Visvesvaraya Technological University

BELGAUM, KARNATAKA

ವಿಶ್ವೇಶ್ವರಯ್ಯ ತಾಂತ್ರಿಕ ವಿಶ್ವವಿದ್ಯಾಲಯ

ಬೆಳಗಾವಿ, ಕರ್ನಾಟಕ

COMPUTER GRAPHICS MINI PROJECT REPORT ON

"TYPE ATTACK"

Submitted to Visvesvaraya Technological University in partial fulfillment of the requirement for the award of degree of

Bachelor of Engineering

in

Computer Science and Engineering.

Submitted by:

Name	USN
Mohammed Moin Mirza	**4JN18CS048**
Zaheer Abbas	**4JN18CS128**

Under the guidance of

Mrs. Namitha M.V B.E., M.Tech.
Asst. Prof., Dept. of CS&E
JNNCE, SHIMOGA

Department of Computer Science & Engineering

J N N College of Engineering

Shivamogga - 577 204

2021

National Educational Society ®

J N N College of Engineering

Department of Computer Science & Engineering

CERTIFICATE

This is to certify that the mini project report entitled

"TYPE ATTACK"

Using OpenGL

Submitted by:

Name	USN
Mohammed Moin Mirza	**4JN18CS048**
Zaheer Abbas	**4JN18CS128**

Students of 6th semester B.E CSE in the partial fulfillment of the requirement for award of degree of the Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belgaum during the year 2020-2021.

Signature of Guide

Mrs. Namitha, B.E,,M.Tech
Asst. Prof., Dept of CS&E.

Signature of HOD

Dr. K.M Poornima M.Tech, Ph.D.
Professor & HOD, Dept of CS&E.

Signature of Examiners

Examiner 1: _____

Examiner 2: _____

ACKNOWLEDGEMENT

We would like to acknowledge our profound gratitude to all those who have helped in implementing this mini project.

We are grateful to our institution **Jawaharlal Nehru National College of Engineering** and **and Department of Computer Science Engineering** for imparting us the knowledge with which we can do our best.

We would like to thank our beloved guides **Mrs. Namitha M.V, Asst. Professor, Dept of CS&E, JNNCE** who have helped us a lot in making this project and for their continuous encouragement and guidance throughout the project work.

We would like to thank **Dr. K.M Poornima, HOD of CS&E Dept** and **Dr. Manjunatha P, The Principal, JNNCE, Shimoga** for all their support and encouragement.

Finally, we also would like to thank the whole teaching and non-teaching staff of Computer Science and Engineering.

Thanking you all,

Project Associates,

Mohammed Moin Mirza	4JN18CS048
Zaheer Abbas	4JN18CS128

ABSTRACT

In our project we are going to implement a Game called Type Attack. The basic idea of the game was that there would be a hero spaceship and alien ships. The user had to type the letters of the enemy ship to kill the enemy. Now, the game has the feature of hero spaceship, with which he can attack alien ship and get the updated score. The user will use the space bar key to shoot the enemy and move the spaceship using the right and left keys. The user gets point on each kill of the enemy ship, the points are based on kind of enemy he kills. The score gets updated every time a user kills the enemy ship. There are 3 kinds of aliens with each fixed points to them. The score gets updated according to the enemy that the user kills.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Preamble

1.1 Introduction

Pictorial synthesis of real/imaginary objects from computer-based models is Computer Graphics. Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

1.1.1 History of Computer Graphics

Computer Graphics is the creation, manipulation, and storage of models and images of picture objects by the aid of computers. This was started with the display of data on plotters and CRT. Computer Graphics is also defined as the study of techniques to improve the communication between user and machine, thus Computer Graphics is one of the most effective mediums of communication between machine and user. William fetter was credited with coining the term Computer Graphics in 1960, to describe his work at Boeing. One of the first displays of computer animation was future world (1976), which included an animation of a human face and was hand-produced by Carmull and Fred Sparkle at the University of Utah.

There are several international conferences and journals where the most significant results in computer-graphics are published. Among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on Graphics journals.

1.1.2 Introduction to OpenGL

As a software interface for graphics hardware, OpenGL's main purpose is to render two and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

OpenGL draws *primitives*—points, line segments, or polygons—subject to several selectable modes. Primitives are defined by a group of one or more *vertices*. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors, normal's, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way.

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

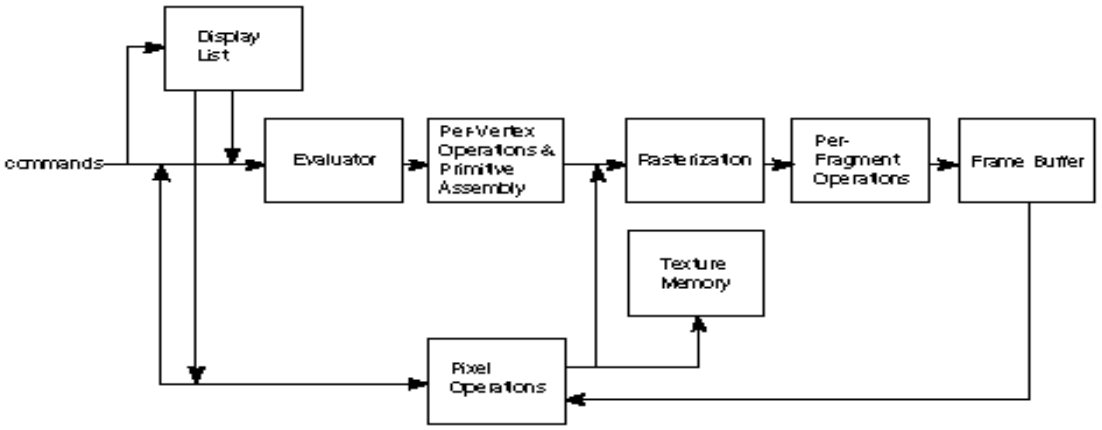

Fig 1.1: OpenGL Block Diagram

It can choose to accumulate some of the commands in a *display list* for processing at a later time. The *evaluator* stage of processing provides an efficient means for

approximating curve and surface geometry by evaluating polynomial commands of input values. *Rasterization* produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each *fragment produced* is fed into the last stage, *per – fragment operations*,which performs the final operations on the data before it's stored as pixels in the *frame buffer.*

1.2 Problem Statement

This project is about the "TYPE ATTACK". A Game where a user spaceship should defeat the alien spaceships and win the space war – The Game will have the following features in terms of technical perspective. The project is implemented in C++ and the Operating System is Ubuntu 20.04.

1. Collision detection of the enemy when the user spaceship fires a bullet.
2. Keep track of the score of the user, that is the enemies that have been killed.
3. Three types of enemies to make it a little bit more interesting.
4. All this will be done using Modern OpenGL using GLFW library

1.3 Objective

The objective of the project is:

a. Provide an Old Arcade Game kind of feel.

b. Detect collision of objects in the game.

c. Entertain the user with the game features that are currently present.

d. Learn about OpenGL.

1.4 Organization of the report

Chapter 1 provides the information about the basics of OpenGL. In Chapter 2, all the OpenGL functions used in our program are described. Chapter 3 gives the idea of the project and its actual implementation. Chapter 4 discusses the testing and limitations of the program. Chapter 5 concludes by giving the direction for future enhancement.

Chapter 2

A Preview of OpenGL Functions

When starting to draw any graphics in OpenGL using C language it needs a header file called <GLFW/glfw3.h>. The header file contains definitions and explanations of all functions and constants we'll need, whereas the graphics functions are kept in the library file. The library is to be downloaded and linked with the compiler GCC.

- **stdio.h:**
  This is a standard input header file which is used in any program. This file contains all the built-in functions like printf(),scanf(),fopen(),fclose() etc. It also contains data types and global variables. Some of the examples are BUFSIZ,EOF, and NULL etc.
- **stdlib.h:**
  This is also one the standard library header file which contains the entire standard library functions like exit, _exit,free, alloc, malloc etc.Some of the constants and data types are NULL,size_t etc.
- **GLFW/glfw3:**
  GLFW is a cross platform library that helps us create graphical outputs necessary for a particular Operating System. This header file contains a number of built in functions of a graphics library.

The different OpenGL functions used in our project are described as follows:

- **Name:** glfwCreateWindow()
- **Signature**: GLFWwindow* glfwCreateWindow(width, height, "Name", NULL, NULL)
  Description: Creates a window with the name of "Name", and with the width and height specified in the parameters
- **Name:** glfwMakeContextCurrent()
  Signature: void glfwMakeContextCurrent(window)
  Description: Makes the parameter as the current window in the context of operations.

- **Name:** glfwSetFramebufferSizeCallback()
  Signature: GLFWframebuffersizefun glfwSetFramebufferSizeCallback(window, function)
  Description: Resizes the window with the given function callback.
- **Name:** glfwSwapInterval( )
  Signature: void glfwSwapInterval(interval)
  Description: Swaps the front and back buffers based on the time mentioned in the parameter.
- **Name:** glGenTextures( )
  Signature: void glGenTextures (size, *Textures)
  Description: Creates texture objects in OpenGL with the specified parameters.
- **Name:** glGenVertexArrays( )
  Signature: void glGenVertexArrays(count, vertex_array_obj)
  Description: create a vertex array object in OpenGL so that the shaders can render the objects.
- **Name:** glCreateProgram( )
  Signature: GLuint glCreateProgram()
  Description: Creates a program to attach the vertex and fragment shaders to the OpenGL.
- **Name:** glCreateShader( )
  Signature: GLuint glCreateShader(shader_type)
  Description: Creates a shader of the type passed in the parameter .
- **Name:** glShaderSource ()
  Signature: void glShaderSource(shader_program, shader_code, int)
  Description: Stores the source code of shader in the buffer.
- **Name:** glCompileShader()
  Signature: void glCompileShader(shader_program)
  Description: Compiles the shader program with the given program source
- **Name**: glAttachShader()
  Signature: void glAttachShader(main_shader_program, shader_fragment)
  Description: Attaches the shader to the parameter passed program to create the program.
- **Name:** glLinkProgram()

Signature: void glLinkProgram(main_shader_program)

Description: Links the OpenGL shader objects to the current context of the program.

- **Name:** glUseProgram( )

  Signature: void glUseProgram(main_shader_program)

  Description: Allows us to use the shader programs created in the program.

- **Name:** glBindVertexArray()

  Signature: void glBindVertexArray()

  Description: Binds the vertex array object to the OpenGL program context to render the objects generated.

- **Name:** glfwWindowShouldClose( )

  Signature: int glfwWindowShouldClose(GLFWwindow* window)

  Description: Checks the close flag of the specified window. This function returns the value of the close flag of the specified window.

- **Name:** glTexSubImage2D( )

  Signature: void glTexSubImage2D(GLenum target, GLint level, GLint xoffset, GLint yoffset, GLsizei width, GLsizei height, GLenum format, GLenum type, const void *pixels)

  Description: Render the Textures as 2D Images onto the screen and pass the parameters as necessary.

- **Name:** glDrawArrays( )

  Signature: void glDrawArrays(GLenum mode, GLint first, GLsizei count)

  Description: Draws the OpenGL objects passed into the parameters on to the screen window.

- **Name**: glfwSwapBuffers( )

  Signature: void glfwSwapBuffers(GLFWwindow *window)

  Description: Swaps the front and back buffers of the specified window. This function swaps the front and back buffers of the specified window when rendering with OpenGL or OpenGL ES. If the swap interval is greater than zero, the GPU driver waits the specified number of screen updates before swapping the buffers. The specified window must have an OpenGL or OpenGL ES context.

- **Name:** glfwPollEvents()

  Signature: void glfwPollEvents();

Description: Processes all pending events. This function processes only those events that are already in the event queue and then returns immediately. Processing events will cause the window and input callbacks associated with those events to be called.

- **Name:** glClearColor()
  Signature: void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)
  Description: Clears the Color with color mentioned in the parameter on to the screen.
- **Name:** glBindTexture()
  Signature: void glBindTexture(GLenum target, GLuint texture)
  Description: Binds the texture to the current context of the OpenGL object and passes it to the buffer.
- **Name:** glTexImage2D()
  Signature: void glTexImage2D(GLEnum target, GLint level, GLint internal)
  Description: Creates a 2D Texture Image ready to be displayed on to screen and passes it on to the buffer.
- **Name:** glfwDestroyWindow()
  Signature: void glfwDestroyWindow(GLFWwindow *window)
  Description: Destroys the specified window and its context. This function destroys the specified window and its context. On calling this function, no further callbacks will be called for that window. If the context of the specified window is current on the main thread, it is detached before being destroyed.
- **Name:** glfwTerminate()
  Signature: void glfwTerminate()
  Description: Terminates the GLFW library. This function destroys all remaining windows and cursors, restores any modified gamma ramps and frees any other allocated resources. Once this function is called, you must again call.

Chapter 3

System Design and Implementation

3.1 Introduction

Systems design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development.

This Project is implemented using Modern OpenGl, which is proven to be a very efficient tool in the field of computer graphics, programming is done under Ubuntu 20.04 OS. GLFW library is used to create the static structs and to translate them. C++ programming language is used to implement the logical part of the code. Interface to the program is provided with the help of input devices keyboard.

The TYPE ATTACK game follows the modern OpenGL method, where we have defined a vertex shader, a fragment shader and a shader program to connect them. And the objects are drawn using the Textures.

3.2 Overall Design Process

There are three structs or static objects in our game: Player, Alien, Bullet and three utility structs to help draw the objects: Buffer, Sprite, and Game.

The Initial screen is a texture which draws a 2D texture. This texture is used to draw all the objects that are present in the game. A Vertex shader and a Fragment shader are created to render these objects on the screen. Texture is created after this, a Vertex Array Object (VAO) is created to send this data into the buffer.

We are making use of the following keyboard inputs to trigger the events on the game, with the help of glfwSetKeyCallback():

1. Spacebar (GLFW_KEY_SPACE) to fire bullets.
2. Right Arrow (GLFW_KEY_RIGHT) to move the Spaceship Right.
3. Left Arrow (GLFW_KEY_LEFT) to move the Spaceship Left.
4. Escape (GLFW_KEY_ESCAPE) to close the game window.

3.2.1 User defined functions

- **Name:** key_callback (GLFWwindow *window, int key, int scancode, int action, int mods)

  Signature: void key_callback(GLFWwindow *window, int key, int scancode, int action, int mods)

  Description: When a user clicks a particular key, glfwSetKeyCallback is called, in which we have to pass our key_callback function. The parameter key defines the type of key pressed and the necessary operation is performed.
- **Name:** framebuffer_size_callback (GLFWwindow *window, int width, int height)

  Signature: void framebuffer_size_callback(GLFWwindow *window, int width, int height)

  Description: Resizes the window of the game with respect to width and the height specified in the parameters.
- **Name:** validate_shader (GLuint shader, const char *file = 0)

  Signature: void validate_shader(GLuint shader, const char *file = 0)

  Description: Validates if the shaders are created without any errors.
- **Name:** validate_program (GLuint program)

  Signature: bool validate_program(GLuint program)

  Description: Takes the shader_program id as parameter and returns if the shader was created successfully.
- **Name:** buffer_clear (Buffer *buffer, uint32_t color)

  Signature: void buffer_clear(Buffer *buffer, uint32_t color)

  Description: Clears the screen the data and color passed in as the parameter, basically redraws an object with a specific color.
- **Name:** buffer_draw_text(Buffer *buffer, const Sprite &text_spritesheet, const char *text, size_t x, size_t y, uint32_t color)

  Signature: void buffer_draw_text(Buffer *buffer, const Sprite &text_spritesheet, const char *text, size_t x, size_t y, uint32_t color)

  Description: Draws the text passed in as the string in the parameter and also with the specified color.
- **Name:** buffer_draw_num(Buffer *buffer, const Sprite &text_spritesheet, const char *text, size_t x, size_t y, uint32_t color)

Signature: void buffer_draw_num(Buffer *buffer, const Sprite &text_spritesheet, const char *text, size_t x, size_t y, uint32_t color)

Description: Draws the score or the number passed as the parameter with the given color in the parameter.

- **Name:** buffer_draw_sprite(Buffer *buffer, const Sprite &sprite, size_t x, size_t y, uint32_t color)

  Signature: void buffer_draw_sprite(Buffer *buffer, const Sprite &sprite, size_t x, size_t y, uint32_t color)

  Description: Draws the passed in sprite (static object) with the given dimensions and uses a buffer object to store the data.

- **Name:** main(int argc, char *argv[])

  Signature: int main(int argc, char *argv[])

  Description: Main Code of the program, each frame call is made from and the animations and static objects are created and all the other functions are called here.

3.3 Pseudo code

```
Program :-  Type Attack
Functions:-

Function validate_shader - to validate if the shader is created successfully
Pass in: shader_id, error_log_file
Start
        Call glGetShaderInfoLog to get the info about the status of the shader
        if( error_string_length > 0 )
                Shader not created or Compile Error.
        Endif
End

Function validate_program - to validate if the shader programs are compiled and created
successfully.
Pass in: shader_program_id
        Call glGetProgramInfoLog to get the status of the created shader_program
        if ( errors )
                return false
        Endif
        return true
End
```

Function key_callback - to display the get the current key press from the user and perform necessary computations
Pass in: current_window_object, key, scancode, action, mods
Start
    switch(key)
        case ESCAPE_KEY : Stop the game & Close the current game window
        case KEY_RIGHT : Move the spaceship right
        case KEY_LEFT: Move the spaceship left
        case SPACEBAR: Fire the bullets
    EndSwitch
End

Function buffer_clear - to clear the screen with the color passed in as the parameter
Pass in: buffer, color
Start
    Clear the texture screen with passed in color
End

Function sprite_overlap_check - to check if there is any collision between two objects
Pass in: Sprite &s1, Sprite &s2, s1_position, s2_position
Start
    Returns a boolean if there is any collision between the two objects
    if (s1.width + s1_position <= s2.width + s2_position)
        return true
    Endif
    return false
Endfunction

Function buffer_draw_sprite - Draws the static object which is taken as an parameter in this function with specified color
Pass in: buffer, static_obj, color
Start
    Loop over the static_obj data
    static_obj contains the pixel where it is to be placed onto the screen
    Store the buffer data with the current static_obj passed on each iteration
    Draw the buffer data present with the passed in color
End

Function buffer_draw_number - Display the score update in number format on the screen
Pass in: buffer, text_spritesheet, number_to_display, color
    Loop over the text sprite sheet
    Display the number using the buffer object with the passed in color
    User buffer_draw_sprite to display the text
End

Function buffer_draw_text - Display the Score text
Pass in: buffer, text_spritesheet, text_to_display, color
Start
    Loop over the text sprite sheet
    Update the buffer data with the sprite sheet data for the required pixels
    Display the text passed in as the parameter
    User buffer_draw_sprite to display the text
End

Function framebuffer_size_callback - Resize the window according the viewport
Pass in: current_window, width, height
Start
    Call glViewport to resize the window
    Pass in the width and height to the glViewport function
End

Function main
Pass in: command line arguments (not used)
Start
    Setup GLFW window creation and load OpenGL function using GLEW
    Setup the main while loop and process keyboard input events
    Create vertex, fragment shader and a texture to draw static objects.
    Draw static objects (Aliens and User Spaceship) using the user defined functions
    Update animations of static objects using frame duration
    Check if there are fire press events and kill the aliens, update buffer data
    Delete the static objects at the end of the program to clean up memory
End

3.4 FLOWCHART OF AUTOMATED TELLER MACHINE:

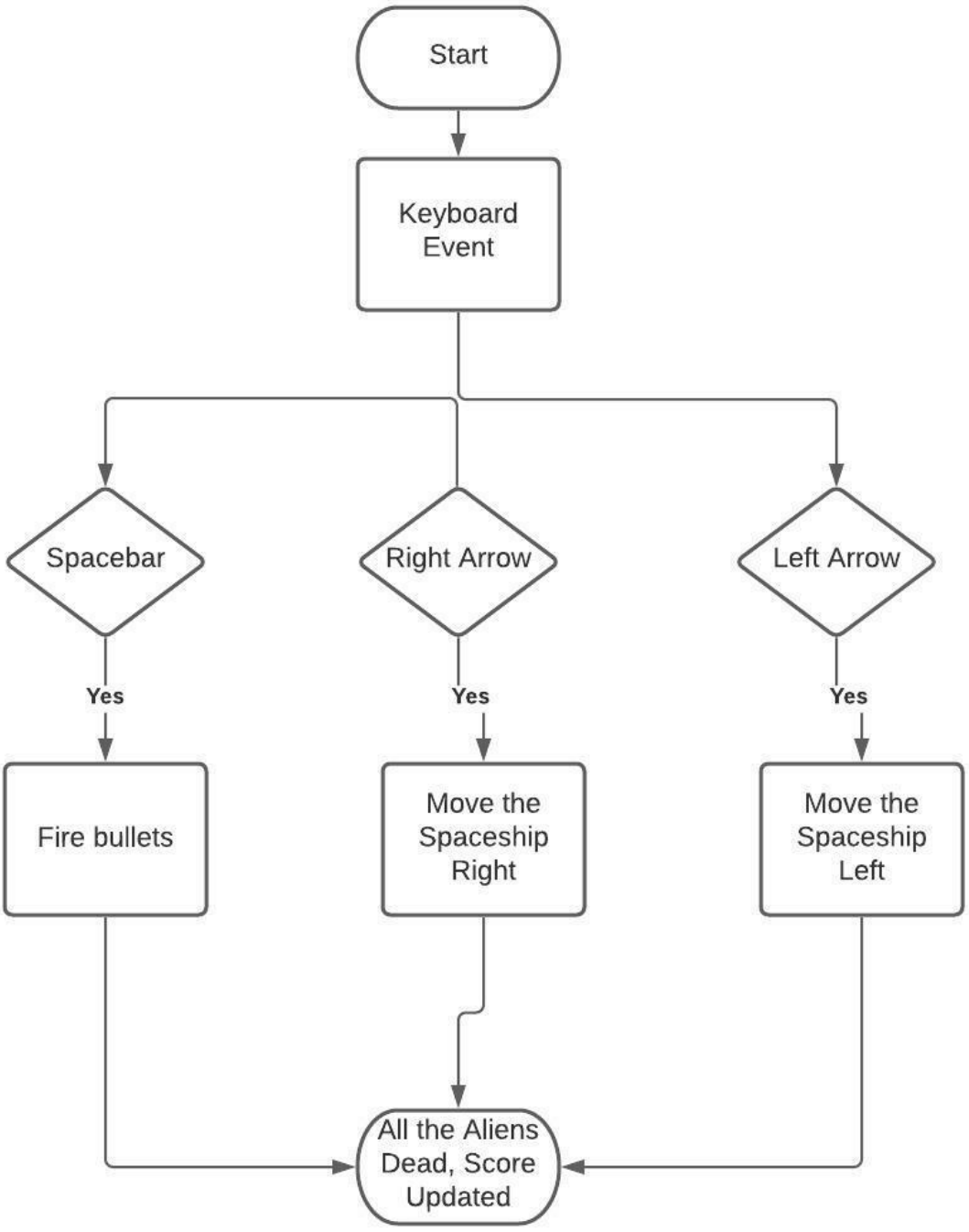

Fig 3.1: Flow chart for Type Attack

CHAPTER 4

RESULTS AND SNAPSHOTS

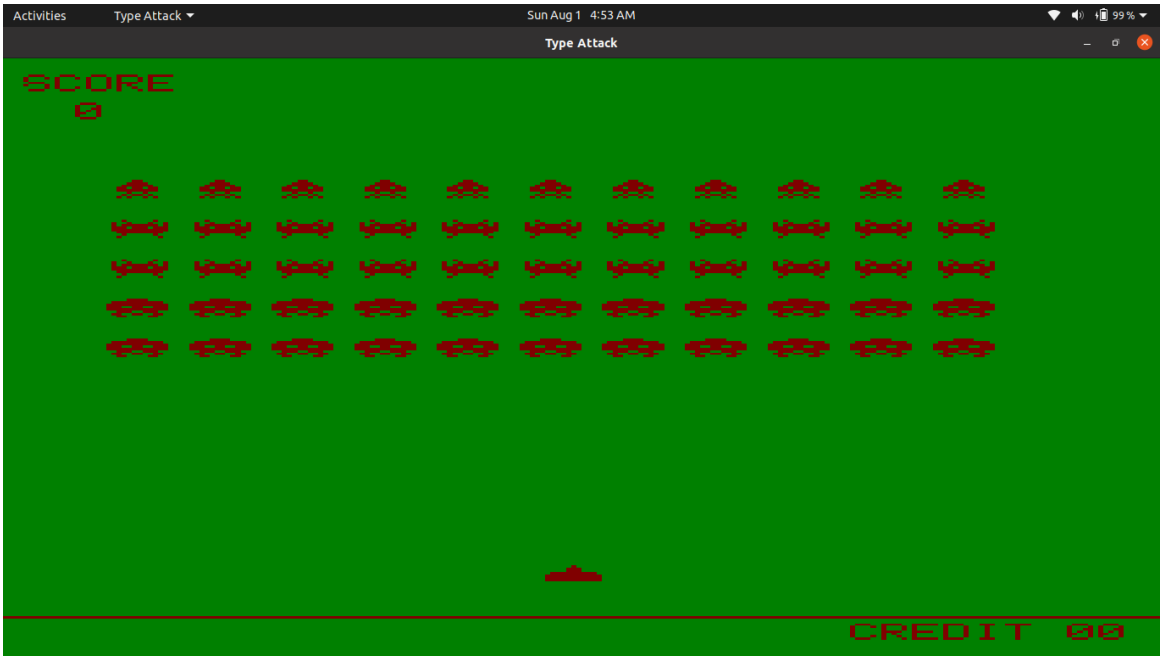

Figure 4.1: Initial Game Screen

This screen is the initial screen that the user is shown with to start the game.

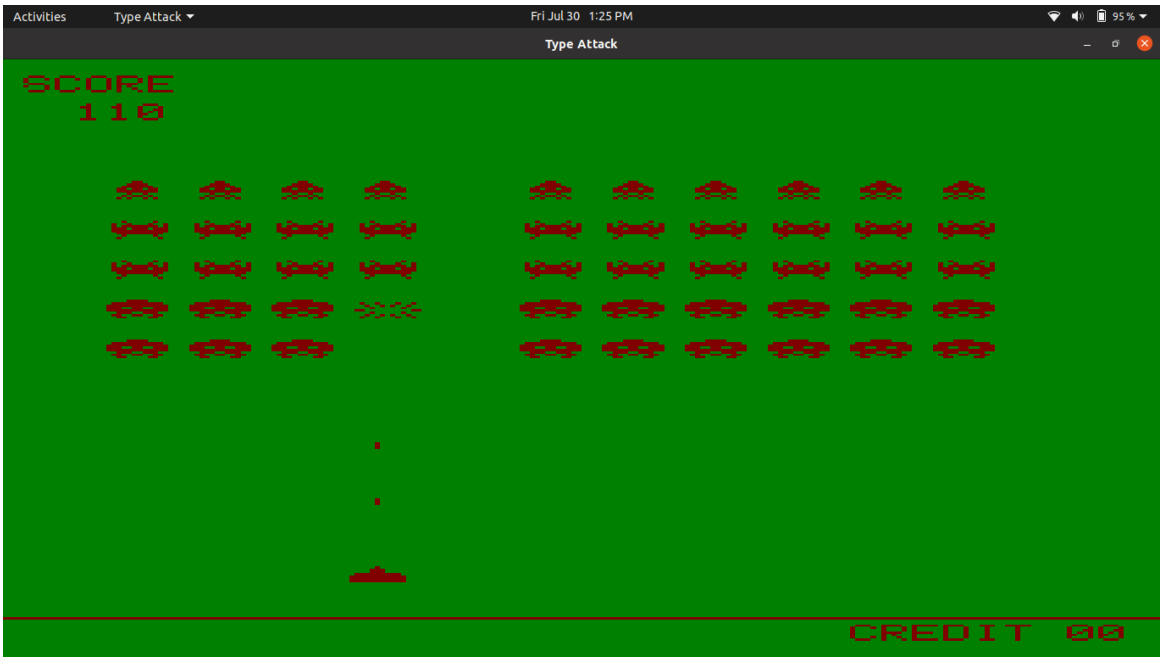

Figure 4.2: User Firing at Aliens

When the user presses the spacebar, the firing starts and the aliens are killed

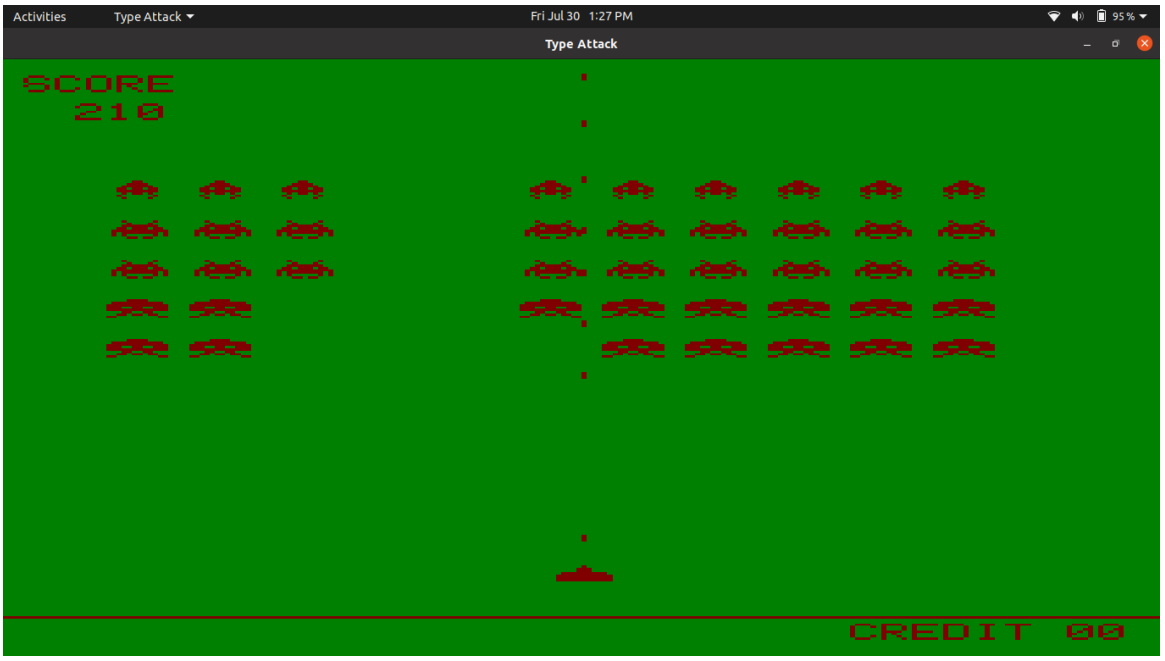

Figure 4.3: Aimless Firing

The user pressing the spacebar and shooting aliens aimlessly.

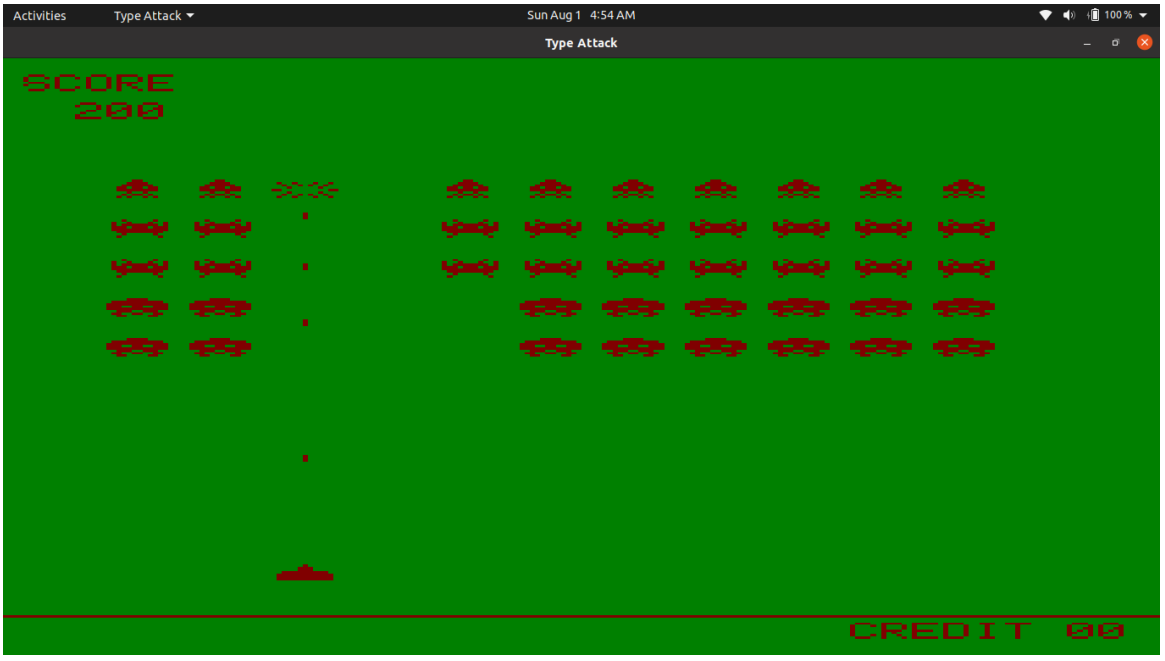

Figure 4.4: Kill top Alien to gain maximum points

The user can kill the top alien row to gain maximum points for each kill.

Figure 4.5: User Killing more Aliens

The User kills more aliens to win and the score is updated real time.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

This project allowed us to explore computer graphics as a real world experience, we made our project related to an old arcade game. We faced some good challenges while building this project. First challenge was to understand how the modern OpenGL pipeline works to the traditional OpenGL. Second was to understand how vertex and fragment shaders work to display the objects on the screen. Third was the texture details so that we can redraw the static objects present in our game. And we can go on. But all the learnings were worth it at the end of building this project.

This project helped us build some confidence regarding how low level libraries and the graphic pipeline works under the hood. And learning all this was fun. That was one of the main objectives of our project.

5.2 Future Scope

Further development in the project can be done by implementing some more features that are listed below.

Features that can be improved,

- ✔ Multiple component rendering screen, currently a single texture is used.
- ✔ Adding levels to the game, currently only one level is present in the game.
- ✔ Making the aliens attack the user spaceship.
- ✔ Adding power up features when the user hits the top level aliens.

REFERENCES

[1] Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version,3 rd / 4 th Edition, Pearson Education, 2011.

[2] Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL, 5 th edition.Pearson Education, 2008

[3] The OpenGL Programming Guide, 5th edition. The official guide to learning OpenGL version 2.1 by OpenGL Architecture Review Board.

[4] https://learnopengl.com/

[5] https://www.khronos.org/developers/books

[6] https://www.glfw.org/