PF Project: Word Shooter

Deadline: Tuesday, December 3, 2024 11:59 PM

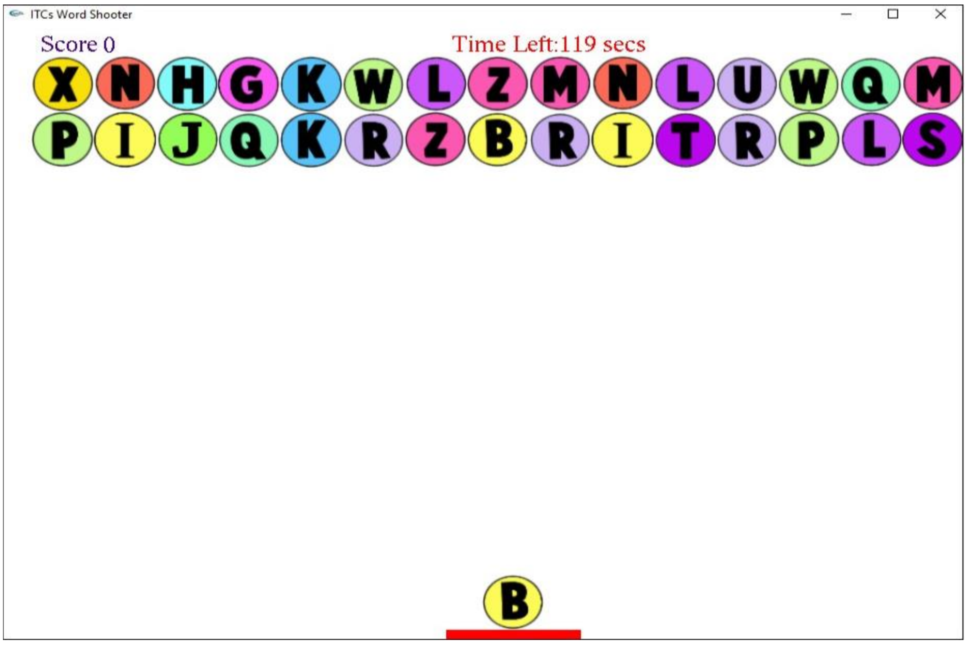
Instructions:

- 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 TA's and instructors on GCR comments.
- Plagiarism is strongly forbidden and will be very strongly punished. If we find that you have used any sort of AI generation, 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.
- Divide and conquer: since you have around 10 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 calling order) phase as quickly as possible and then focus on the other parts. [Roughly you will require at least 50 hours to complete the task]
- 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, design your data structure (arrays, stings, etc.), clarify the major 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 as interesting and appealing as you can think of. Try to understand and do the project yourself.

Good Luck ☺

Goals: In this project you will build a 2D game (ITC's Word Shooter – see Figure 1

The major goal of this project is to consolidate the things you have learned during the course. In this respect it is requested to completely follow the principles you have learned during the course. Moreover, it is an excellent time to put imaginative and creative powers into action.



1. 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".

```
bash install-libraries.sh
```

2. Run the command
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 link.

So each time you need to compile and link your program (game) you will be simply calling the "make"

```
make
```

utility in the game directory on the terminal to perform the compilation and linking. 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.

2. Drawing Board and Shapes

Your first goal will be to write the code for drawing the canvas and basic shapes.

2.1 Canvas

Since we will be building 2D games, 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 alphabets). 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)
```

2.2 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 need alphabets as drawing primitives to draw the complete board, and its pieces.

For this purpose, skeleton code already include functions for drawing alphabets (see below) at specified location.

```
 // Drawing functions provided in the skeleton code

/*Draws a specific alphabet at given position coordinate
                   sx = position of x-
                   axis from left-bottom
             *     sy = position of y-
             *     axis from left-bottom
             *     awidth= width of displayed
             *     alphabet in pixels aheight= height
                   of displayed alphabet pixels.
                 * */
void DrawAlphabet(const Alphabets &cname, int sx, int sy, int
cwidth=60,
                int cheight=60)

// Function draws a string at given x,y coordinates
void DrawString(int x, int y, int width, int height, const
string&score,
float*color)

void DrawShooter(int sx, int sy, int cwidth = 60, int cheight = 60)
```

Skeleton also provides a list of 140 colors (see file util.h) which can be used for drawing strings of different colors. Note that each color is combinations of three individual components red, green and blue and each color is stored as a separate row in the two dimensional array.

2.3 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 30 × 30 – you can use any tile size as you wish – pixel square region on the screen. Game’s screen resolution is 660 × 930 (there are 910 pixel horizontally in each row and there are 660 such rows), so this gives us a total board size of 22 × 31 tiles. 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.
Furthermore you might need to convert pixel-coordinates to cell (or tile)-coordinates and vice versa. It will be extremely helpful for you if you can define two functions for performing these tasks.

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, when it is maximized or minimized, and likely when the window is raised above other windows and previously obscured areas are exposed, or when *glutPostRedisplay()* function is explicitly called.
In short, *Display* function is called automatically by the library and all the things inside it are drawn. How- ever 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) your objects; 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.

2.4 Interaction with the Game

For the interaction with your game you will be using your mouse. You will need to know when and where the left- mouse button is clicked and released inside in the window of your game. Graphics library will call your corresponding registered functions whenever mouse button is clicked. In the skeleton code we have registered a function (see below) to graphics library. This function is called whenever either mouse button is pressed or released(see the skeleton for complete documentation). Your main tasks here will be to add all the necessary functionality needed in these two functions to make the game work.

```
/*This function is called (automatically) whenever your mouse
button is clicked witin
↪inside the game window
 *  You will have to add the necessary code here for shooting,
    etc.
 *
    This function has four arguments: button (Left, Middle or
 *
    Right), state (button is x & y that tells the coordinate of
 *
    current position of move mouse
↪pressed or released),
 * */
void MouseClicked(int button, int state, int x, int y);
 *
```

3. Game Rules:

- You should fill the first two rows of the board by randomly selecting the alphabets.
- An alphabet will be present on the top of shot gun drawn at the bottom. This alphabet will get shot whenever user clicks on some area inside the board.
- You have to move the alphabet until it reaches the position specified by the user. You have to make sure that the alphabet never moves out of the board.
- An array named as ***dictionary*** is provided that contains 370099 words. Once the alphabet hits the user specified position you have to find all the combinations i.e. horizontal, vertical or diagonal alphabets that combines to form a word that matches some word of dictionary.
- Burst all the alphabets that forms the word. Make sure to consider the maximum length of the word in order to score more. For example if on finding the word ***take,*** before bursting do check that either the next alphabet in the row is making it ***taken*** or not**.**
- Whenever the alphabet bursts, all the slots gets empty.
- One point is awarded on bursting of the one alphabet.
- The game will be played for only 2 min. After that, the alphabets will be disappeared from the screen and user will be shown with game over status.

- When you fill the board randomly, then make sure that if any combination of alphabets makes any word then do burst all such words and fill the board randomly with some other alphabets.