University of Dhaka

Department of Computer Science and Engineering

Project Report:

Fundamentals of Programming Lab(CSE-1211)

Project Name:

GO CORONA GO!

Team Members

Ryana Binte Imtiaz
RK-48

Fardin Selim Khan
FH-43

1. Introduction

‘Go Corona Go’ is a Console Application developed mainly with the help of SDL2 (Simple Directmedia Layer 2) library. It is a 2D computer game. The project is done for the Fundamentals of Programming Lab (CSE- 1211) course, Department of Computer Science and Engineering, University of Dhaka.

The game is based on Covid-19 and social distancing awareness. Before starting the game, the user has to select from one of the four characters (COVID-19 Fighters) whom they have to keep safe through maintaining proper social distancing guidelines throughout the journey. The game comes in multiple levels, where the user has to pass the current level to unlock the next one. Difficulty gradually increases as the user levels up. In each level, the chosen COVID-19 Fighter first gets an invitation/call to a gathering (i.e examination hall, wedding reception, conference). When the fighter is all masked up and ready to attend the event, the timed level starts. The seating arrangement of the event is displayed where some of the seats are occupied and some are left empty. The occupants of the seats are of the following 2 types:

1) those wearing a mask (safe to sit directly around)

2) those not wearing a mask (unsafe to sit directly around)

To pass the level, the user has to pick an empty seat for their fighter which is safe within 60 seconds. The quicked this is done, the higher the level score will be.If the user fails to pick a safe seat within 60 seconds, they fail at the level.

The game also contains a quiz part based on the Coronavirus. When the user fails at any level, their fighter dies. But the user can revive the fighter (only once in a journey) by passing the ‘COVID-19 QUIZ’ segment of the game.

The final score is the sum of all the level scores.

The code is written in C programming language. The entire code is divided into a total of two source files (.cpp) and nine custom header files. Computer mouse and keyboard are used as game controllers to interact with the application.

2. Objectives

The game is designed specially for teenagers and young adults.

In such times of crisis when the whole world is captive in the hands of the Coronavirus disease, and we are stuck at our homes in isolation lockdowns after lockdowns, this game hopes to provide teens with moral support and raise awareness regarding social distancing and the deadly virus itself.

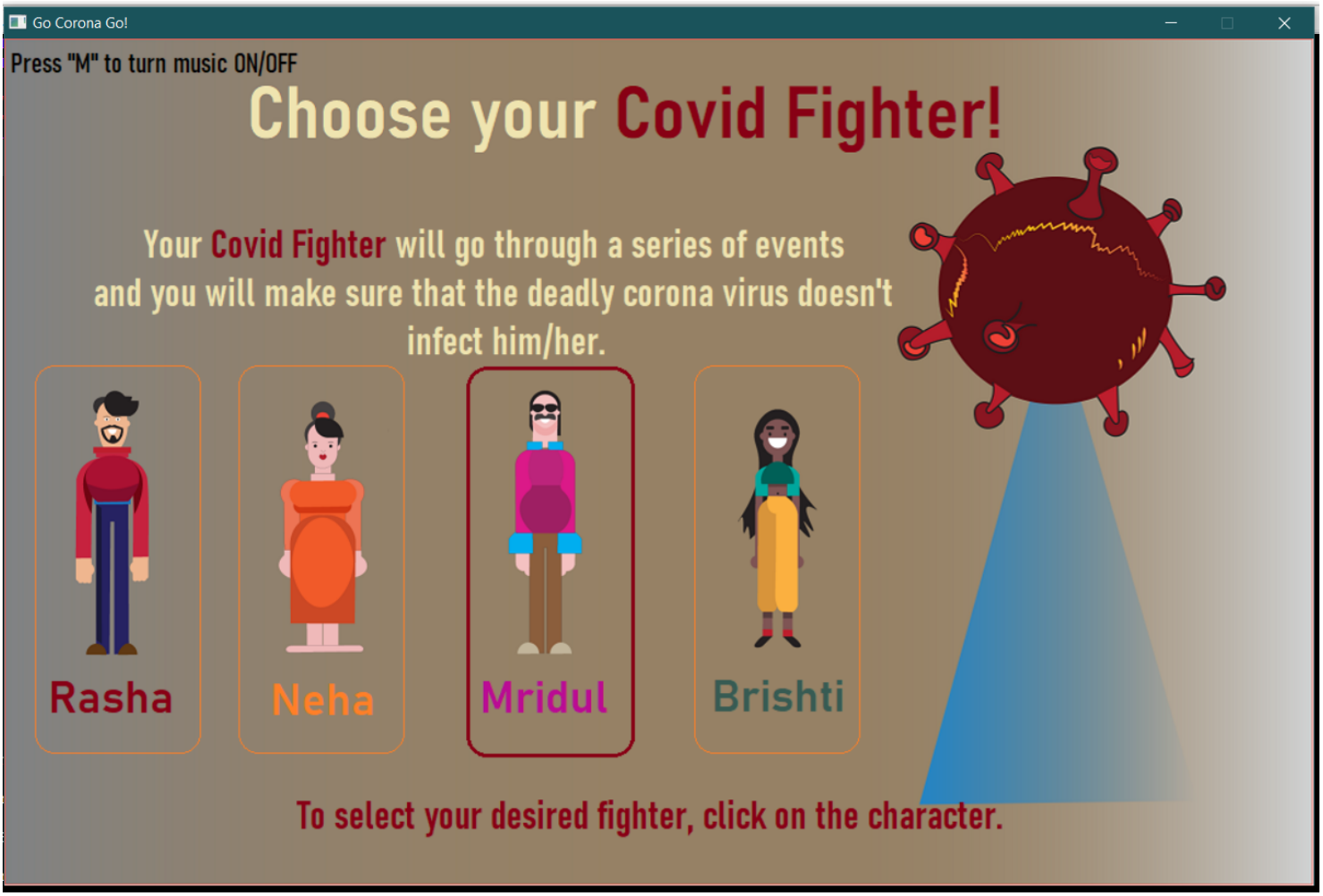
The game is divided into different levels with different setups so that it does not get monotonous for the users, as opposed to a single infinite level game. The different levels in the game can teach them about the importance of social distancing. It also tries to normalize the situation of maintaining safety guidelines so that they can adapt to the coronavirus ‘new normal’ better.

The ‘About Covid-19’ is an informative page where the teens can learn about the virus and the pandemic, which they can later use to win in the ‘COVID-19 Quiz’ section. This tries to educate them about the virus in a fun way.

3. Project Features

The project features are elaborated below:

1. Characters



The user has to choose a character before starting the game from choose character page shown in the above screenshot. The characters are uniquely designed for this game. The chosen character is then rendered as sprites in the journey ahead.

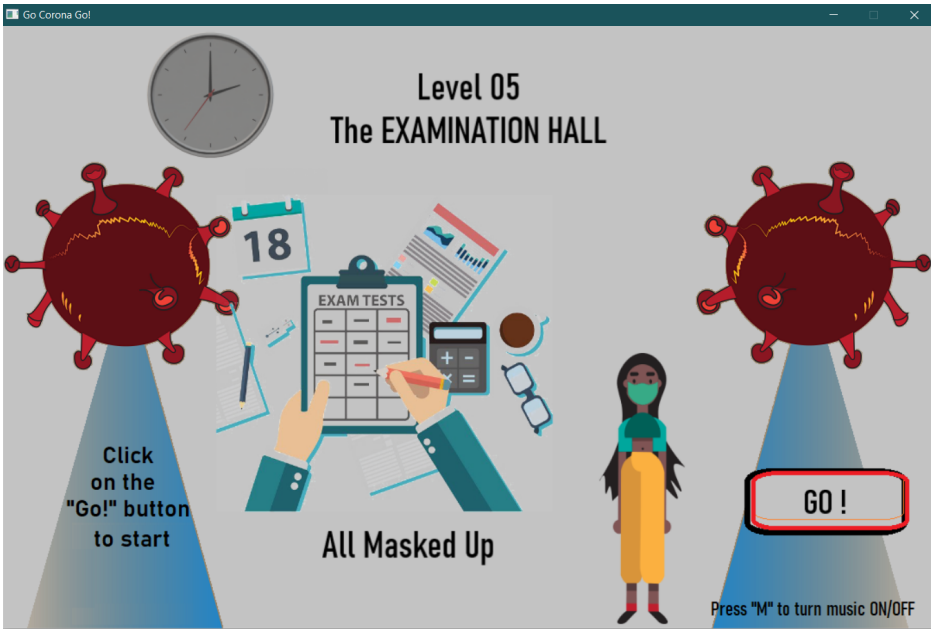
2. Levels:

Each level has three parts; the level invite where the character is invited to the event, the level event which is the seating arrangement, and the level outcome.

I. Level Invites

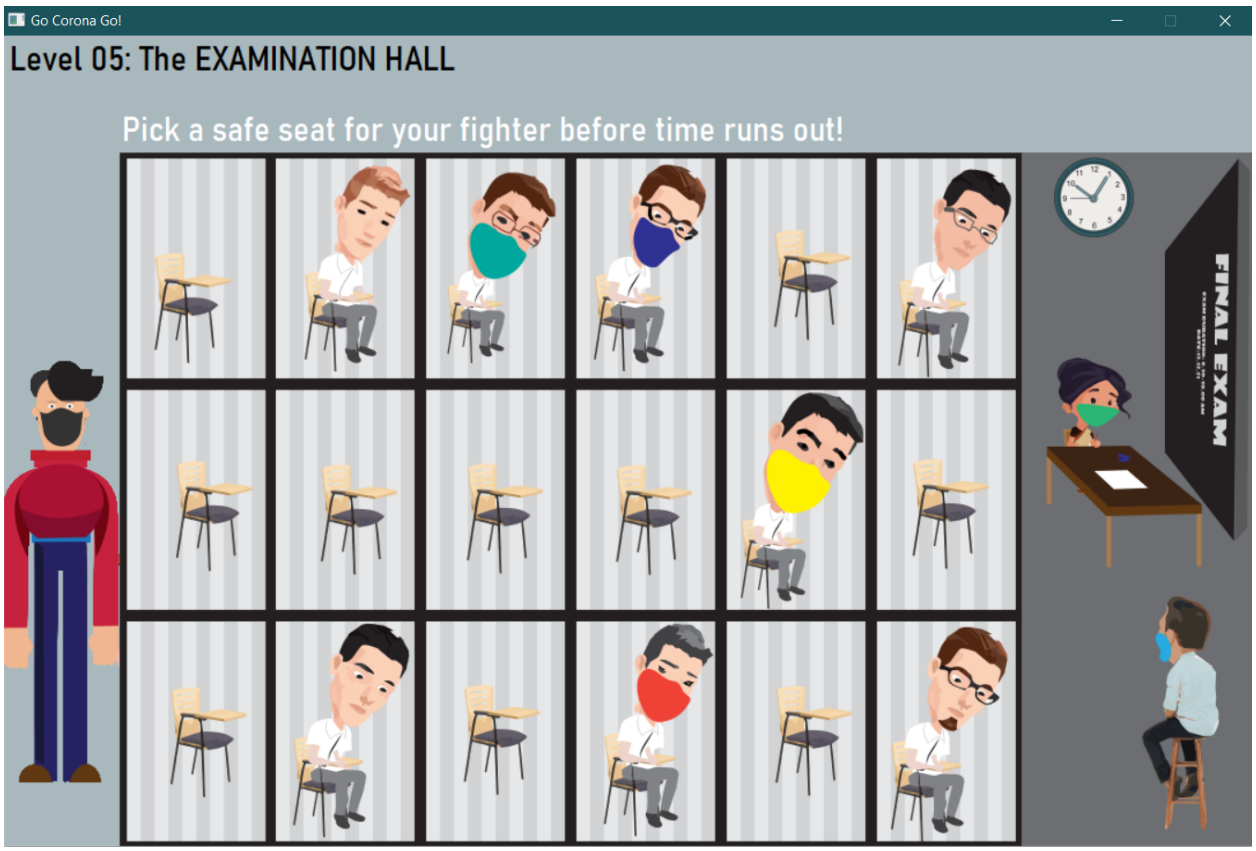
Level invite is displayed before the level event.

The chosen character is rendered along with a corner button that starts the timed level event when pressed.



II. Level Events

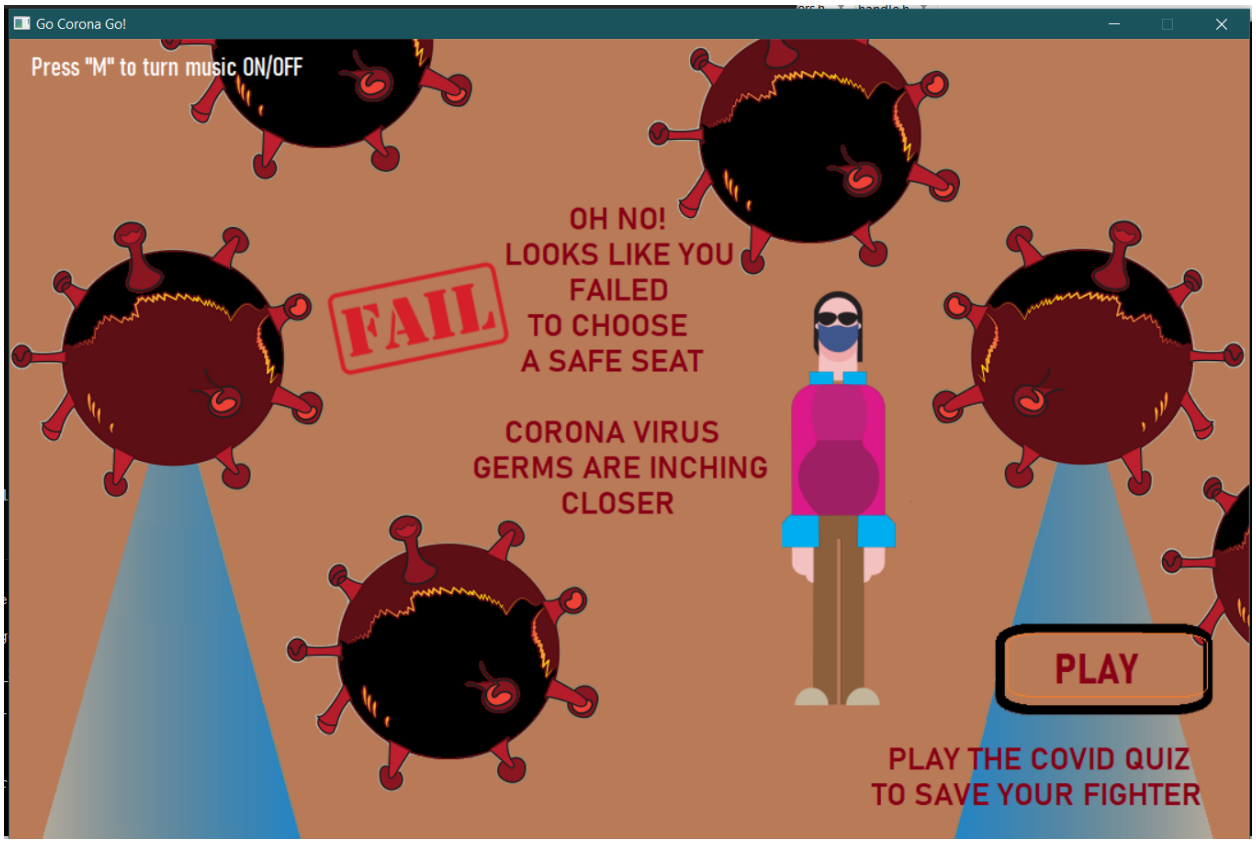
These are the main parts of the levels, and also the game.



A button is rendered for each unoccupied seat, whose outline turns red when the mouse is moved over it. Each level event has a unique theme and design.

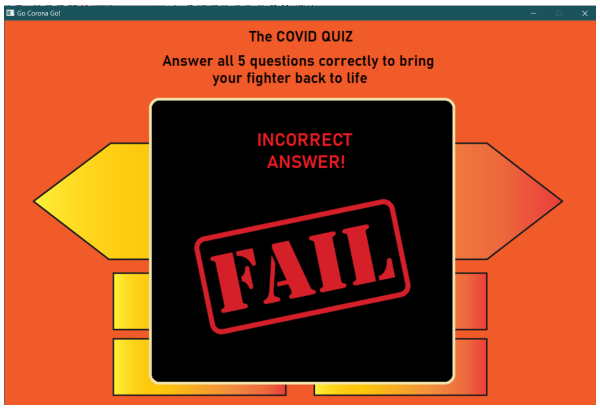
III. Level Outcome

This is the last part of the level feature. There can be five possible level outcomes after a level event; level success, level fail (quiz invite), time up, game over, and game win.



The above screenshot is the level fail (quiz invite) outcome. This outcome pops up when the user fails the level and has not played the covid19 quiz once already yet in his/her journey. The play button when clicked takes the user to the quiz section.

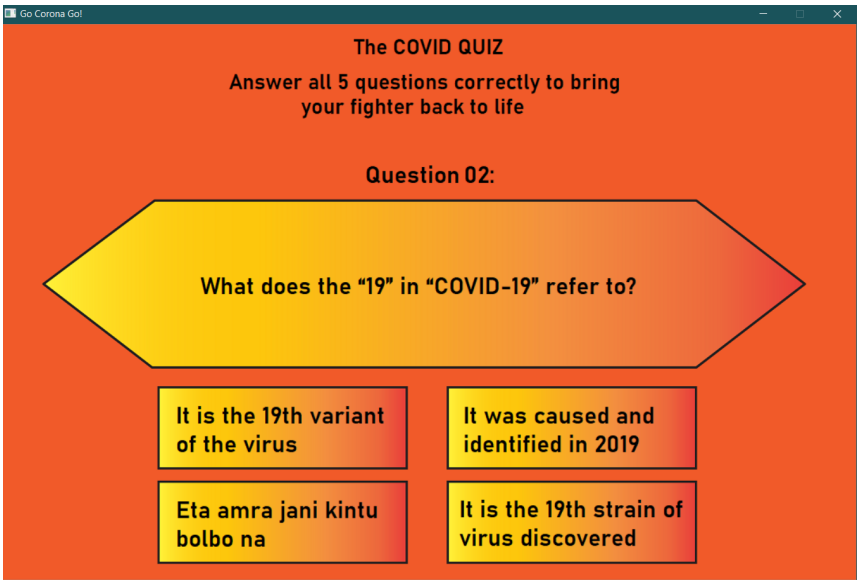
3. Quiz



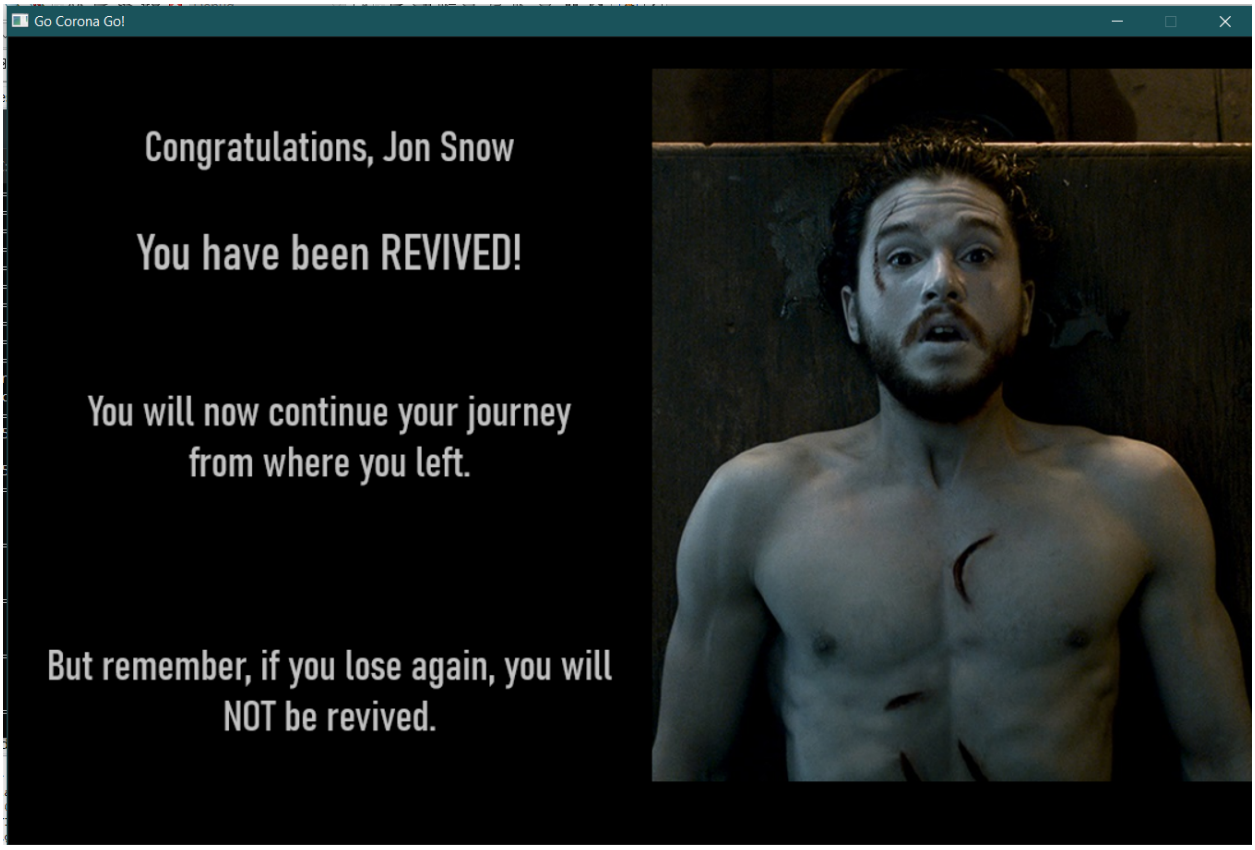
The quiz feature is added as an educational segment.

These are the two Quiz Outcomes. Quiz Fails results in Game Over. Correct Answer takes the user to the next question. If all questions are answered correctly, it takes the user to Revival.

4. Revival



This feature enables users to gain back a lost life.



The user is taken back to the level they failed, from where the journey continues.

5. Timing

Each level has a timer of 60 seconds, which when it goes off, causes the user to lose the level.

Timers are also used to calculate scores in each level.

6. Score Rendering

Level Score is rendered when Level Success is the Level Outcome. The Level Score is inversely proportional to the Time Taken. Both are rendered after passing of a level:

Total Score is calculated by adding all the Level Scores from the Levels passed. Total Score text is rendered in Game Over and Game Finish Texture.

7. Background Music and Sound Effects

At the very beginning of the game, a game background music starts to automatically play. To Stop/Start the background music at any point in the game, the user has to press the “M” key on the keyboard.

Unique sound effects are played for each of the following outcomes:

- Level Success
- Game Over
- Game Win

4. Project Modules

Two source files (.cpp) and nine custom header files are implemented in this project, which are written in the C Programming Language.

Source Files:

1. main.cpp

This source file calls different functions defined in the other files.

2. buttonPress.cpp

This source file contains the following functions:

- int pressedButton(int imgnum)

  This function is called upon button press on a texture that contains multiple buttons. It takes an integer value that indicates which image is on the display when the button is pressed, and returns an integer value that indicates which button was pressed. It uses the SDL_GetMouseState( ) function.

- int next_level_invite(int present_level)

  This function takes integer input that indicates the current level texture, and returns integer that indicates the texture of level invite of the next level.

- int next_image_show(int current_image)

This function takes integer input that indicates the current image texture, and returns integer that indicates the next image texture to be rendered.

- int Button_Level_Total(int image_num)

  This function takes integer input that indicates the current level texture, and returns an integer that indicates the total number of buttons to be rendered. The total number of buttons in any level event is the total number of unoccupied seats/slots.

Custom Header Files:

1. InitClose.h

This contains two function definitions called init() and closing(). call_init() is also there to optimise calling of the init function.
init() function is called right when the application starts. It initializes SDL, renderer color, SDL_ttf, SDL_mixer, and PNG loading by using IMG_INIT_PNG. It creates a Window and renderer for the Window. It returns a bool value that indicates the success or failure of the initialization process. This function is declared in the main.cpp file.
closing() function is called right before closing the application. It frees all the loaded images, sprites, sound effects, music, and global font. It then destroys the Window and quits all the SDL subsystems.

2. texture.h

This file declares all the Texture (SDL_Texture*) Pointers for loading media and rendering, the window renderer, the Window, the globally used font, Mix_Music for background music, and Mix_Chunk for different sound effects. It contains the Texture wrapper class which enables Texture Loading and Rendering, Color Modulation, and Alpha Blending.

3. loadmedia.h

Only one function is defined in this file which is the loadmedia() function. It loads all the images used in this game with the loadTexture("") function. It also loads sprites for all the buttons used, sets sprites, and sets the buttons in places. This function is called after the init function successfully completes.

4. rendermedia.h

This contains the definition of the renderMedia function, which takes an integer argument that indicates which media loaded Texture to render to the Window. It clears the screen using SDL_RenderClear(), renders textures to the screen using SDL_RenderCopy(), and updates the screen using SDL_RenderPresent().

For buttons, it alters the values of the global integer variables BUTTON_WIDTH and BUTTON_HEIGHT (declared in buttons.h) to set values according to demand. It then calls render_buttons() function (declared and defined in buttons.h) to render the buttons.

For Game Over and Game, it also renders Total Score text to the screen. For level invites, it updates the global integer variable revival_back_to (declared in buttons.h) so that it can keep track of the level invite texture that is to be rendered after the user enables the revival feature by passing the quiz segment.

For Character Sprites rendering, it loads the sprites and sets them in place. These are not rendered from this function. Character sprites are rendered from rendercharacters.h header file, which is a separate custom header file for rendering (chosen) character.

For music and sound effects, it loads the media. They are rendered from rendersound.h header file which is a separate custom header file for rendering music and sound effects.

5. rendersound.h

start_music(), control_music(), and music_render() are the functions defined in this custom header file.

start_music() is called at the very beginning of the game and it starts playing the game background music.

control_music() is used so that the user can press the “M” key to Start/Stop the background music. This function is called when the mentioned key is pressed. It

checks if music is being played with the Mix_PausedMusic() function. If the function returns value 1, it indicates that the music is not being played. In that case, it starts playing music with the Mix_ResumeMusic() function. Otherwise, it stops music with the Mix_PauseMusic() function.

music_render() function renders the sound effects for Level Success, Game Over, and Game Win. It takes an integer argument that indicates which sound effect to render. First, it checks if the music is being played and if it is, it stops the music, plays the sound effects, then starts the music again using Mix_PauseMusic() and Mix_ResumeMusic().

6. rendercharacters.h

Only one function is defined in this file which is the render_Character() function. It takes two integer arguments, one of which indicates the texture (image) in display so that the function knows exactly where on the screen to render the sprite. The other argument indicates which character the user is playing with so that it knows which sprite among the four loaded character sprites to render.

7. buttons.h

This is where almost all the global variables and user defined functions are declared. In memory text streams are declared for score and time taken texts, and text colors are set accordingly. Text color is set to black black for rendering text to Level Success texture, and to black for rendering text to Game Over/Win texture. Enumeration structure to hold button sprites is declared. Mouse button sprites array of type SDL_Rect are declared. Textures for button and character sprite sheets are declared.
The file contains a Class for mouse buttons.
LButton() initializes internal variables.
setPosition() takes two integer arguments as pixel coordinates and sets the top left position to that point.
handleEvent() takes argument of type SDL_Event* pointer which is also declared in this scope. It uses the SDL_GetMouseState() function to get the position of the mouse when the mouse is moved or clicked through, then checks if the mouse is inside the button. When the position of the mouse is inside the button, it checks if the cursor is moving over the button or if the button is being clicked. Then it sets

the globally used sprite accordingly. It returns an integer value which indicates if the button is clicked.
render() takes an integer argument that tells it which button sprite sheet texture to render to the screen. It renders the already loaded textures accordingly.

render_buttons() function takes two integer arguments, the first of which tells it which image is being displayed on the screen. The second argument tells it how many buttons are to be rendered to that screen. It then clears the screen, starts a loop the size of the second argument, and calls LButton::render(), passing an integer to indicate which button sprite sheet texture to render to the screen. Finally, it updates the screen using SDL_RenderPresent() passing the rendered as argument.

button_module() is called to handle button events for textures with single buttons. The function takes two integer arguments, the first of which tells it which image is being displayed on the screen. The second argument tells it which image texture to render next when the button is pressed. It calls the call_renderMedia() which calls the renderMedia() to render the current image and button textures. It calls handleEvent() to render button sprites and check whether the button is clicked. All of this is executed in the game loop. When the button is clicked, it returns an integer that indicates which image texture is to be rendered next.

level_button_check() function simply calls the handleEvent() to render button sprites and check whether any button is clicked on any Level. It returns an integer that indicates if any button was pressed.

8. navigate.h

This file contains the definitions of two functions; clic2view() and Time_Up(), both of which are declared in the buttons.h header file.

Time_Up() function calls the call_renderMedia() which calls the renderMedia() to render the Texture when the 60 seconds time limit on any level is passed. It then calls the render_Character() function defined in the rendercharacters.h file to render the chosen character to the screen. It calls clic2view() and returns the return value it gets from clic2view() to the caller. This function is called when the 60 seconds time limit on any level is passed.

clic2view() is called when a button on a screen with multiple buttons is clicked. It decides which image to display after clicking which button on which texture. The function takes two integer arguments, the first of which tells it which image is being displayed on the screen. The second argument tells it which button on the screen has been clicked. It calls the call_renderMedia() which calls the renderMedia() to render the Texture accordingly. It calls the render_Character() function defined in the rendercharacters.h file to render the chosen character if required.

9. handle.h

Only one function is defined in this file which is the handle() function. This function is directly called from the main() function of the main.cpp source file. The global integer variable image_number (declared in buttons.h) which indicates the texture at display is passed to this function as the argument. The function does not have a return value, but it alters the value of image_number, button_number, char_num, all of which are global integer variables.
It calls the call_renderMedia() which calls the renderMedia() to render the Texture accordingly. To handle button events, it sets up a loop the size of total buttons in the screen, and calls handleEvent() to render button sprites and check whether the button is clicked. When handleEvent() returns a value that indicates that a button was clicked, it calls the pressedButton() function (defined in pressedButton.cpp) to find out which button was clicked on that texture. After that, it calls the clic2view() function, passing image_number, and the return value from pressedButton() to the function so that clic2view() function knows which button was pressed on which image texture.

For levels, it uses SDL_GetTicks() to calculate time and score. It calls the Time_Up() function (defined in navigate.h) when the time is up.
For Level Success Level Outcome, it keeps updating the total score. It also renders the time taken and score text to the screen.

5. Team Member Responsibilities

File Name	Implemented by
main.cpp	Ryana Binte Imtiaz
buttonPress.cpp	Fardin Selim Khan
buttons.h	Ryana Binte Imtiaz
texture.h	Fardin Selim Khan
InitClose.h	Fardin Selim Khan
loadmedia.h	Fardin Selim Khan
rendermedia.h	Ryana Binte Imtiaz
navigate.h	Ryana Binte Imtiaz
rendersound.h	Ryana Binte Imtiaz
rendercharacters.h	Ryana Binte Imtiaz
handle.h	Ryana Binte Imtiaz

6. Platform, Library & Tools

The code is written in the C programming language.
IDE CodeBlocks was used for the project.
The libraries used are

- SDL
- SDL_image
- SDL_ttf
- SDL_mixer

7. Limitations

The following goals could not be achieved in this project:

- Using files to store information that would enable leaderboard feature (where highscores are displayed) and saving progress that would enable continuation from where a user left the game.
- Display of animations

8. Conclusions

Doing this project has helped us learn the C programming language in a fun and interesting way. It has also served to improve our team working, collaborating, and leadership skills. This is our first ever project and it seemed difficult and incomprehensible at the very beginning when we were assigned with it. But when we started working on it and as we learned more, it all seemed to unfold with ease. Although the project has mentionable limitations and all the targeted goals could not be achieved, it was an enjoyable experience and we are satisfied with our final outcome at the moment. It does however have a lot of room for improvement.

9. Future plan

We plan to do the following things with our project in the near future:

- Use files to store information that would enable leaderboard feature (where highscores are displayed) and saving progress that would enable continuation from where a user left the game.
- Display of animations
- Adding more levels to the game
- Improving the graphics

Repositories

GitHub Repository: https://github.com/ryana242/Go-Corona_Go.git

Youtube Video: https://youtu.be/DcqXeqQ-1AQ

References

- *SDL tutorials from Lazy Foo's Production : https://lazyfoo.net/tutorials/SDL*
- *Transparent background open source png images from Pngall : http://www.pngall.com*
- *Sound effects from https://freesound.org/*