University of Dhaka
Department of Computer Science & Engineering

Course Name : CSE-1211

Project name : Pong Classic

Submitted to :

Mr. Abu Ahmed Ferdaus
Associate Professor,CSEDU

Mr. Hasnain Heickal
Assistant Professor,CSEDU

Submitted by :

1.Tasmim Rahman Adib
(SH-19)
Department of Computer Science & Engineering
Session : 2018-19

2.Ahmed Zaman Pranta
(FH-43)
Department of Computer Science & Engineering
Session : 2018-19

Date of Submission : November 19,2019

Table of Contents :

Introduction :

The project is intended to make the students familiar with some application of basic C programming using Simple DirectMedia Layer 2(SDL2). Implementation of the students theoretical knowledge of C language in some real life scenario was the objective of this project.

Project Introduction :

Our game is a 2D based game which is basically a single user game. While making our game, we'd kept our mind that we should make a game which can be played in quick and as well as remove monotony. The game is built over the idea of user view.The opponent of this game is CPU. It is a simple ping pong balancing game which is built under the logarithm of balancing and accuracy hitting.

Game Outline :

This is a simple pong game which is created over the ideas of "Balancing" & "Perfect hitting".

User has to compete with CPU here which is named "BOT" in the game. A ball is served from the center to a random direction.User has to hit the ball with his hitting techniques and balancing strategy. He's to hit the ball in order to make the ball out of opponents window. Whenever he's able to do so,he would score a point.

Previous rules are also applicable for the CPU(BOT).

The velocity of ball gets increased after every 6 hits from both sides(If a game runs to minimum 6 hits,otherwise not applicable). The player(User or CPU) whoever would be able to score 5 points first,would be declared winner.

Game Objective :

The Game was designed with the aim to keep the player busy always. Before starting on the project I had thought about the conception of “great games” and “boring games” and finally decided that a game that requires the player to be alert all the time is less likely to be considered as “boring”. So, the main objective of the game was to provide its user an entertaining experience, and also to make the game as effective for brain as possible.

Challenges for gamers :

While playing our game,user has to be very concentrate while the ping pong ball is served from the center. This is so as the ball is generated by a random function value, for which it can be served to any direction. While the ball is in play,the players will try to keep the ball in range of it’s window.If the ball goes out of one’s window,the opponent will get a point. While playing this game,the users will brainstorm their accuracy in balancing and perfect hitting.So being concentrated is a must to this game.

Game features :

1. **Interactive Menu :** This is the opening door towards our project.So we’d to make this one simply user-friendly and attractive. Though we tried our best to make the menu outlook attractive, but due to our shortage of knowledge we couldn’t work on that that much. But we’ve worked whole-heartedly to make our game easy to play.

Our game consists of 4 options :

1. **New game** : Starting of a game.
2. **High Score** : Keeping the record of highest term play.
3. **Credits** : Some descriptions about the contributors.
4. **Quit** : Exit from the game.

2. An eye-soothing interface : While making the interface of different things,we've kept the idea of "differentiating different objects clearly" . Thus we've used a B&G color background interface texture in order to differentiate both the players. The colors of paddles and balls are not the same.

Source code properties :

1. The source code was written by following an efficient modular programming system, dividing the whole code in different separate parts and making new user defined header files. This method was encouraged by our teachers and it made our tasks a lot easier while combining all the features and joining the whole project together.

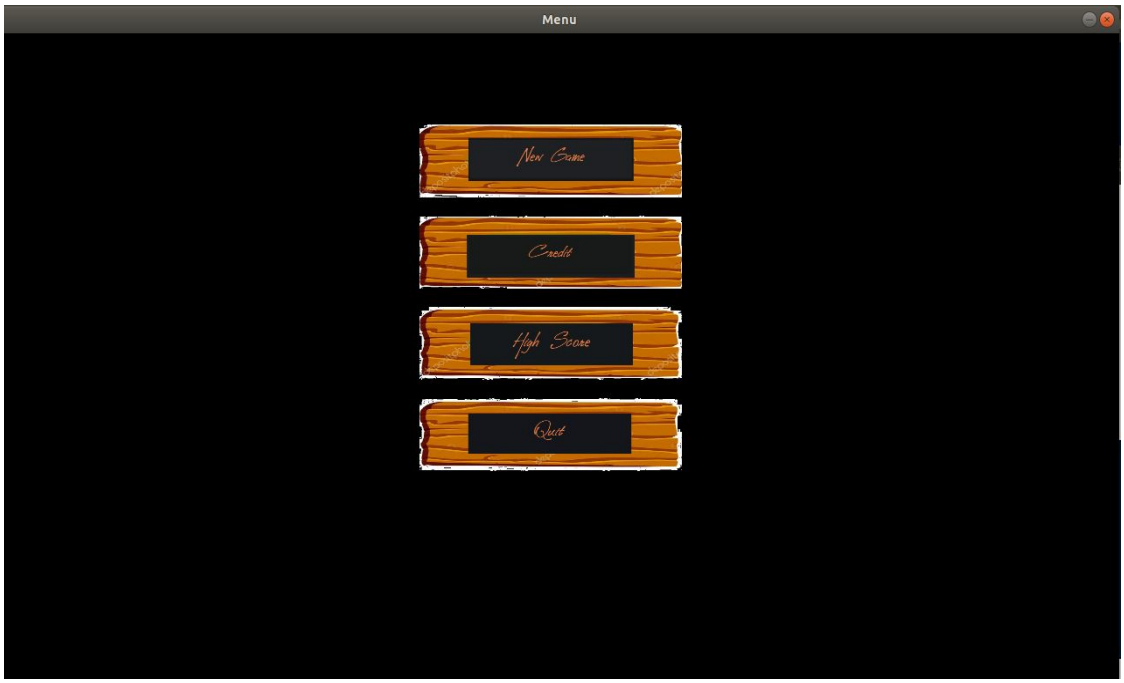
2. We tried to use as many different types of programming tools as possible, so that it covers the maximum amount of what we have been taught in C language. The coding part of this project contains :

   a. Basic input output
   b. File input output
   c. Conditional logic
   d. For loop and While loop
   e. Single and multi dimensional arrays
   f. String handling
   g. Mouse operations
   h. Keyboard operations

i. Pointers
j. Structures
k. Sorting
l. Reading image files
m. Graphics operations And so on.

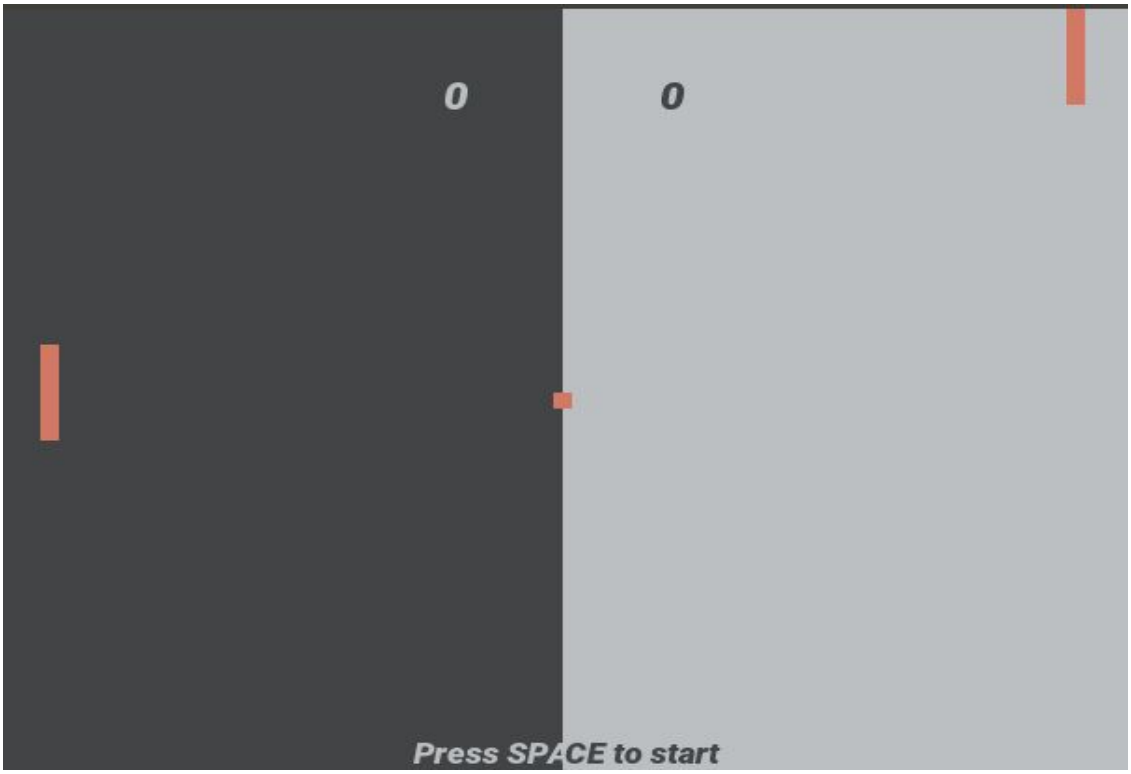
Graphical interfaces :

Main Menu :

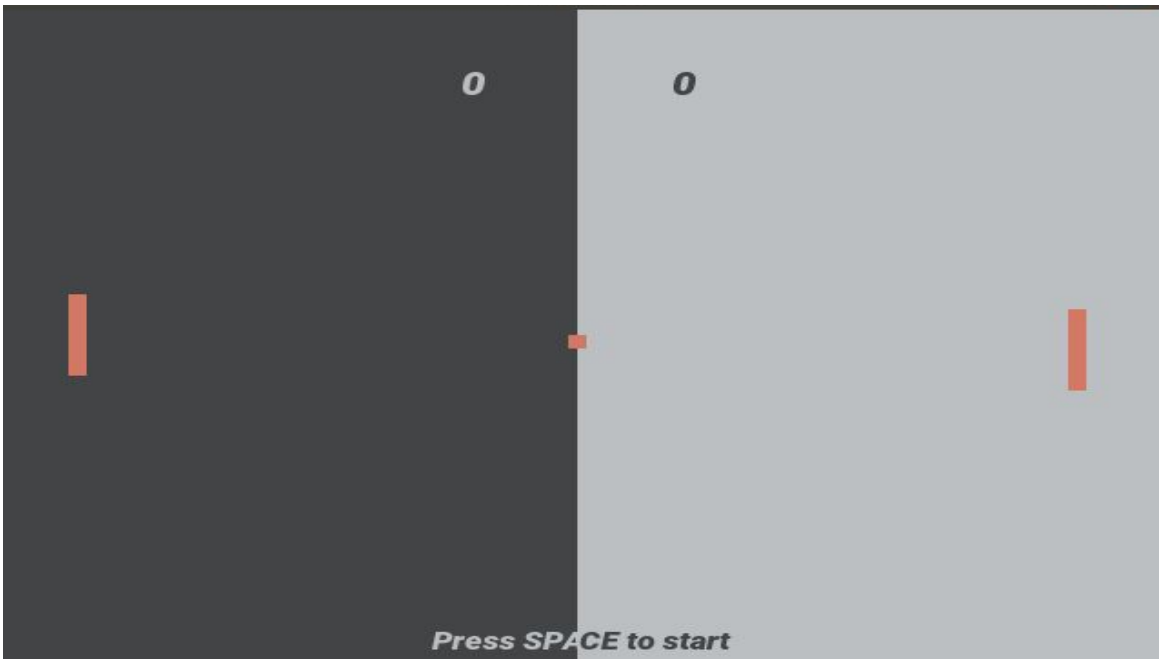


Classic B&W background interface used in the game:

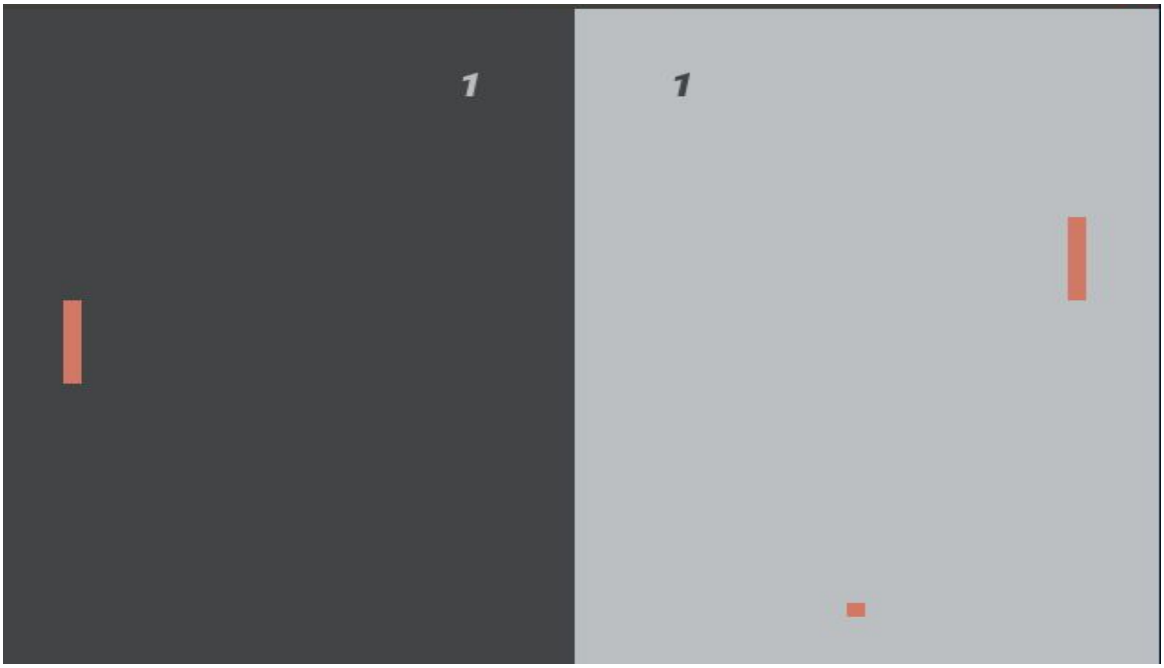
Movement of paddles :



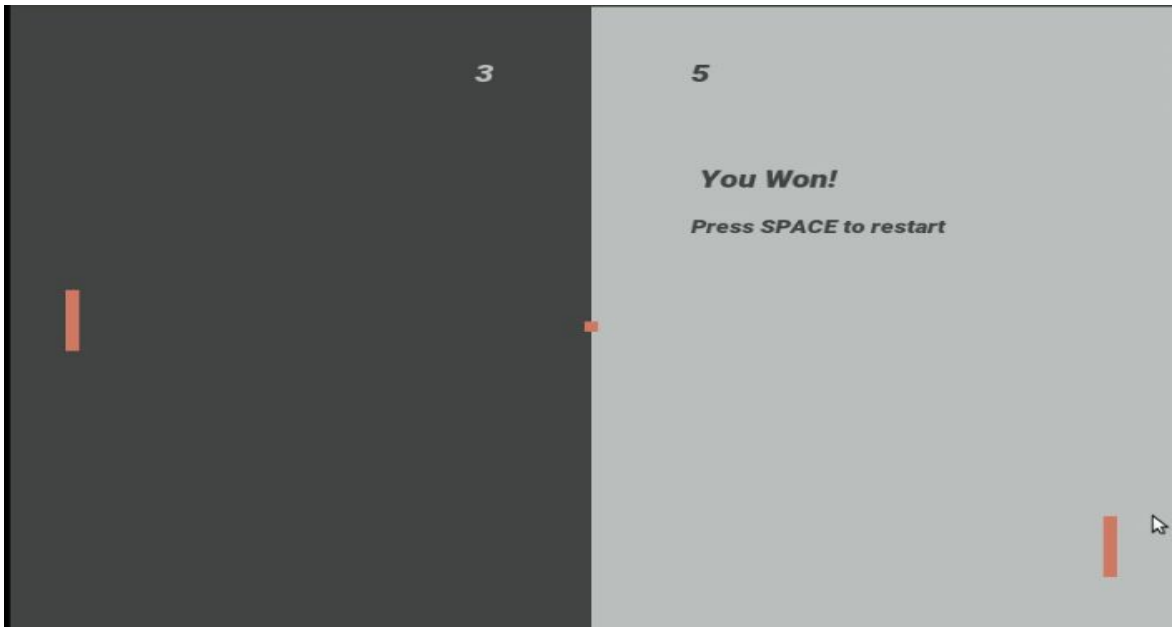
Initializing a game :



Game is in play :

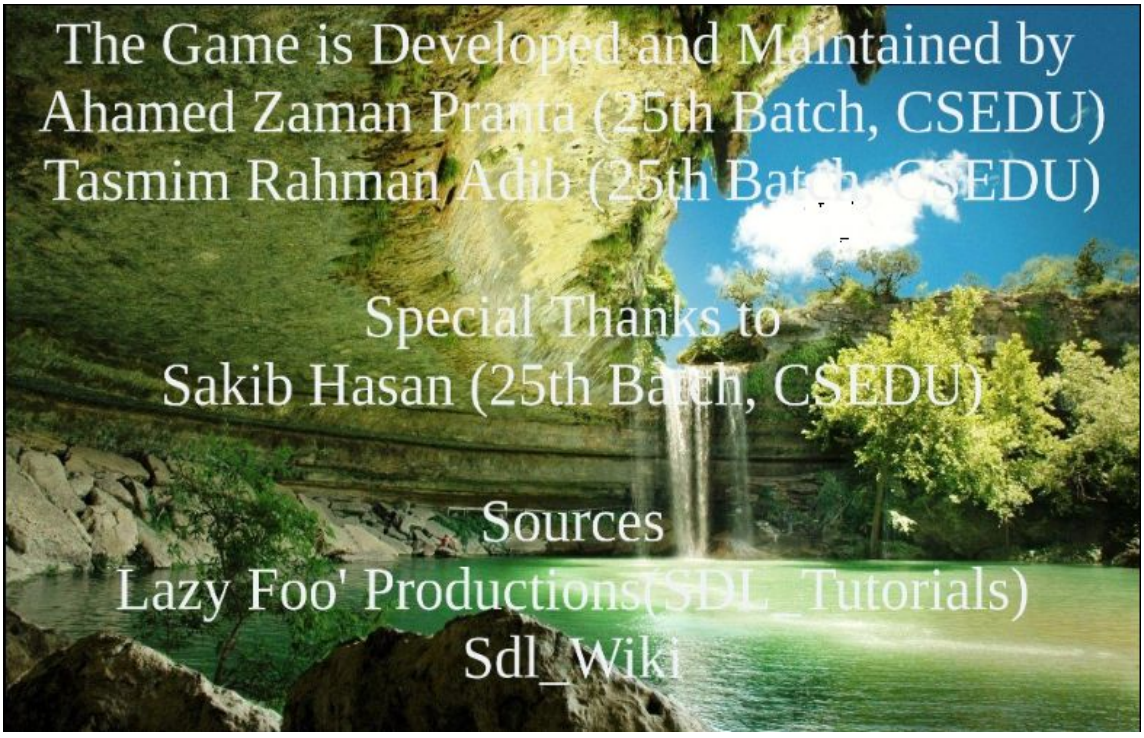


Winner declaration image :



High Score :

Credits :



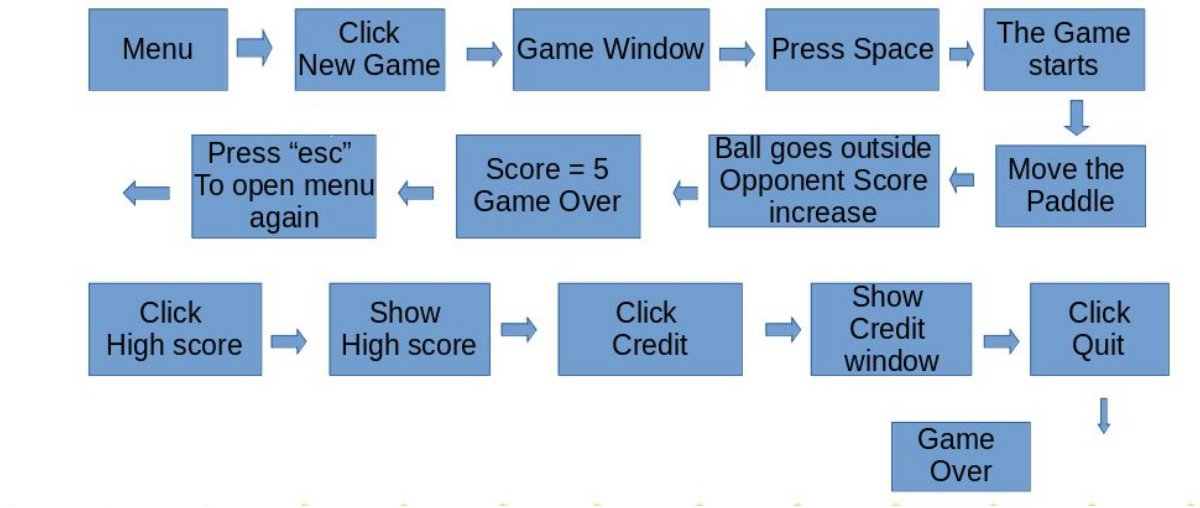
Exit :

User can exit from a math anytime by shutting down the window.There's also an option of exit in main menu.We've also added a feature of pausing the game.User can pause the game with "Esc" in keyboard.

How the game works :

Our target was to make a easy going and lag-free game.So we added a less touch of complexity while functioning the game. The logic steps of the game play operation are not very complex. The basic idea of a match is shown below with the help of a flow diagram.

Flowchart of total gameplay



Challenges while coding :

Though our experience of working was worth mentioning,but the way of working wasn’t that much soothing either.We faced some problem with SDL2 graphics design library as there is still lack of reliable tutorials to know about the functions.
Our interactive menu was initially done by calling one function from another. This caused the functions to become recursive and may sometime fall into deep holes of recursions. This was handled by re designing the whole menu operations using only break functions and looping and minimal function calling.
However,after handling all the problems we faced,we’re happy to report that our game is a bug-free one.

The functions used in source code :

1. bool initMenu() : to build up the menu window.
2. void loadTextureMenuEnter(string path) : creating New Game tab menu window.
3. void loadTextureMenuExit(string path) : creating Quit tab in menu window.
4. void loadTexturelevel(string path) : creating Credit tab in menu window.
5. void loadTextureMenuhighscore(string path) : setting High Score tab in menu window.
6. bool initcredit( ) : to build up the Credit window.
7. void loadTexturecredit(string path) : to add a pic in Credit window.
8. bool inithighscore( ) : to build up the High Score window.
9. void sethighscore( ) : to set the high score in High Score window.
10. int predict( ) : this function give a prediction of the ball position which help us to set the left paddle (conducting with AI) .
11. void input( ) : we get all the mouse and key board input in this function.
12. bool checkLeftCollision( ) : detection the ball whether it collides with the left paddle.
13. bool checkRightCollision( ) : detection the ball whether it collides with the right paddle.
14. void update( ) : this function we use to update all the values like score, positon of paddle etc.
15. void render( ) : we use it to render all the things like paddle, ball, pong board etc.
16. void cleanUp( ) : to clear all the window.
17. void gameLoop( ) : it is the main game loop function.
18. void initialize( ) : to initialize the main window, ttf texture, sdl renderer, sdl window, sdl texture etc.
19. int main(int argc, char *argv[ ]) : we just call up all the function in the main function.

List of Header files :

1. SDL2/SDL.h
2.SDL2/SDL_image.h
3.SDL2/SDL_ttf.h
4.cmath
5.ctime
6.iostream

General overview :

~The project was a very good way to encourage us to develop our programming skills and make something useful with our knowledge.
~This project also enabled us to know many unknown stuffs about C/C++ programming.
~This project is a perfect example of all the exceptional things that can be done with some basic C/C++ programming.
~The project promotes originality and creativity and the skill of working in a team.
~The project also summarizes the basic tools of C language as they had to be implemented while making the game.

The source code :

The main part :

```
int main(int argc, char *argv[ ]) {

    srand(time(NULL));
    initialize();
    gameLoop();
    return 0;

}
```

Overall code :

```
#include <SDL2/SDL.h>
#include<SDL2/SDL_image.h>
#include <SDL2/SDL_ttf.h>
#include <cmath>
#include <ctime>
#include <iostream>
using namespace std;

SDL_Window*     window;
SDL_Renderer*   renderer;

SDL_Texture*    font_image_score1;
SDL_Texture*    font_image_score2;
SDL_Texture*    font_image_winner;
SDL_Texture*    font_image_restart;
SDL_Texture*    font_image_launch1;
SDL_Texture*    font_image_launch2;

bool initMenu();
```

```
void loadTextureMenuM( string path );
void loadTexturelevel( string path );
void loadTextureMenuExit( string path );
void loadTextureMenuEnter( string path );

bool initcredit();
void loadTexturecredit( string path );

bool inithighscore();
void loadTexturehighscore( string path );
void sethighscore();

SDL_Window *menuWindow, *creditwindow, *highscorewindow;
SDL_Renderer *menuRenderer, *creditrenderer, *highscorerenderer;
SDL_Texture *menuTextureEnter, *menuTextureExit, *menuTexture, *menuTextureMenulevel,
*menuTexturehighscore, *Texturecredit, *Texturehighscore;

// Screen resolution
int SCREEN_WIDTH = 840;
int SCREEN_HEIGHT = 620;

//menu screen resolution
const int SCREEN_mWIDTH = 1280;
const int SCREEN_mHEIGHT = 720;

void loadTextureMenuM( string path )
{
   SDL_Surface *tSurface = NULL;
   tSurface = IMG_Load( path.c_str() );
   SDL_SetColorKey( tSurface, SDL_TRUE, SDL_MapRGB( tSurface->format, 255, 255, 255));
   menuTexture = SDL_CreateTextureFromSurface( menuRenderer, tSurface);
   SDL_FreeSurface( tSurface );
}

void loadTextureMenuEnter( string path )
{
   SDL_Surface *tSurface = NULL;
   tSurface = IMG_Load( path.c_str() );
   SDL_SetColorKey( tSurface, SDL_TRUE, SDL_MapRGB( tSurface->format, 255, 255, 255));
   menuTextureEnter = SDL_CreateTextureFromSurface( menuRenderer, tSurface);
   SDL_FreeSurface( tSurface );
}

void loadTextureMenuExit( string path )
{
   SDL_Surface *tSurface = NULL;
   tSurface = IMG_Load( path.c_str() );
   SDL_SetColorKey( tSurface, SDL_TRUE, SDL_MapRGB( tSurface->format, 255, 255, 255));
```

```
    menuTextureExit = SDL_CreateTextureFromSurface( menuRenderer, tSurface);
    SDL_FreeSurface( tSurface );
}

void loadTexturelevel( string path )
{
    SDL_Surface *tSurface = NULL;
    tSurface = IMG_Load( path.c_str() );
    SDL_SetColorKey( tSurface, SDL_TRUE, SDL_MapRGB( tSurface->format, 255, 255, 255));
    menuTextureMenulevel = SDL_CreateTextureFromSurface( menuRenderer, tSurface);
    SDL_FreeSurface( tSurface );
}

void loadTextureMenuhighscore( string path )
{
    SDL_Surface *tSurface = NULL;
    tSurface = IMG_Load( path.c_str() );
    SDL_SetColorKey( tSurface, SDL_TRUE, SDL_MapRGB( tSurface->format, 255, 255, 255));
    menuTexturehighscore = SDL_CreateTextureFromSurface( menuRenderer, tSurface);
    SDL_FreeSurface( tSurface );
}

bool initMenu()
{
    SDL_Init( SDL_INIT_VIDEO );
    SDL_SetHint( SDL_HINT_RENDER_SCALE_QUALITY , "1");
    menuWindow = SDL_CreateWindow( "Menu", SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED, SCREEN_mWIDTH, SCREEN_mHEIGHT, SDL_WINDOW_SHOWN
);
    menuRenderer = SDL_CreateRenderer( menuWindow, -1, SDL_RENDERER_ACCELERATED );
    SDL_SetRenderDrawColor( menuRenderer, 0, 0, 0, 0 );
    IMG_Init( IMG_INIT_PNG );

    return true;
}
bool initcredit()
{
    SDL_Init( SDL_INIT_VIDEO );
    SDL_SetHint( SDL_HINT_RENDER_SCALE_QUALITY , "1");
    creditwindow = SDL_CreateWindow( "Credits", SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED, SCREEN_mWIDTH,SCREEN_mHEIGHT, SDL_WINDOW_SHOWN
);
    creditrenderer = SDL_CreateRenderer( creditwindow, -1, SDL_RENDERER_ACCELERATED );
    SDL_SetRenderDrawColor( creditrenderer, 0, 0, 0, 0 );
    IMG_Init( IMG_INIT_PNG );

    return true;
}
```

```
void loadTexturecredit( string path )
{
   SDL_Surface *tSurface = NULL;
   tSurface = IMG_Load( path.c_str() );
   SDL_SetColorKey( tSurface, SDL_TRUE, SDL_MapRGB( tSurface->format, 255, 255, 255));
   Texturecredit = SDL_CreateTextureFromSurface( creditrenderer, tSurface);
   SDL_FreeSurface( tSurface );
}

bool inithighscore()
{
   SDL_Init( SDL_INIT_VIDEO );
   SDL_SetHint( SDL_HINT_RENDER_SCALE_QUALITY , "1");
   highscorewindow = SDL_CreateWindow( "High Score", SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED, SCREEN_mWIDTH,SCREEN_mHEIGHT, SDL_WINDOW_SHOWN
);
   highscorerenderer = SDL_CreateRenderer( highscorewindow, -1, SDL_RENDERER_ACCELERATED
);
   SDL_SetRenderDrawColor( highscorerenderer, 0, 0, 0, 0 );
   IMG_Init( IMG_INIT_PNG );

   return true;
}

void loadTexturehighscore( string path )
{
   SDL_Surface *tSurface = NULL;
   tSurface = IMG_Load( path.c_str() );
   SDL_SetColorKey( tSurface, SDL_TRUE, SDL_MapRGB( tSurface->format, 255, 255, 255));
   Texturehighscore = SDL_CreateTextureFromSurface( highscorerenderer, tSurface);
   SDL_FreeSurface( tSurface );
}

void sethighscore(int score1, int score2){
        FILE *fp, *fpr;
        fp = fopen("score,txt", "r");
        int hs1,hs2;
        fscanf(fp, "%d %d",&hs1, &hs2);
        fclose(fp);
        fpr = fopen("score.txt", "w");
        if(score1 > hs1 && score2 > hs2){
                hs1 = score1;
                hs2 = score2;
        }
        fprintf(fpr, "BOT %d - %d YOU", hs1,hs2);
        fclose(fpr);
}
```

```
SDL_Color dark_font = {67, 68, 69};
SDL_Color light_font = {187, 191, 194};

bool done = false;
bool QUIT = false;
bool quitMenu = false;
bool quitcredit = false;
bool quithighscore = false;

// Controllers
bool mouse = true;
bool keyboard = false;

// Mouse coordinates;
int mouse_x, mouse_y;

// Paddle lengths
const int PADDLE_WIDTH = 10;
const int PADDLE_HEIGHT = 60;

// Paddle position
int left_paddle_x = 40;
int left_paddle_y = SCREEN_HEIGHT / 2 - 30;

int right_paddle_x = SCREEN_WIDTH - (40+PADDLE_WIDTH);
int right_paddle_y = SCREEN_HEIGHT / 2 - 30;

// Launch ball
bool launch_ball = false;

// Ball dimensions
const int BALL_WIDTH = 10;
const int BALL_HEIGHT = 10;

// Ball position
int x_ball = SCREEN_WIDTH / 2;
int y_ball = SCREEN_HEIGHT / 2;

// Ball movement
int ball_dx = 0;
int ball_dy = 0;

int speed = 8;
int hit_count = 0;
```

```
float angle = 0.0f;

bool bounce = false;

// Match score
int score1 = 0;
int score2 = 0;

bool left_score_changed = true;
bool right_score_changed = true;

// Prediction
int final_predicted_y;

// Font names
string fonts[] = {"Roboto-BlackItalic.ttf", "Roboto-Bold.ttf"};

void renderTexture(SDL_Texture *tex, SDL_Renderer *ren, SDL_Rect dst, SDL_Rect *clip = nullptr) {
    SDL_RenderCopy(ren, tex, clip, &dst);
}

void renderTexture(SDL_Texture *tex, SDL_Renderer *ren, int x, int y, SDL_Rect *clip = nullptr) {
    SDL_Rect dst;
    dst.x = x;
    dst.y = y;
    if (clip != nullptr){
        dst.w = clip->w;
        dst.h = clip->h;
    }
    else
        SDL_QueryTexture(tex, NULL, NULL, &dst.w, &dst.h);

    renderTexture(tex, ren, dst, clip);
}

SDL_Texture* renderText(const string &message, const string &fontFile, SDL_Color color, int fontSize,
SDL_Renderer *renderer) {
    TTF_Font *font = TTF_OpenFont(fontFile.c_str(), fontSize);
    SDL_Surface *surf = TTF_RenderText_Blended(font, message.c_str(), color);
    SDL_Texture *texture = SDL_CreateTextureFromSurface(renderer, surf);
    SDL_FreeSurface(surf);
    TTF_CloseFont(font);
    return texture;
}

// Imprecise prediction of ball position on the y-axis after right paddle collision
int predict() {
    // Find slope
```

```
    float slope = (float)(y_ball - y_ball+ball_dy)/(x_ball - x_ball+ball_dx);
    // Distance between paddles
    int paddle_distance = right_paddle_x - (left_paddle_x+PADDLE_WIDTH);
    // Prediction without taking into consideration upper and bottom wall collisions
    int predicted_y = abs(slope * -(paddle_distance) + y_ball);
    // Calculate number of reflexions
    int number_of_reflexions = predicted_y / SCREEN_HEIGHT;
    // Predictions taking into consideration upper and bottom wall collisions
    if (number_of_reflexions % 2 == 0)                    // Even number of reflexions
        predicted_y = predicted_y % SCREEN_HEIGHT;
    else                                                  // Odd number of reflexsion
        predicted_y = SCREEN_HEIGHT - (predicted_y % SCREEN_HEIGHT);

    return predicted_y;
}

// Get user input
void input() {
    SDL_Event event;    // stores next event to be processed
    // Queuing events
    while(SDL_PollEvent(&event)) {
        // Track mouse movement
        if (event.type == SDL_MOUSEMOTION)
            SDL_GetMouseState(&mouse_x, &mouse_y);

        // Clicking 'x' or pressing F4
        if (event.type == SDL_QUIT){
            done = true;

            //quitcredit = true;
             quitMenu = false;
             QUIT = false;
            }
        // Pressing a key
        if (event.type == SDL_KEYDOWN)
            switch(event.key.keysym.sym) {

                // Pressing ESC exits from the game
                case SDLK_ESCAPE:
                    done = true;
                    quitMenu = false;
                    QUIT = false;
                   // quitMenu = true;
                    break;

                // Pressing space will launch the ball if it isn't already launched
                case SDLK_SPACE:
                    if (!launch_ball) {
```

```
                int direction = 1+(-2)*(rand()%2);                // either 1 or -1
                angle = rand()%120-60;                            // between -60 and 59
                ball_dx = direction*speed*cos(angle*M_PI/180.0f);   // speed on the x-axis
                ball_dy = speed*sin(angle*M_PI/180.0f);             // speed on the y-axis

                // Find slope
                float slope = (float)(y_ball - y_ball+ball_dy)/(x_ball - x_ball+ball_dx);

                // Distance between left paddle and center
                int paddle_distance = SCREEN_WIDTH/2 - (left_paddle_x+PADDLE_WIDTH);

                // Predicting where the left paddle should go in case ball is launched left
                final_predicted_y = abs(slope * -(paddle_distance) + y_ball);

                launch_ball = true;
              }
              break;

            // Pressing F11 to toggle fullscreen
            case SDLK_F11:
              int flags = SDL_GetWindowFlags(window);
              if(flags & SDL_WINDOW_FULLSCREEN)
                SDL_SetWindowFullscreen(window, 0);
              else
                SDL_SetWindowFullscreen(window, SDL_WINDOW_FULLSCREEN);
              break;
          }
    }
}

// Check if collision with left paddle occurs in next frame
bool checkLeftCollision() {
    if (!(x_ball + ball_dx <= left_paddle_x + PADDLE_WIDTH))
        return false;
    if (x_ball < left_paddle_x)
        return false;
    if (!(y_ball + BALL_WIDTH >= left_paddle_y && y_ball <= left_paddle_y + PADDLE_HEIGHT))
        return false;
    return true;
}

// Check if collision with right paddle occurs in next frame
bool checkRightCollision() {
    if (!(x_ball + BALL_WIDTH + ball_dx >= right_paddle_x))
        return false;
    if (x_ball > right_paddle_x + PADDLE_WIDTH)
        return false;
    if (!(y_ball + BALL_WIDTH > right_paddle_y && y_ball <= right_paddle_y + PADDLE_HEIGHT))
```

```
        return false;
    return true;
}

// Update game values
void update() {

    // Right paddle follows the player's mouse movement on the y-axis
    if (mouse == true)
        right_paddle_y = mouse_y;

    /* AI */
    if (x_ball < SCREEN_WIDTH*3/5 && ball_dx < 0) {
        // Follow the ball
        if (left_paddle_y + (PADDLE_HEIGHT - BALL_HEIGHT)/2 < final_predicted_y-2)
            left_paddle_y += speed/8 * 3;
        else if (left_paddle_y + (PADDLE_HEIGHT - BALL_HEIGHT)/2 > final_predicted_y+2)
            left_paddle_y -= speed/8 * 3;
    }

    // Ball is anywhere on the screen but going right
    else if (ball_dx >= 0) {

        // Left paddle slowly moves to the center
        if (left_paddle_y + PADDLE_HEIGHT / 2 < SCREEN_HEIGHT/2)
            left_paddle_y += 2;
        else if (left_paddle_y + PADDLE_HEIGHT / 2 > SCREEN_HEIGHT/2)
            left_paddle_y -= 2;
    }

    if (right_paddle_y + PADDLE_HEIGHT > SCREEN_HEIGHT)
        right_paddle_y = SCREEN_HEIGHT - PADDLE_HEIGHT;

    // Left paddle shouldn't be allowed to go above the screen
    if (left_paddle_y < 0)
        left_paddle_y = 0;

    // Left paddle shouldn't be allowed to below the screen
    else if (left_paddle_y + PADDLE_HEIGHT > SCREEN_HEIGHT)
        left_paddle_y = SCREEN_HEIGHT - PADDLE_HEIGHT;

    // We're done updating values if the ball hasn't been launched yet
    if (!launch_ball)
        return;

    // Three hits => increment ball speed and reset hit counter
    if (hit_count == 3) {
        speed++;
```

```
        hit_count = 0;
    }

    // Smooth collision between ball and left paddle
    if (checkLeftCollision()) {
            if (bounce) {
                // y coordinate of the ball in relation to the left paddle (from 0 to 70)
                int left_relative_y = (y_ball - left_paddle_y + BALL_HEIGHT);

                // Angle formed between ball direction and left paddle after collision
                angle = (2.14f * left_relative_y - 75.0f);

                ball_dx = speed*cos(angle*M_PI/180.0f); // convert angle to radian, find its cos() and multiply by
the speed
                ball_dy = speed*sin(angle*M_PI/180.0f); // convert angle to radina, find its sin() and multiply by
the speed
                bounce = false;                         // finished bouncing
            }
            x_ball = left_paddle_x + PADDLE_WIDTH;      // deposit ball on left paddle surface (smooth
collision)
            bounce = true;                              // bounce ball on next frame

    }
    // Smooth collision between ball and right paddle
    else if (checkRightCollision()) {
            if (bounce) {
                // y coordinate of the ball in relation to the right paddle (from 0 to 70)
                int right_relative_y = (y_ball - right_paddle_y + BALL_HEIGHT);

                // Angle formed between ball direction and right paddle after collision
                angle = (2.14 * right_relative_y - 75.0f);

                ball_dx = -speed*cos(angle*M_PI/180.0f);// convert angle to radian, find its cos() and multiply
by the negative of speed
                ball_dy = speed*sin(angle*M_PI/180.0f); // convert angle to radian, find its sin() and multiply by
the speed
                bounce = false;                         // finished bouncing
            }
            x_ball = right_paddle_x - BALL_WIDTH;       // deposit ball on surface right paddle surface
(smooth collision)
            hit_count++;                               // increment hit counter
            bounce = true;                              // bounce ball on next frame
                  final_predicted_y = predict();              // predict ball position for AI to intercept
    }

    // Upper and bottom walls collision
    else if ( (y_ball + ball_dy < 0) || (y_ball + BALL_HEIGHT + ball_dy >= SCREEN_HEIGHT) ) {
        ball_dy *= -1;                                 // reverse ball direction on y-axis
```

```
    }

    // No collision occurs, update ball coordinates
    else {
        x_ball += ball_dx;
        y_ball += ball_dy;
    }

    // If ball goes out...
    if (x_ball > SCREEN_WIDTH || x_ball < 0) {

        // Change score
        if (x_ball > SCREEN_WIDTH) {
            score1++;
            left_score_changed = true;
        } else {
            score2++;
            right_score_changed = true;
        }

        // Reset ball position as before launch
        x_ball = SCREEN_WIDTH / 2;
        y_ball = SCREEN_HEIGHT / 2;

        // Ball is fixed
        ball_dx = 0;
        ball_dy = 0;
        launch_ball = false;

        // Speed and hit counter are reset to their initial positions
        speed = 8;
        hit_count = 0;
    }
}

// Render objects on screen
void render() {

    // Clear screen (background color)
    SDL_SetRenderDrawColor( renderer, 67, 68, 69, 255 );        // dark grey
    SDL_RenderClear(renderer);

    // Color left background with light grey
    SDL_Rect left_background = { SCREEN_WIDTH / 2, 0, SCREEN_WIDTH / 2, SCREEN_HEIGHT };
    SDL_SetRenderDrawColor( renderer, 187, 191, 194, 255 );
    SDL_RenderFillRect( renderer, &left_background );

    // Paddle color
```

```
    SDL_SetRenderDrawColor( renderer, 212, 120, 102, 255 );

    // Render filled paddle
    SDL_Rect paddle1 = { left_paddle_x, left_paddle_y, PADDLE_WIDTH, PADDLE_HEIGHT };
    SDL_RenderFillRect( renderer, &paddle1 );

    // Render filled paddle
    SDL_Rect paddle2 = { right_paddle_x, right_paddle_y, PADDLE_WIDTH, PADDLE_HEIGHT };
    SDL_RenderFillRect( renderer, &paddle2 );

    // Render ball
    SDL_Rect ball = { x_ball - BALL_WIDTH / 2, y_ball, BALL_WIDTH, BALL_HEIGHT };
    SDL_RenderFillRect(renderer, &ball);

    // Render scores
    if (left_score_changed) {
        font_image_score1 = renderText(to_string(score1), "Roboto-BlackItalic.ttf", light_font, 24, renderer);
        left_score_changed = false;
    }
    renderTexture(font_image_score1, renderer, SCREEN_WIDTH * 4 / 10, SCREEN_HEIGHT / 12);

    int score_font_size = 24;
    if (right_score_changed) {
        font_image_score2 = renderText(to_string(score2), "Roboto-BlackItalic.ttf", dark_font,
score_font_size, renderer);
        right_score_changed = false;
    }

    renderTexture(font_image_score2, renderer, SCREEN_WIDTH * 6 / 10 - score_font_size/2,
SCREEN_HEIGHT/ 12);

    // Render text indicating the winner
    if (score1 == 5) {
        font_image_winner = renderText("BOT Won!", fonts[0], light_font, 24, renderer);
        renderTexture(font_image_winner, renderer, SCREEN_WIDTH * 1 / 10 + 3, SCREEN_HEIGHT / 4);
// align with score
        font_image_restart = renderText("Press SPACE to restart", fonts[0], light_font, 18, renderer);
        renderTexture(font_image_restart, renderer, SCREEN_WIDTH * 1 / 10 + 3, SCREEN_HEIGHT / 3);
        if (launch_ball) {
            score1 = 0;
            score2 = 0;
            left_score_changed = true;
            right_score_changed = true;
        }
    } else if (score2 == 5) {
        font_image_winner = renderText(" You Won!", fonts[0], dark_font, 24, renderer);
        renderTexture(font_image_winner, renderer, SCREEN_WIDTH * 6 / 10 - score_font_size/2,
SCREEN_HEIGHT / 4);   // align with score
```

```
        font_image_restart = renderText("Press SPACE to restart", fonts[0], dark_font, 18, renderer);
        renderTexture(font_image_restart, renderer, SCREEN_WIDTH * 6 / 10 - score_font_size/2,
SCREEN_HEIGHT / 3);
        if (launch_ball) {
            score1 = 0;
            score2 = 0;
            left_score_changed = true;
            right_score_changed = true;
        }
    }

    // Draw "Press SPACE to start"
    else if (!launch_ball) {
        renderTexture(font_image_launch1, renderer, SCREEN_WIDTH / 2 - 80, SCREEN_HEIGHT - 25);
        renderTexture(font_image_launch2, renderer, SCREEN_WIDTH / 2 + 1, SCREEN_HEIGHT - 25);
    }

    // Swap buffers
    SDL_RenderPresent(renderer);
}

void cleanUp() {

    // Destroy textures
    SDL_DestroyTexture(font_image_score1);
    SDL_DestroyTexture(font_image_score2);

    // Destroy renderer and window
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);

    // Shuts down SDL
    SDL_Quit();
}

void gameLoop(){

            SDL_Event e;


    while( !QUIT ){
        initMenu();

        loadTextureMenuEnter( "image/new game.png");
        loadTexturelevel("image/credit.png");
        loadTextureMenuhighscore( "image/high score.png");
        loadTextureMenuExit( "image/quit.png");
```

```
while ( !quitMenu )
{
   while( SDL_PollEvent( &e) != 0)
   {
      if(e.type == SDL_QUIT )
      {
         quitMenu = true;
         QUIT = true;

      }

      if( e.type == SDL_MOUSEBUTTONUP )
      {
         int x, y;
         SDL_GetMouseState( &x, &y);
         if( x >= 480 && x <= 860 && y >= 100 && y <= 180)
         {
            if( e.type == SDL_MOUSEBUTTONUP)
            {
               quitMenu = true;
               done = false;
               SDL_DestroyRenderer( menuRenderer );
               SDL_DestroyWindow( menuWindow );
            }
         }


          if( x >= 480 && x <= 860 && y >= 200 && y <= 280)
         {
            if( e.type == SDL_MOUSEBUTTONUP)
            {
              inithighscore();
              loadTexturehighscore("image/sadamata.png");
               quitMenu = true;
               done = true;

               quithighscore = false;

              SDL_DestroyRenderer( menuRenderer );
               SDL_DestroyWindow( menuWindow );
            }
         }

         if( x >= 480 && x <= 860 && y >= 300 && y <= 380)
         {
            if( e.type == SDL_MOUSEBUTTONUP)
            {
              initcredit();
```

```
                loadTexturecredit("image/crej.png");
                  quitMenu = true;
                  done = true;

                  quitcredit = false;

                 SDL_DestroyRenderer( menuRenderer );
                  SDL_DestroyWindow( menuWindow );
               }
            }

            if( x >= 480 && x <= 860 && y >= 400 && y <= 480)
            {
               if( e.type == SDL_MOUSEBUTTONUP)
               {
                  quitMenu = true;
                  done = true;
                  QUIT = true;
                  SDL_DestroyRenderer( menuRenderer );
                  SDL_DestroyWindow( menuWindow );
               }
            }
         }
      }

      SDL_Rect mRect;

         mRect.x = 0;
         mRect.y = 0;
         mRect.w = 1280;
         mRect.h = 720;

         SDL_Rect renderQuad = { 480, 100, 300, 80 };
         SDL_Rect renderQuad1 = { 480, 200, 300, 80};
         SDL_Rect renderQuad2 = { 480, 300, 300, 80};
         SDL_Rect renderQuad3 = { 480, 400, 300, 80};
         SDL_RenderClear( menuRenderer );
         SDL_RenderCopy( menuRenderer, menuTextureEnter, &mRect, &renderQuad );
          SDL_RenderCopy( menuRenderer, menuTexturehighscore, &mRect, &renderQuad1 );
         SDL_RenderCopy( menuRenderer, menuTextureMenulevel, &mRect, &renderQuad2 );
         SDL_RenderCopy( menuRenderer, menuTextureExit, &mRect, &renderQuad3);
         SDL_RenderPresent( menuRenderer );
         SDL_Delay(2);
   }

while(!done){
   input();
   update();
```

```
   render();
}

while(!quithighscore){
        while( SDL_PollEvent( &e) != 0)
       {
          if(e.type == SDL_QUIT )
          {
                  quithighscore = true;
             quitMenu = false;
            // quitEnd = false;
             QUIT = true;
             SDL_DestroyRenderer( highscorerenderer );
             SDL_DestroyWindow( highscorewindow );
          }

         if (e.type == SDL_KEYDOWN){
       switch(e.key.keysym.sym) {

          // Pressing ESC exits from the game
          case SDLK_ESCAPE:
            quithighscore = true;
             quitMenu = false;
             QUIT = false;
            // quitMenu = true;
             break;
          default:
                  break;

                  }

                  }
       }

       SDL_Rect lRect;

          lRect.x = 0;
          lRect.y = 0;
          lRect.w = 1280;
          lRect.h = 720;

          SDL_Rect renderQuad8 = { 280, 100 , 780, 500 };
          SDL_RenderClear( menuRenderer );
          SDL_RenderCopy( highscorerenderer, Texturehighscore, &lRect, &renderQuad8 );
          SDL_RenderPresent( highscorerenderer );
          SDL_Delay(2);
}
```

```
   while(!quitcredit){
         while( SDL_PollEvent( &e) != 0)
        {
           if(e.type == SDL_QUIT )
           {
                   quitcredit = true;
              quitMenu = false;
             // quitEnd = false;
              QUIT = true;
              SDL_DestroyRenderer( creditrenderer );
              SDL_DestroyWindow( creditwindow );
           }

          if (e.type == SDL_KEYDOWN){
        switch(e.key.keysym.sym) {

           // Pressing ESC exits from the game
           case SDLK_ESCAPE:
             quitcredit = true;
              quitMenu = false;
              QUIT = false;
             // quitMenu = true;
              break;
           default:
                   break;

                   }

                   }
        }

        SDL_Rect nRect;

           nRect.x = 0;
           nRect.y = 0;
           nRect.w = 1280;
           nRect.h = 720;

           SDL_Rect renderQuad7 = { 280, 100 , 780, 500 };
           SDL_RenderClear( menuRenderer );
           SDL_RenderCopy( creditrenderer, Texturecredit, &nRect, &renderQuad7 );
           SDL_RenderPresent( creditrenderer );
           SDL_Delay(2);
   }
   cleanUp();
  }
}
```

```
void initialize() {

   // Initialize SDL
   SDL_Init(SDL_INIT_EVERYTHING);

   // Create window in the middle of the screen
   window = SDL_CreateWindow( "Pong Classic",
         SDL_WINDOWPOS_UNDEFINED,
         SDL_WINDOWPOS_UNDEFINED,
         SCREEN_WIDTH,
         SCREEN_HEIGHT,
         SDL_WINDOW_SHOWN);

   // Create renderer in order to draw on window
   renderer = SDL_CreateRenderer( window, -1, SDL_RENDERER_ACCELERATED |
SDL_RENDERER_PRESENTVSYNC );

   // Initialize font
   TTF_Init();

   // Holds text "Press SPACE to start"
   font_image_launch1 = renderText("Press SPA", fonts[0], light_font, 18, renderer);
   font_image_launch2 = renderText("CE to start", fonts[0], dark_font, 18, renderer);
}


int main(int argc, char *argv[ ]) {

   srand(time(NULL));
   initialize();
   gameLoop();
   return 0;

}
```

Special thanks :

1. Mr. Abu Ahmed Ferdaus
   Associate Professor, CSEDU

2. Mr. Hasnain Heickal
   Assistant Professor,CSEDU

3. Mr. Md Mahmudur Rahman
   Lecturer,CSEDU

4. Sakib Hassan
   CSEDU-25th

References :

1. http://lazyfoo.net/tutorials/SDL
2. https://stackoverflow.com
3. https://www.youtube.com
4. https://www.programmingsimplified.com
5. https://www.fontsquirrel.com
6. C The Complete Reference (4th Edition) By Herbert Schildt