command to compile your code and link binary libraries. This will create an exe. file named game-release which will be executable. Run that game-release file to verify the modification in your game by $: ./game-release command. If you run the skeleton code that we have provided you, a window like below will appear on your screen (figure 3).

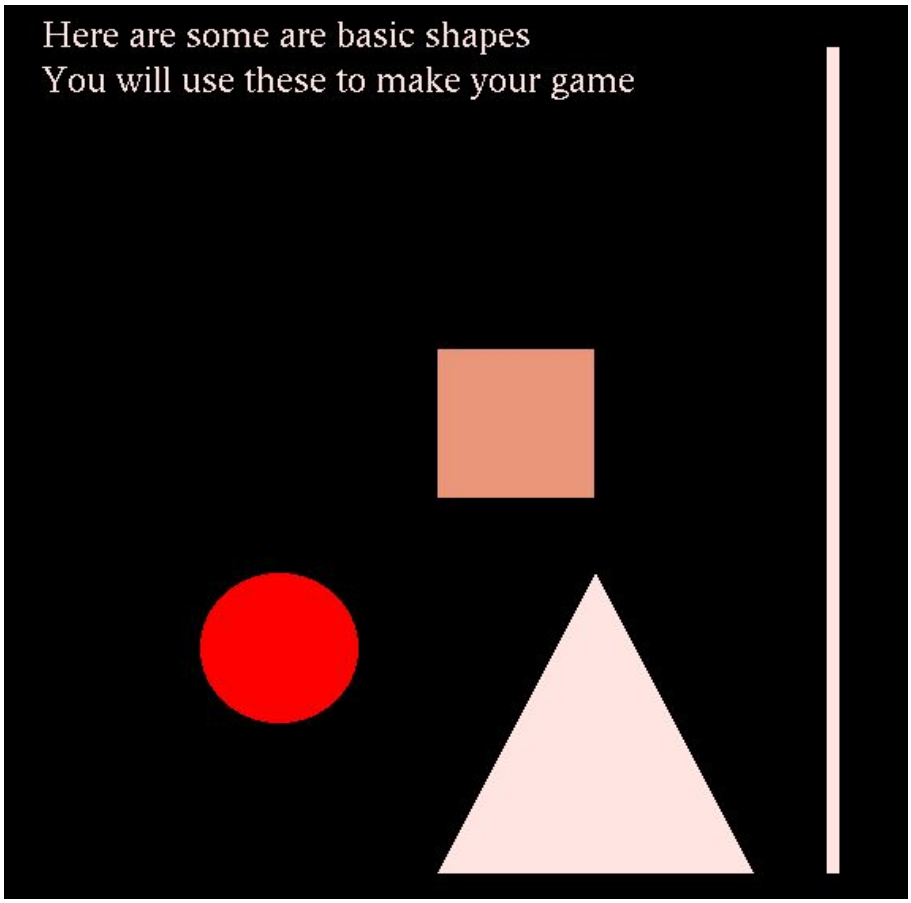

Figure 3: Shows skeleton code output

Drawing Board and Shapes

Canvas:

Your first goal will be to write the code for drawing the canvas and basic shapes. Since we are building 2D game, our first step towards building any game will be to define a canvas (our 2D world or 2D coordinate space in number of horizontal and vertical pixels) for drawing the game objects (in our case hurdles and snake). For defining the canvas size you will be using (calling) the function “SetCanvas” (see below) and providing two parameters to set the drawing world width and height in pixels.

/ Function sets canvas size (drawing area) in pixels. that is what dimensions (x and y) your game will have. Note that the bottom-left coordinate has value (0,0) and top-right coordinate has value (width-1,height-1). To draw any object you will need to specify its location */*
void SetCanvasSize(int width, int height)

// Drawing functions provided in the skeleton code
/* To draw a triangle we need three vertices with each vertex having 2-coordinates [x, y] and a color for the triangle.This function takes 4 arguments first three arguments (3 vertices + 1 color) to draw the triangle with the given color.
*three component color vector */
void DrawTriangle(int x1, int y1, int x2, int y2, int x3, int y3,float color[ ])

```
// Function draws a circle of given radius and color at the given point location (sx,sy).
void DrawCircle(float x, float y, float radius, float *color);

// Function draws a line between point P1(x1,y1) and P2(x2,y2)  of given width and colour
void DrawLine(int x1, int y1, int x2, int y2, int lwidth = 3, float *color =NULL);

// Function draws a square of given size at given sx,sy coordinates
void DrawSquare(int sx, int sy, int size,float color[]);
// Function draws a string at given x,y coordinates
void DrawString(int x, int y, const string& str, float * color = NULL);
```

Drawing Primitives

Once we have defined the canvas our next goal will be to draw the game board and its elements using basic drawing primitives. For drawing each object we will need to specify its elementary point's locations (x & y coordinates) in 2D canvas space and its size. You will only need squares as drawing primitives to draw the complete board, and its pieces. For this purpose, skeleton code already include functions for drawing lines, squares circles, curves, rounded rectangles (see below), triangles at specified location.
Recall that a line needs two vertices (points) where as a triangle needs three vertices so to draw these primitives we will need to provide these vertices (points) locations along with shape's color – c.f . Skeleton already provides a list of ≈ 140 colors which can be used for coloring different shapes note that each color is combinations of three individual components red, green and blue.

Drawing Board

Initially it might seem drawing and managing the board is extremely difficult. However this difficulty can be overcome using a very simple trick of divide and conquer. The trick revolve around the idea of the board being split into tiles. "Tile" or "cell" in this context refers to an 10 × 10 pixels square box, you can use any tile size as you wish – pixel square on the screen. Snake's screen resolution is 650 × 650, so this gives us a total board size of 65x65 tiles. Since we will be working independent of pixel units, so we can define tile size in our coordinates units. So drawing and managing the board will require these two steps:
1. Splitting the board in tiles.
2. Finding and storing what part of piece to draw in each tile.

Snake board divided into a grid of tiles or cells. Note that once we have divided the board into a grid we can build a 2D table in which we can record the contents corresponding to each tile, i.e. what shape it contains and at what location. Given the table of contents we can loop over this table and draw the complete board.
Remember that you can do your drawing only in the Display() function, that is only those objects will be drawn on the canvas that are mentioned inside the Display function. This Display function is automatically called by the graphics library whenever the contents of the canvas (window) will need to be drawn i.e. when the window is initially opened, and likely when the window is raised above other windows and previously obscured areas are exposed, or when glutPostRedisplay() is explicitly called. In short, Display function is called automatically by the

library and all the things inside it are drawn. However whenever you need to redraw the canvas you can explicitly call the Display() function by calling the function glutPostRedisplay(). For instance, you will calling the Display function whenever you wanted to animate (move) snake; where first you will set the new positions of your objects and then call the glutPostRedisplay() to redraw the objects at their new positions. Also see the documentation of Timer function.

Interaction with the Game

For the interaction with your game you will be using arrow keys on your keyboard. Each key on your keyboard has an associated ASCII code. You will be making use of these ASCII codes to check which key is pressed and will take appropriate action corresponding to the pushed key. E.g. to move the snake right you will check for the pressed key if the pressed key is left arrow you will move the snake left (change its position by adding some value in x-axis). Keyboard keys are divided in two main groups: printable characters (such as a, b, tab, etc.) and non-printable ones (such as arrow keys, ctrl, etc.). Graphics library will call your corresponding registered functions whenever any printable and non-printable key from your keyboard is pressed. In the skeleton code we have registered two different functions (see below) to graphics library. These two functions are called whenever either a printable or non-printable ASCII key is pressed (see the skeleton for complete documentation). Your main tasks here will be to add all the necessary functionality needed to make the game work.

```
/*This function is called (automatically by library) whenever any non-printable key (such as up-arrow, down-arraw) is pressed from the keyboard. You will have to add the necessary code here
when the arrow keys are pressed or any other key is pressed...
* This function has three argument variable key contains the ASCII of the key pressed, while x and y tells the program coordinates of mouse pointer when key was pressed.*/
void NonPrintableKeys(int key, int x, int y)

/* This function is called (automatically by library) whenever any printable key (such as x,b, enter, etc.) is pressed from the keyboard. This function has three argument variable key contains  the ASCII of the key pressed, while x and y tells the program coordinates of mouse pointer when key was pressed.*/
void PrintableKeys(unsigned char key, int x, int y)
```

Collision Detection

Finally, once you have done the drawing and animation of objects (snake , board, food) on your canvas. Your final goal will be to detect collisions (collision test) between objects (snake and food , snake and hurdles) and take necessary actions, e.g. to check whether a snake and a food collide if collide disappear food from that location and display it on some other random location increase length of snake, score and progress bar. If collide with hurdle or itself terminate the game.

Acknowledgments:

We acknowledge that attention-instructions, libraries to draw basic primitives and code-skeleton are taken from Computer Programing course that was taught in spring 2015 at FAST University Islamabad by faculty members: **Sibt ul Hussain, Fareed Ahmed, Usman Farrokh.**