Sarawak Campus

Assignment Cover Sheet

(for individual and group assignments)

This cover sheet is to be attached to all assignments, both hard copy and electronic format

ASSIGNMENT DETAILS

Unit Code	COS10009	Unit Title	INTRODUCTION TO PROGRAMMING
Tutorial/Lab Group	Group 6 wed 2-5pm	Lecturer/Tutor Name	
Assignment Title	Custom Program Report		
Due date	05/26/2024	Date Received	

DECLARATION

For both individual and group assignments, in the case of assignment submission on behalf of another student, it is assumed that permission has been given. The University takes no responsibility for any loss, damage, theft, or alteration of the assignment.

To be completed if this is an individual assignment

I declare that this assignment is my individual work. I have not worked collaboratively, nor have I copied from any other student's work or from any other source/s, except where due acknowledgment is made explicitly in the text, nor has any part been written for me by another person.

Student Details	Student ID Number	Student Name	Student Signature
Student 1	104388564	Adil Rumy	

To be completed if this is a group assignment

We declare that this is a group assignment and that no part of this submission has been copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part been written for us by another person.

Student Details	Student ID Number(s)	Student Name(s)	Student Signature (s)
Student 1			
Student 2			
Student 3			
Student 4			
Student 5			

MARKER'S COMMENTS

Total Mark		Marker's Signature		Date	

EXTENSION CERTIFICATE

This assignment has been given an extension by

Unit Convenor			
Extended due date		Date Received	

COS10009 INTRODUCTION TO PROGRAMMING

Custom Program

Name	**ID**
Adil Rumy	104388564

Index of contents

Introduction

My program is a football team creator game, inspired by the famous draft mode in EA Sports's famous football game series FIFA (currently known as FC). In this game, a random set of players are presented, and the user must choose the right player to play for their team based on their rating and if they will fit well with the other players chosen. This game is a Co-op game, meaning it is meant to be played with a friend, where each of you will make a team and face each other. The team with the highest score wins.

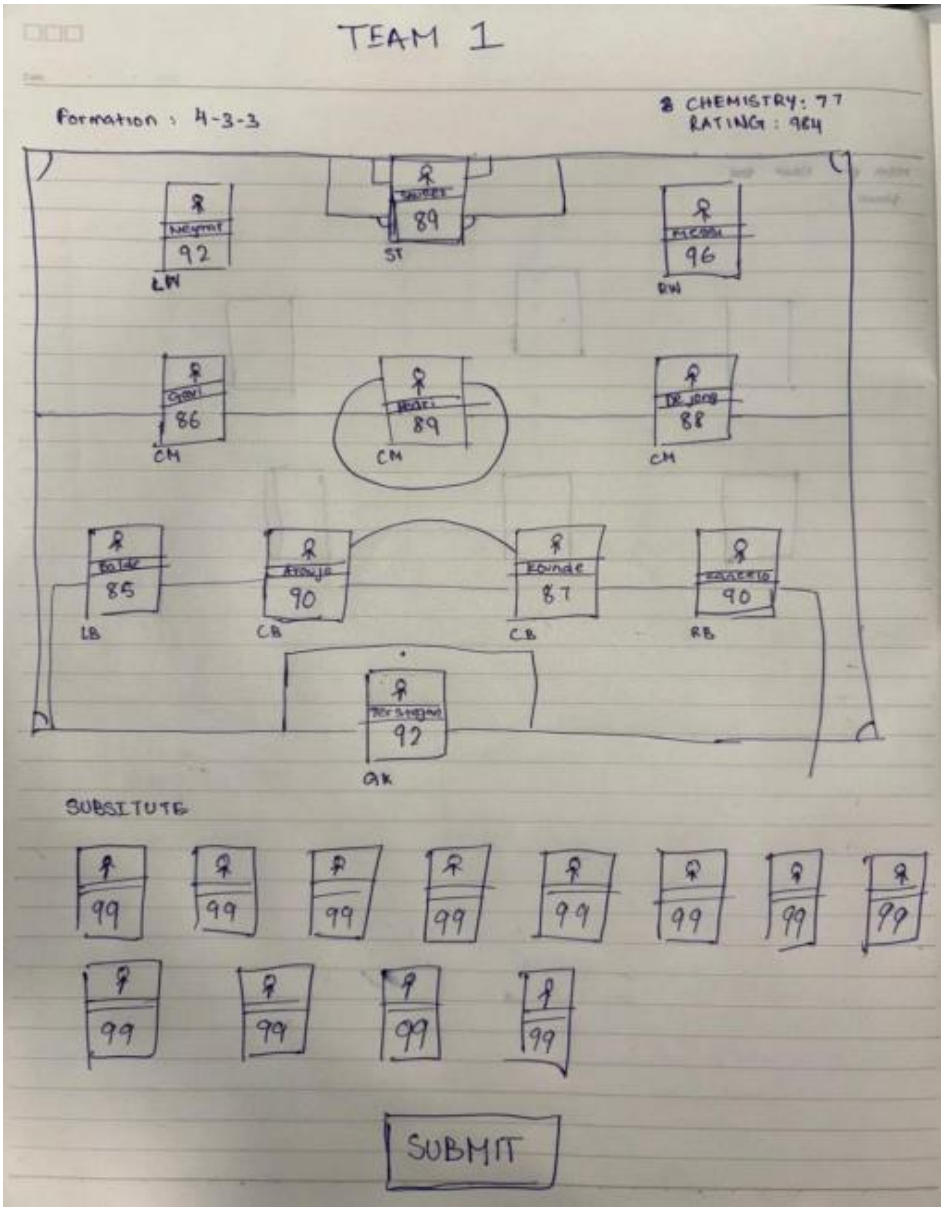


Figure 1. Initial design of the game.

figure 2 and 3. final design of the game.

There were quite a few changes from the initial to final designs. The user can select their player for each position, on the left-hand side. In the right, a reference of the football positions for the beginners is laid out with the team's overall score. Once the team is fully selected, the submit button is presented.

Libraries and defined macros:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <time.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_mixer.h>

#define WINDOW_WIDTH 1500
#define WINDOW_HEIGHT 800
#define M_PI 3.14159265358979323846
```

Figure 4. libraries included and macros defined in the program.

Some Data Types

Player Struct:

Field Name	Data type	Description	Example Value
name	STRING	Stores the name of the player	“Manuel Neuer”
position	STRING	Stores the position of the player	“GK”
country	STRING	Stores the country of the player	“GERMANY”
club	STRING	Stores the club of the player	“BAY”
rating	INTEGER	Stores the rating of the player	90

Other variables:

Field Name	Data type	Description	Example Value
selected	Array of INTEGER	array storing the index of the chosen player in the display array variable.	1
completed	Array of BOOLEAN	Array storing the Boolean value of if the previous player was selected or not.	true
start	BOOLEAN	Variable that indicates if the game should start.	false
numofplayers	INTEGER	Variable that stores the total number of players in the text file.	100
players	Array of type Player	Array storing every player in the text file.	“Bernardo Silva” “RW” “Portugal” “MCI” 88

Functions

Function /Procedure/Method	Description
drawField()	function to draw the football field Using SDL
readPlayers()	function to read players' details from file
total_rating()	function to calculate the teams overall rating
total_chemisty()	function to calculate the team's overall chemistry
load_score()	function to draw the score on the screen
load_text()	function to draw the normal texts
load_volume()	function to draw the volume on the esc menu
options()	Player type function to get 3 random options for the user to select
display_error()	function for if there is an error in displaying the players
create_options()	function to create the option card
display_options()	function to display the card options to the user
free_and_destroy()	function to free the display array, filename array, and to destroy the cards textures
select()	function that allows the user to select the card they want
selected_card()	function to move the selected card to its new position
main()	main function

1. drawField()

```
//function to draw the football field
void drawField(SDL_Renderer *renderer) {
    //draw green backgroud
    SDL_SetRenderDrawColor(renderer, 5, 114, 5, 255);
    //clear the renderer with the draw color
    SDL_RenderClear(renderer);

    //drawing white lines inside the box
    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255); //to draw the white circle in the centre
        int centerX = 750; //centre of the circle in X
        int centerY = 400; //centre of the circle in Y
        int radius = 90; //radius of the circle
        for (int angle = 0; angle <= 360; angle += 1) {
            double x = centerX + radius * cos(angle * M_PI / 180); //to find the X of a certain point in the circle
            double y = centerY + radius * sin(angle * M_PI / 180); //to find the Y of a certain point in the circle
            SDL_RenderDrawPoint(renderer, (int)x, (int)y); //draws a pixel in the calculated X and Y, making it a circle after the loop
        }

    //drawing the top and bottom smaller rectangle inside the outerline
    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
    SDL_Rect rect1 = {600, 50, 300, 125};
    SDL_RenderDrawRect(renderer, &rect1);

    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
    SDL_Rect rect2 = {600, 625, 300, 125};
    SDL_RenderDrawRect(renderer, &rect2);

    //draws the Y center line that passes through the circle
    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
    SDL_RenderDrawLine(renderer, 250, 400, 1249, 400);

    //draw the outerline
    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
    SDL_Rect rec3 = {250, 50, 1000, 700};
    SDL_RenderDrawRect(renderer, &rec3);
}
```

Figure 5. drawField()'s code

In this function we draw the football field seen on the main playing screen. The main background color is set to green whereas the lines in the field are going to be white. The first thing drawn is the center circle, where we declare the middle X coordinates, Y coordinates and radius. Once they have been declared we start a for loop that will run until the angle reaches 360. In the loop we calculate a point of the circle's coordinates using the angle, its respected X or Y coordinate and M_PI to convert it into radians. The function includes three rectangle drawings which are the top inside rectangle, bottom inside rectangle and the outer lines. Finally, the function also includes a line drawing at the center of Y that passes through the circle.

(refer to figure 2 or 3 to see the outputted drawing)

2. readPlayer()

```c
//function to read players' details from file
int readPlayers(Player* players, int max_players) {

    FILE* file = fopen("Players.txt", "r"); //opens file and check if it was successfully opened
    if (file == NULL) {
        printf("Error opening file.\n");
        return 0;
    }

    int numofplayers = 0;
    // we do fscanf(....)!=EOF to make sure that it reads all lines untill the end of the file
    while (fscanf(file, " %59[^\n] %9[^\n] %19[^\n] %9[^\n] %d", players[numofplayers].name, players[numofplayers].position,
    players[numofplayers].country, players[numofplayers].club, &players[numofplayers].rating) != EOF){
        /* in the above line. We use %59[^\n] %9[^\n] %19[^\n] %9[^\n] to limit the amount of characters read,
           so that if the string is too long it doesnt lead to buffer overflow*/

        numofplayers++; //increments numofplayers
        if (numofplayers >= max_players) { //once numofplayers exceeds the max players available, it leaves the loop
            printf("Maximum number of players reached.\n");
            break;
        }
    }

    fclose(file); //closes file
    return numofplayers; //returns the number of players
}
```

Figure 6. readPlayers()'s code

In this function it first opens the file Players.txt to read. It checks if the file was successfully opened. If so, it starts scanning the lines from the text file into players array. The first line is stored into the arrays respected position's name, the second line is stored into the position, third is the country, fourth is the club and finally the fifth is the rating. The number of characters read is limited due to not wanting any buffer overflows. The scanning of the players is a condition inside the while loop, and while it did not reach the end of the file it will continue, if it read the lines successfully it will increment numofplayers and if numofplayers exceeds the maximum number of players allowed it leaves the loop. At the end of the function, it closes the file and returns the number of players

3. total_rating()

```c
//Function to calculate the teams overal rating
int total_rating(Player* options_array, int selected, int team_rating){
    if (selected == -1){ //incase if the user clicks outside the wanted region
        team_rating+=0;
    }else{
        team_rating+=options_array[selected].rating; //increments the selected players rating into the total team's rating
    }

    return team_rating;
}
```

Figure 7. total_rating() 's code

In this function we calculate the total rating of the team by adding the user's selected player's rating into the overall rating and then returns it. For a safety measure the if selected is equal to –1 is added to not hinder the teams overall rating.

4. total_ chemistry()

```c
//Function to calculate the teams overal chemistry
int total_chemisty(Player* team){
    int chem_position[11];
    int final_chem=0;

    //for loop is used to iterate through all 11 players in the team, and add up the similarities
    for (int i = 0; i < 11; i++){
        int player_chem = 0;
        for(int x = 0; x < 11; x++){
            if (i != x) {
                if (strcasecmp(team[i].club, team[x].club) == 0){
                    //if the player shares the same club it increments the chemisty of the player
                    player_chem++;
                }

                if(strcasecmp(team[i].country, team[x].country) == 0){
                    //if the player shares the same country it increments the chemisty of the player
                    player_chem++;
                }
            }
        }

        chem_position[i] = player_chem;
        final_chem += player_chem; //adds the players chemistry into the teams overall chemisty
    }

    return final_chem;//returns the teams chemistry
}
```

Figure 8. total_chemisty() 's code

In this function it loops through all the 11 user selected players and compares if any player shares the same country or club. If for example we are looking into the first player, it will compare the club and country of the first player to every other player in the team, if the first player shares the same club or country, the players chemistry will increment. Finally, the players' chemistry will be added into the team's overall chemistry and will be returned.

5. load_score()

```c
//function to draw the score on the screen
void load_score(SDL_Renderer *renderer, TTF_Font *font, SDL_Color color, int score){
    char text[50];
    int texW=0; //text's width
    int texH=0; //text's height
    snprintf(text, sizeof(text),"%d %%", score); //stores the score into text

    //creating the surface and textures for the text
    SDL_Surface *TextScore_sur= TTF_RenderText_Solid(font,text,color);
    SDL_Texture *TextScore_tex=SDL_CreateTextureFromSurface(renderer,TextScore_sur);

    SDL_QueryTexture(TextScore_tex,NULL,NULL,&texW,&texH); //queries the the texts infromation
    SDL_Rect ScoreRect={1330, 100, texW+30, texH+30}; //position and size of the text
    SDL_RenderCopy(renderer,TextScore_tex,NULL,&ScoreRect); //renders the text into the screen

    //destory and free the texture and surface
    SDL_DestroyTexture(TextScore_tex);
    SDL_FreeSurface(TextScore_sur);
}
```

Figure 9. load_score() 's code

This function helps draw the team's score, it first stores the score with the “%” into a variable called text. From this variable the surface is made and with the surface the texture is made. Then puts the text in a rectangle and specifies the position and size of the text in the screen. Finally, it renders the text, destroys the texture, and frees the surface.

6. load_text()

```c
//function to draw the normal texts
void load_text(SDL_Renderer *renderer, TTF_Font *font, SDL_Color color, char text[], int X, int Y, int addW, int addH){
    int texW=0;
    int texH=0;

    //creating the surface and textures for the text
    SDL_Surface *TextScore_sur= TTF_RenderText_Solid(font,text,color);
    SDL_Texture *TextScore_tex=SDL_CreateTextureFromSurface(renderer,TextScore_sur);

    SDL_QueryTexture(TextScore_tex,NULL,NULL,&texW,&texH); //queries the the texts infromation
    SDL_Rect ScoreRect={X, Y, texW+addW, texH+addH}; //position and size of the text
    SDL_RenderCopy(renderer,TextScore_tex,NULL,&ScoreRect); //renders the text into the screen

    //destory and free the texture and surface
    SDL_DestroyTexture(TextScore_tex);
    SDL_FreeSurface(TextScore_sur);
}
```

Figure 10. load_text() 's code

This function helps draw any normal text wanted, it is like load_score(), however does not include a new variable to store the text to render. Instead, it takes the text straight as a parameter with the addition of the desired text’s position and size, and then renders it out the same way as load_score().

7. load_volume()

```
//function to draw the volume on the esc menu
void load_volume(SDL_Renderer *renderer, TTF_Font *font, SDL_Color color, int volume){
    int output_volume=volume * 0.78; //to make the shown volume to not exceed 100%
    char text1[50];
    int texW=0;
    int texH=0;

    snprintf(text1, sizeof(text1),"Current volume:%d %%", output_volume); //stores the volume into text1

    //creating the surface and textures for the text
    SDL_Surface *TextScore_sur1= TTF_RenderText_Solid(font,text1,color);
    SDL_Texture *TextScore_tex1=SDL_CreateTextureFromSurface(renderer,TextScore_sur1);

    SDL_QueryTexture(TextScore_tex1,NULL,NULL,&texW,&texH); //queries the the texts infromation
    SDL_Rect ScoreRect={600, 200, texW+35, texH+30}; //position and size of the text
    SDL_RenderCopy(renderer,TextScore_tex1,NULL,&ScoreRect); //renders the text into the screen

    //destory and free the texture and surface
    SDL_DestroyTexture(TextScore_tex1);
    SDL_FreeSurface(TextScore_sur1);
}
```

Figure 11. load_volume() 's code

This function helps to draw the volume located in the ESC menu, this function is more like load_score() compared to load_text(). It first makes the volume into a more presentable value that does not exceed 100, and then copies the output volume with "Current volume: %" into the variable text1. From then onwards it creates its textures and surfaces and renders like load_score().

8. options()

```c
// Function to get 3 random options for the user to select
Player* options(Player* players, int numofplayers, Player* display, char* PosToFind) {
    Player PlayersInthePosition[15];
    // Allocate memory for three players
    display = malloc(3 * sizeof(Player));
    if (display == NULL) { //checks if the memory was successfully allocated
        printf("Memory allocation failed.\n");
        return NULL;
    }

    //store all the players found for the given position in an array
    int Index = 0;
    /*if the player in the main Player array shared the same desired position,
      it stores that player into an array dedicated to only those players in that position*/
    for (int i = 0; i < numofplayers; i++) {
        if (strcmp(players[i].position, PosToFind) == 0) {
            PlayersInthePosition[Index++] = players[i];
            /*once the player has been added into the new array,
              the index increments for the next player*/
        }
    }

    //shuffle the array of players
    for (int i = Index - 1; i > 0; i--) {
        int j = rand() % (i + 1);
        Player temp = PlayersInthePosition[i];
        PlayersInthePosition[i] = PlayersInthePosition[j];
        PlayersInthePosition[j] = temp;
    }

    //copy the first three elements to the display array
    for (int x = 0; x < 3 && x < Index; x++) {
        display[x] = PlayersInthePosition[x];
    }

    return display;
}
```

Figure 12. options()'s code

In this function it gets the 3 random players needed for the user to select. It takes the main player array, with the number of players, the array of the 3 random players, and the position to find the players. It first allocates memory for three Player structures and assigns the address of this memory block to the pointer display. It also checks if the memory allocation was successful. Next is declares a variable index which will keep track of the number of players found in the position. If the player in the main Player array shares the same desired position, it stores that player into an array dedicated to only those players in that position, and then index is incremented. After this the function shuffles the array which contains all the players with the same position. It shuffles the array randomly and then stores the first 3 players in the array display and returns it.

9. display_error()

```c
//fucntion for if there is an error in displaying the players
int display_error(Player* display,SDL_Renderer *renderer,SDL_Window *window){
    if (display == NULL) {
        printf("display is empty");
        free(display);
        SDL_DestroyRenderer(renderer);
        SDL_DestroyWindow(window);
        IMG_Quit();
        SDL_Quit();
        return 1;
    }
}
```

Figure 13. display_error ()'s code

This function checks if the display array is empty or not. If it is empty, it will print a message, destroy the renderer and window, free the array and quit SDL.

10. create_options()

```c
//function to create the option card
int create_options(char **filename,Player* display,SDL_Renderer *renderer,SDL_Window *window,SDL_Texture *cardTexture[3]){
    SDL_Surface *cardsurface[3];
    for (int j = 0; j < 3; j++) {
        filename[j] = malloc(100 * sizeof(char)); //allocates 100 characters into filename
        if (filename[j] == NULL) {
            printf("Memory allocation failed.\n");
            for (int k = 0; k < j; k++) {
                free(filename[k]);  //frees the memory
            }
            SDL_DestroyRenderer(renderer);
            SDL_DestroyWindow(window);
            SDL_Quit();
            return 1;
        }
    }

    for (int j=0;j<3;j++){
        sprintf(filename[j], "images/%s.png", display[j].name); //stores "images/ .png" with the display name before the .png
        cardsurface[j] = IMG_Load(filename[j]); //loads the image into the surface
        if (!cardsurface[j]) {
            printf("IMG_Load Error: %s\n", IMG_GetError());
            SDL_DestroyRenderer(renderer);
            SDL_DestroyWindow(window);
            SDL_Quit();
            return 1;
        }

        cardTexture[j] = SDL_CreateTextureFromSurface(renderer, cardsurface[j]); //creates a texture with the surface
        SDL_FreeSurface(cardsurface[j]); //free the surface
        if (!cardTexture[j]) {
            printf("SDL_CreateTextureFromSurface Error: %s\n", SDL_GetError());
            SDL_DestroyRenderer(renderer);
            SDL_DestroyWindow(window);
            SDL_Quit();
            return 1;
        }
    }
}
```

Figure 14. create_options()'s code

This function is used to create the images of the options. It first allocates memory to the filename array and then stores "images/ .png" with the display's name before the .png. It then loads the image and creates a texture out of it. It also checks if there are any errors.

11. display_options()

```
//function to display the card options to the user
void display_options(SDL_Renderer *renderer,SDL_Texture *cardTexture[3]){
    int X=50;
    int Y=100;
    int W=150;
    int H=200;

    SDL_Rect cardRect1 = {X, Y, W, H};
    SDL_RenderCopy(renderer, cardTexture[0], NULL, &cardRect1); //render the first option

    Y+=225;
    SDL_Rect cardRect2 = {X, Y, W, H};
    SDL_RenderCopy(renderer, cardTexture[1], NULL, &cardRect2); //render the second option

    Y+=225;
    SDL_Rect cardRect3 = {X, Y, W, H};
    SDL_RenderCopy(renderer, cardTexture[2], NULL, &cardRect3); //render the third option
}
```

Figure 15. display _options()'s code

This function renders the options in its position. It has the same X coordinate, width, and height. But the Y coordinate changes by an addition of 225.

12. free_and_destroy()

```
//function to free the display array, filename array, and to destroy the cards textures
void free_and_destroy(Player* display,char **filename,SDL_Texture *cardTexture[3]){
    free(display); //free the display array
    for (int j = 0; j < 3; j++) {
        free(filename[j]); //freeing the filenames
    }
    for (int x=0;x<3;x++){
        SDL_DestroyTexture(cardTexture[x]); //destroying the textures
    }
}
```

Figure 15. free_and_destory()'s code

This function frees the display and filename arrays, furthermore it destroys all the textures.

13. select()

```
//function that allows the user to select the card they want
int select(int mouseX, int mouseY, char **filenames, SDL_Texture **textures, bool *complete,Mix_Chunk* SelectSound){
    int selected=-1; //declares the variable selected and asigns -1 to it

    if (mouseX > 50 && mouseX < 200 && mouseY > 100 && mouseY < 300) //the first card's borders
    {
        Mix_PlayChannel(-1, SelectSound, 0); //plays the select sound effect
        selected = 0; //if first card is selected, it asigns 0 to the variable selected
        *complete = true; //asigns true to complete to indicate the card has been selected
    }
    else if (mouseX > 50 && mouseX < 200 && mouseY > 325 && mouseY < 525) //the second card's borders
    {
        Mix_PlayChannel(-1, SelectSound, 0);
        selected = 1; //if second card is selected, it asigns 1 to the variable selected
        *complete = true;
    }
    else if (mouseX > 50 && mouseX < 200 && mouseY > 550 && mouseY < 750) //the third card's borders
    {
        Mix_PlayChannel(-1, SelectSound, 0);
        selected = 2; //if third card is selected, it asigns 2 to the variable selected
        *complete = true;
    }

    //when one of the card is selected, the rest of the card's textures will be destroyed
    if (selected==0){
        SDL_DestroyTexture(textures[1]);
        SDL_DestroyTexture(textures[2]);
    }else if (selected==1){
        SDL_DestroyTexture(textures[0]);
        SDL_DestroyTexture(textures[2]);
    }else if (selected==2){
        SDL_DestroyTexture(textures[0]);
        SDL_DestroyTexture(textures[1]);
    }

    return selected; //selected value is returned
}
```

Figure 16. select()'s code

With the help of this function the user can select the card they want, it first declares and assigns –1 to the variable called selected. Then it checks where the mouse was clicked, the first condition in the if statement is for the first card, the second condition is for the second card and the third condition is for the third card. Inside each of the if conditions it plays the select sound effect and makes the complete variable which is a parameter true, the only difference between each condition is that the selected is assigned a different value. 0, 1 and 2 are assigned to selected if the first, second or third card is chosen, respectively. Finally, the cards which were not selected are destroyed and the selected variable is returned.

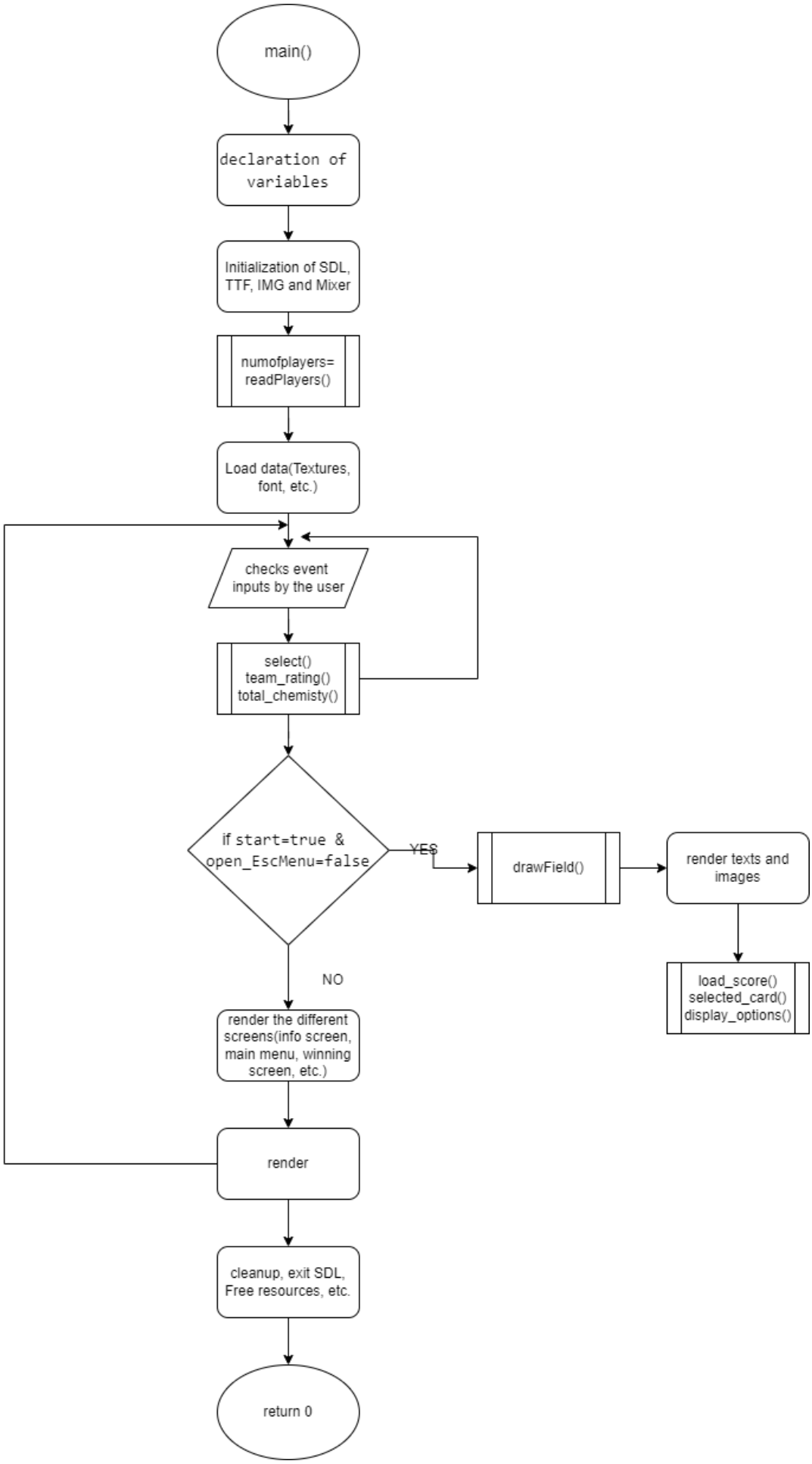
14. selected_card()

```
//function to move the selected card to its new position
void selected_card(int newX,int newY,int selected,SDL_Texture **textures,SDL_Renderer *renderer){
    SDL_Rect selected_cardRect = {newX, newY, 100, 150}; // New position for the selected card
    SDL_RenderCopy(renderer, textures[selected], NULL, &selected_cardRect); //renders the card onto the screen
}
```

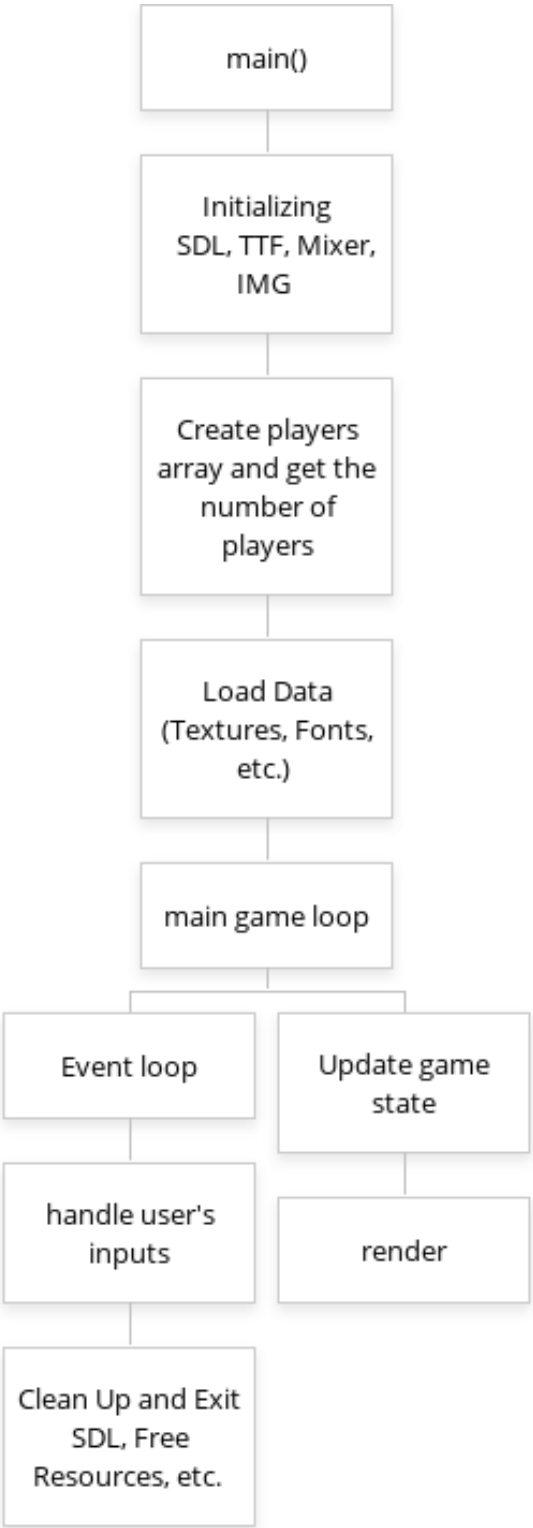
Figure 17. selected_card()'s code

In this function it moves and renders the selected card into its new position. It takes the new coordinates, selected card, the card's texture, and the renderer as parameters.

main()


main()
declaration of variables
Initialization of SDL, TTF, IMG and Mixer
numofplayers= readPlayers()
Load data(Textures, font, etc.)
checks event inputs by the user
select() team_rating() total_chemisty()
if start=true & open_EscMenu=false
YES
drawField()
render texts and images
load_score() selected_card() display_options()
NO
render the different screens(info screen, main menu, winning screen, etc.)
render
cleanup, exit SDL, Free resources, etc.
return 0

structured diagram



```c
//main function
int main(int argc, char *argv[]) {
    bool completed[11] = {false}; //array of boolean values to see if a player is selected
    bool start=false;  //boolean value to inidicate the game started
    bool leave_MainMenu=false; //boolean value to indicate to leave the main menu
    bool open_EscMenu=false; //boolean value inidicating that the esc menu is open
    bool submit=false; //boolean value to inidicate the submit button has been pressed
    bool final_score=false; //boolean value to indicate that the final score should be shown for the first team
    bool final_score_2=false; //boolean value to indicate that the final score should be shown for the second team
    bool team2_ongoing=false; //boolean value to indicate the second part of the game is ongoing
    bool winning_screen=false; //boolean value to inidicate the winning screen is open
    bool leave_info=false; //boolean value to indiciate to leave the info screen
    Player TeamOne[11]; //array of all the chosen players for the first team
    Player TeamTwo[11]; //array of all the chosen players for the second team
    int final_chem=0; //final team chemistry for the first team
    int final_chem_2=0; //final team chemistry for the second team
    int selected[11]; //array of the index of the selected player
    int team_rating=0; //final team rating for the first team
    int team_rating_2=0; //final team rating for the second team
    int volume = MIX_MAX_VOLUME / 2; //music volume
    int VolSliderAdd=100; //initial volume slider position
    int score=0; //displayed score in the playing screen for the first team
    int score_2=0; //displayed score in the playing screen for the second team
    int play_cheer=0; //variable to make sure the cheering sound effect plays only once
    bool isRunning = true; //needed for the main loops condition
    SDL_Event event;
    srand(time(NULL));
```

Figure 18. Variable declarations and assignments in the main() function

For completed and selected, the values are arranged as GK, LCB, RCB, LB, RB, CDM, LCM, RCM, ST, LW, RW. With the index 0 being GK, 1 being LCB, 2 being RCB and so on.

```c
    //SDL initiation
    if (SDL_Init(SDL_INIT_VIDEO) != 0) {
        printf("SDL initialization failed: %s\n", SDL_GetError());
        return 1;
    }

    //window creation
    SDL_Window *window = SDL_CreateWindow("Football Drafts simulator", SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, WINDOW_WIDTH, WINDOW_HEIGHT, SDL_WINDOW_SHOWN);
    if (window == NULL) {
        printf("Failed to create window: %s\n", SDL_GetError());
        SDL_Quit();
        return 1;
    }
    //rederer creation
    SDL_Renderer *renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
    if (renderer == NULL) {
        printf("Failed to create renderer: %s\n", SDL_GetError());
        SDL_DestroyWindow(window);
        SDL_Quit();
        return 1;
    }
    //intiating the image
    int imgFlags = IMG_INIT_PNG;
    if (!(IMG_Init(imgFlags) & imgFlags)) {
        printf("SDL_image initialization failed: %s\n", IMG_GetError());
        SDL_DestroyRenderer(renderer);
        SDL_DestroyWindow(window);
        SDL_Quit();
        return 1;
    }
```

```c
    //initilising text
    TTF_Init();
    //initiating music and sound. Also opening audio channel
    Mix_Init(MIX_INIT_MP3);
    Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 2048);
    //initialising font and color
    TTF_Font *font=TTF_OpenFont("Russia Five.ttf",25);
    SDL_Color color={255,255,255};
```

Figure 19 and 20. Initializations

```c
    //define maximum number of players
    int max_players = 500;
    Player* players = malloc(max_players * sizeof(Player));
    if (players == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    //read players details from file
    int numofplayers = readPlayers(players, max_players);
    if (numofplayers == 0) {
        printf("No players found in the file.\n");
        free(players);
        SDL_DestroyWindow(window);
        SDL_Quit();
        return 1;
    }
```

Figure 21

In this part of the function, the variable max_players is declared and assigned 500, this indicates only 500 players are allowed. With max_players, memory is allocated to the Player type array named players, and it checks if the memory was successfully allocated or not. Then the variable numofplayers is declared and the readPlayers() function is called. The returned value in the readPlayers() function is assigned to numofplayers. It also checks if the numofplayers is empty or not. If it is empty, it means there are no players in the text file.

```c
    //load select sound affect
    Mix_Chunk* SelectSound = Mix_LoadWAV("audio/select.mp3");
    if (!SelectSound) {
        printf("Failed to load select sound: %s\n", Mix_GetError());
    }

    //load cheering sound affect
    Mix_Chunk* CheeringSound = Mix_LoadWAV("audio/cheering.mp3");
    if (!CheeringSound) {
        printf("Failed to load cheering sound: %s\n", Mix_GetError());
    }

    //load music
    Mix_Music* main_music = Mix_LoadMUS("audio/waka_waka.mp3");
    if (!main_music) {
        printf("Failed to load music track: %s\n", Mix_GetError());
    }
```

Figure 22. creating the surfaces, textures and loading the music and sound effects

```
//loading the main menu
SDL_Surface *main_menu_surface= IMG_Load("images/Main_Menu.png");
SDL_Texture *main_menu_texture= SDL_CreateTextureFromSurface(renderer,main_menu_surface);
SDL_FreeSurface(main_menu_surface);

//loading the esc menu
SDL_Surface *menu_surface= IMG_Load("images/Menu.png");
SDL_Texture *menu_texture= SDL_CreateTextureFromSurface(renderer,menu_surface);
SDL_FreeSurface(menu_surface);

//loading the info screen
SDL_Surface *info_surface= IMG_Load("images/info.png");
SDL_Texture *info_texture= SDL_CreateTextureFromSurface(renderer,info_surface);
SDL_FreeSurface(info_surface);

//loading the submit button
SDL_Surface *SubmitBtn_sur= IMG_Load("images/BtnSubmit.png");
SDL_Texture *SubmitBtn_tex= SDL_CreateTextureFromSurface(renderer,SubmitBtn_sur);
SDL_FreeSurface(SubmitBtn_sur);

//loading the reference
SDL_Surface *ref_sur= IMG_Load("images/reference.png");
SDL_Texture *ref_tex= SDL_CreateTextureFromSurface(renderer,ref_sur);
SDL_FreeSurface(ref_sur);

//loading the slider
SDL_Surface *slide_sur= IMG_Load("images/slider.png");
SDL_Texture *slide_tex= SDL_CreateTextureFromSurface(renderer,slide_sur);
SDL_FreeSurface(slide_sur);

//loading the winning screen
SDL_Surface *win_sur= IMG_Load("images/winning_screen.png");
SDL_Texture *win_tex= SDL_CreateTextureFromSurface(renderer,win_sur);
SDL_FreeSurface(win_sur);
```

Figure 23. creating the surfaces, textures and loading images

```
    //getting the options to display

    Player* displayGK = options(players, numofplayers,displayGK,"GK");
    display_error(displayGK,renderer,window);

    Player* displayLCB = options(players, numofplayers,displayLCB,"LCB");
    display_error(displayLCB,renderer,window);

    Player* displayRCB = options(players, numofplayers,displayRCB,"RCB");
    display_error(displayRCB,renderer,window);

    Player* displayLB = options(players, numofplayers,displayLB,"LB");
    display_error(displayLB,renderer,window);

    Player* displayRB = options(players, numofplayers,displayRB,"RB");
    display_error(displayRB,renderer,window);

    Player* displayCDM = options(players, numofplayers,displayCDM,"CDM");
    display_error(displayCDM,renderer,window);

    Player* displayLCM = options(players, numofplayers,displayLCM,"LCM");
    display_error(displayLCM,renderer,window);

    Player* displayRCM = options(players, numofplayers,displayRCM,"RCM");
    display_error(displayRCM,renderer,window);

    Player* displayST = options(players, numofplayers,displayST,"ST");
    display_error(displayST,renderer,window);

    Player* displayLW = options(players, numofplayers,displayLW,"LW");
    display_error(displayLW,renderer,window);

    Player* displayRW = options(players, numofplayers,displayRW,"RW");
    display_error(displayRW,renderer,window);
```

Figure 24. Getting the options for the first team

In this part of the main function, we call the options() function and store the returned array into its designated display array. It then checks if the array has been successfully assigned.

```
//Load the image and create the texture from surface for the player options
char *filename_gk[3];
SDL_Texture *cardTexture_gk[3];
create_options(filename_gk,displayGK,renderer,window,cardTexture_gk);

char *filename_lcb[3];
SDL_Texture *cardTexture_lcb[3];
create_options(filename_lcb,displayLCB,renderer,window,cardTexture_lcb);

char *filename_rcb[3];
SDL_Texture *cardTexture_rcb[3];
create_options(filename_rcb,displayRCB,renderer,window,cardTexture_rcb);

char *filename_lb[3];
SDL_Texture *cardTexture_lb[3];
create_options(filename_lb,displayLB,renderer,window,cardTexture_lb);

char *filename_rb[3];
SDL_Texture *cardTexture_rb[3];
create_options(filename_rb,displayRB,renderer,window,cardTexture_rb);

char *filename_cdm[3];
SDL_Texture *cardTexture_cdm[3];
create_options(filename_cdm,displayCDM,renderer,window,cardTexture_cdm);

char *filename_lcm[3];
SDL_Texture *cardTexture_lcm[3];
create_options(filename_lcm,displayLCM,renderer,window,cardTexture_lcm);

char *filename_rcm[3];
SDL_Texture *cardTexture_rcm[3];
create_options(filename_rcm,displayRCM,renderer,window,cardTexture_rcm);

char *filename_st[3];
SDL_Texture *cardTexture_st[3];
create_options(filename_st,displayST,renderer,window,cardTexture_st);

char *filename_lw[3];
SDL_Texture *cardTexture_lw[3];
create_options(filename_lw,displayLW,renderer,window,cardTexture_lw);

char *filename_rw[3];
SDL_Texture *cardTexture_rw[3];
create_options(filename_rw,displayRW,renderer,window,cardTexture_rw);
```

Figure 23. loading the images for the player options

Using the display array which we got from the previous codes, the newly declared texture and filename, we call the create_options() function, so that the images are created for the options.

The code in figure 22 and figure 23 is repeated for team 2, but with a different name. The names just have an addition of 2 at the end of the variable name.

```
// Play music
Mix_PlayMusic(main_music, -1);
```

Figure 24. Plays the game music

```
while (isRunning)
{
    while (SDL_PollEvent(&event))
    {
        switch (event.type)
        {
        // Handles OS "Exit" event
        case SDL_QUIT:
            isRunning = false;
            break;

        //accessing the esc menu
        case SDL_KEYDOWN:
            if (event.key.keysym.sym == SDLK_ESCAPE) { //checks if the esc button on the keyboard is pressed
                open_EscMenu = true; //opens the esc menu
            }
            break;
```

Figure 25. Main loop

In this part of the main function, the main loop is present with the isRunning as a condition, if it is equal to true it will keep looping. Inside the main loop there is another while loop to check for any pending events. The first case is when the program is closed, the second case is to check if a key is pressed, which in this program is supposed to be the escape key. If the esc key is pressed it assigns the open_EscMenu variable true, meaning it should open the escape menu.

```
//allowing the user to click buttons and cards
case SDL_MOUSEBUTTONDOWN:
    if (event.button.button == SDL_BUTTON_LEFT)
    {
        //stores the coordinates of where the mouse is clicked
        int mouseX = event.motion.x;
        int mouseY = event.motion.y;

        //these buttons should be able to be pressed only in the main menu
        if (leave_MainMenu==false  && start==false){
            if (mouseX > 550 && mouseX < 950){
                if (mouseY > 350 && mouseY < 500){ //next button
                    Mix_PlayChannel(-1, SelectSound, 0);
                    leave_MainMenu = true;
                    SDL_DestroyTexture(main_menu_texture);
                }
                else if (mouseY > 550 && mouseY < 700){ //quit button
                    Mix_PlayChannel(-1, SelectSound, 0);
                    printf("program was closed from main menu");
                    SDL_DestroyWindow(window);
                    SDL_Quit();
                    return 1;
                }
            }
        }else if(leave_MainMenu=true && leave_info==false){
            //buttons should be able to be pressed only in the info page
            if (mouseX > 650 && mouseX < 850){
                if (mouseY > 690 && mouseY < 770){ //start button
                    Mix_PlayChannel(-1, SelectSound, 0);
                    start = true;
                    SDL_DestroyTexture(info_texture);
                    leave_info=true;
                }
            }
        }
```

Figure 26.

The next case checks for the mouse button clicks, specifically the left click. It first stores the coordinates of the left click into variables. The first if condition is meant for the main menu's start and quit button, and then the else if condition is for the info page's start button. Boolean values are used to make sure these buttons can only be pressed in their respected screens. In all the conditions if the button is pressed it will play the select sound effect.

```
//main playing screen, the  buttons and cards can only be clicked if this condiiton is true
if (start==true && open_EscMenu== false ){
    if (completed[0] == false){ //checks if the GK position has been selected or not
        if (team2_ongoing==false){ //checks if the selection for team 2 started
            //assigns the selected card's index into its respected spot in the selected array
            selected[0]=select(mouseX, mouseY, filename_gk, cardTexture_gk, &completed[0], SelectSound);
            TeamOne[0]=displayGK[selected[0]]; //stores the selected player into the first team's final array
            team_rating= total_rating(displayGK,selected[0],team_rating); //calculates and stores  the first teams current rating
        }else if (team2_ongoing==true){ //checks if the selection for team 2 started
            selected[0]=select(mouseX, mouseY, filename_gk2, cardTexture_gk2, &completed[0], SelectSound);
            TeamTwo[0]=displayGK2[selected[0]]; //stores the selected player into the second team's final array
            team_rating_2= total_rating(displayGK2,selected[0],team_rating_2);//calculates and stores the second teams current rating
        }
    }
    else if (completed[1] == false){//checks if the LCB position has been selected or not
        if (team2_ongoing==false){
            selected[1]=select(mouseX, mouseY, filename_lcb, cardTexture_lcb, &completed[1], SelectSound);
            TeamOne[1]=displayLCB[selected[1]];
            team_rating= total_rating(displayLCB,selected[1],team_rating);
        }else if (team2_ongoing==true){
            selected[1]=select(mouseX, mouseY, filename_lcb2, cardTexture_lcb2, &completed[1], SelectSound);
            TeamTwo[1]=displayLCB2[selected[1]];
            team_rating_2= total_rating(displayLCB2,selected[1],team_rating_2);
        }
    }
    else if (completed[2] == false){//checks if the RCB position has been selected or not
        if (team2_ongoing==false){
            selected[2]=select(mouseX, mouseY, filename_rcb, cardTexture_rcb, &completed[2], SelectSound);
            TeamOne[2]=displayRCB[selected[2]];
            team_rating= total_rating(displayRCB,selected[2],team_rating);
        }else if (team2_ongoing==true){
            selected[2]=select(mouseX, mouseY, filename_rcb2, cardTexture_rcb2, &completed[2], SelectSound);
            TeamTwo[2]=displayRCB2[selected[2]];
            team_rating_2= total_rating(displayRCB2,selected[2],team_rating_2);
        }
    }
    else if (completed[3] == false){//checks if the LB position has been selected or not
        if (team2_ongoing==false){
            selected[3]=select(mouseX, mouseY, filename_lb, cardTexture_lb, &completed[3], SelectSound);
            TeamOne[3]=displayLB[selected[3]];
            team_rating= total_rating(displayLB,selected[3],team_rating);
        }else if (team2_ongoing==true){
            selected[3]=select(mouseX, mouseY, filename_lb2, cardTexture_lb2, &completed[3], SelectSound);
            TeamTwo[3]=displayLB2[selected[3]];
```

```
        }
    else if (completed[10] == false){//checks if the RW position has been selected or not
        if (team2_ongoing==false){
            selected[10]=select(mouseX, mouseY, filename_rw, cardTexture_rw, &completed[10], SelectSound);
            TeamOne[10]=displayRW[selected[10]];
            team_rating= total_rating(displayRW,selected[10],team_rating);
            final_chem= total_chemisty(TeamOne); //calculates the first teams chemisty
        }else if (team2_ongoing==true){
            selected[10]=select(mouseX, mouseY, filename_rw2, cardTexture_rw2, &completed[10], SelectSound);
            TeamTwo[10]=displayRW2[selected[10]];
            team_rating_2= total_rating(displayRW2,selected[10],team_rating_2);
            final_chem_2= total_chemisty(TeamTwo); //calculates the second teams chemisty
        }
    }
}
```

Figure 27 and 28.

Now the main card selection starts, once start is equal to true and open_EscMenu is false, the cards can be selected. Selection of the cards are similar for each position, it first checks if the card has been selected or not. If it is not, it checks which teams' selection is currently going on, if for example it is still the first team's selection, it will call the select() function and assign its returned value into its designated position in the selected array. Then it stores the selected player into the TeamOne array and finally calculates the current team's rating. In figure 28, it performs the same as the other if conditions however with the addition of it calculating the team's chemistry for both team 1 and 2.

```
if (open_EscMenu==true){ //checks if the user wants to open the esc menu
    //enabling the user to change the value
    if (mouseX >= 650 && mouseX <= 850 && mouseY >= 280 && mouseY <= 300) {
        // Calculate new volume based on mouse position
        volume = (mouseX - 650) * MIX_MAX_VOLUME / 200;
        Mix_VolumeMusic(volume);
        VolSliderAdd=(mouseX-650);

    }

    if (mouseX > 550 && mouseX < 950){//enabling the user to click on the start or quit button
        if (mouseY > 350 && mouseY < 500){ //start button
            Mix_PlayChannel(-1, SelectSound, 0);
            open_EscMenu = false;
        }
        else if (mouseY > 550 && mouseY < 700){ //quit button
            Mix_PlayChannel(-1, SelectSound, 0);
            printf("program was closed from the esc menu");
            SDL_DestroyWindow(window);
            SDL_Quit();
            return 1;
        }
    }
}
```

Figure 28. allows the Escape menu button to be clicked

Once the escape menu is opened it has 3 options, the first option is to change the volume, the user can click anywhere in the long rectangle to change the volume. The other two options are the continue and quit button, respectively. Once the continue button is pressed, the Boolean value is assigned false, hence the buttons cannot be pressed again.

```
if (winning_screen==true){ //checks if the winning screen should be opened
    if (mouseX > 550 && mouseX < 950 && mouseY > 550 && mouseY < 700){ //quit button
        Mix_PlayChannel(-1, SelectSound, 0);
        printf("program was closed from the winning screen");
        SDL_DestroyWindow(window);
        SDL_Quit();
        return 1;
    }
}

if (submit==true){ //checks if the submit button is open
    if (mouseX > 1275 && mouseX < 1475 && mouseY > 650 && mouseY < 750){ //submit button
        Mix_PlayChannel(-1, SelectSound, 0);
        if (team2_ongoing==false){ //checks if team 2 has started
            for (int x=0;x<11;x++){ //resets the completed and selected array for team 2
                completed[x] = false;
                selected[x]=-1;
            }
            team2_ongoing=true; //allows team 2 to start selection
            submit=false; //turns the submit button off
        }else{ //checks if team 2 has already been selected
            winning_screen=true; //open the winning screen
            submit=false; //turns off the button
        }
    }
}
}
break;
}
}
```

Figure 29. allows the winning screen and the submit buttons to be clicked

If the winning screen is open the user can click on the quit button's coordinates, to close the program. If the user completes their team, the submit button is open but it also checks if the second team has been started. If it has not, it allows it to start. However, if it already started it allows the winning screen to open.

```
//make the background black and clears the render
 SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
 SDL_RenderClear(renderer);

 if (start==true && open_EscMenu== false){ //checks if the game started and if the esc menu is not open
     drawField(renderer); //draws the football field

     SDL_Rect refRect = {1275, 150, 200, 475};
     SDL_RenderCopy(renderer, ref_tex, NULL, &refRect); //renders the team positions reference

     load_text(renderer,font,color,"Current Score:",1255,50,35,30); //renders the text onto the screen
     if (team2_ongoing==false){ //checks if team 2 started
         load_text(renderer,font,color,"Team 1",700,5,15,15); //loads the team 1 text onto the screen
         if (final_score==false){ //checks if the final score is ready
             score=team_rating * 0.1; //makes the score more presentable
             load_score(renderer,font,color,score); //renders the score
         }else{
             score=(team_rating+final_chem) * 0.1; //adds the chemistry if the final score is ready
             load_score(renderer,font,color,score); //renders the score
         }
     }else if (team2_ongoing==true){
         load_text(renderer,font,color,"Team 2",700,5,15,15);//loads the team 2 text onto the screen
         if (final_score_2==false){
             score_2=team_rating_2 * 0.1;
             load_score(renderer,font,color,score_2);
         }else{
             score_2=(team_rating_2+final_chem_2) * 0.1;
             load_score(renderer,font,color,score_2);
         }
     }
```

Figure 30.

In this part, it first makes the background color black and clears the render. It checks if the game has started and if the esc menu is closed. If so, it will draw the football field and the reference image is rendered to give users guidance. It then checks which team is being selected, depending on which team, it renders a text on the top of the screen. It also checks if the final score is ready, if it is not, it will render the normal score until the final score is ready. This is the same for team two as well.

```
921             //load the selected card to its new position
922             if (completed[0]==true){ //checks if the player was selected
923                 if (team2_ongoing==false){ //decides which team is ongoing
924                     selected_card(705,625,selected[0],cardTexture_gk,renderer); //renders the selected card onto its new position
925                 }else if(team2_ongoing==true){
926                     selected_card(705,625,selected[0],cardTexture_gk2,renderer);
927                 }
928 
929                 if  (completed[1]==true){
930                     if (team2_ongoing==false){
931                         selected_card(550,500,selected[1],cardTexture_lcb,renderer);
932                     }else if(team2_ongoing==true){
933                         selected_card(550,500,selected[1],cardTexture_lcb2,renderer);
934                     }
935 
936                     if  (completed[2]==true){
937                         if (team2_ongoing==false){
938                             selected_card(850,500,selected[2],cardTexture_rcb,renderer);
939                         }else if(team2_ongoing==true){
940                             selected_card(850,500,selected[2],cardTexture_rcb2,renderer);
941                         }
942 
943                         if  (completed[3]==true){
944                             if (team2_ongoing==false){
945                                 selected_card(300,450,selected[3],cardTexture_lb,renderer);
946                             }else if(team2_ongoing==true){
947                                 selected_card(300,450,selected[3],cardTexture_lb2,renderer);
948                             }
949 
950                             if  (completed[4]==true){
951                                 if (team2_ongoing==false){
952                                     selected_card(1100,450,selected[4],cardTexture_rb,renderer);
953                                 }else if(team2_ongoing==true){
954                                     selected_card(1100,450,selected[4],cardTexture_rb2,renderer);
```

```
                                                }

                                                if  (completed[10]==true){
                                                    if (team2_ongoing==false){
                                                        selected_card(1100,100,selected[10],cardTexture_rw,renderer);
                                                    }else if(team2_ongoing==true){
                                                        selected_card(1100,100,selected[10],cardTexture_rw2,renderer);
                                                    }

                                                    final_score=true; //lets us know that the final score is ready
                                                    submit=true; //opens the submit button
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
                }
            }
```

Figure 31 and 32. Loading the selected cards into its new positions

It first checks if the card has been selected by the user, it then checks which team selection is currently going on, finally then renders the selected card into its new position. The same code and logic are repeated for all positions but with different coordinates. However, in the final position it has two more lines of code where it assigns true to final_score indicating the final score is ready and allows the submit button to operate as well.

```
//once the previous card is stored in its new position, the next batch of cards will be displayed
int textX=10; //sets the text's X coordinate
int textY=50; //sets the text's Y coordinate
int textW=10; //add to the text's width
int textH=10; //add to the text's height

if (completed[0]==false){ //checks if the GK card is not selected
    load_text(renderer,font,color,"Select your GK",textX,textY,textW,textH); //renders the text
    if (team2_ongoing==false){ //check which team is currently being selected
        display_options(renderer,cardTexture_gk); //display the options for the user to select from
    }else if(team2_ongoing==true){
        display_options(renderer,cardTexture_gk2);
    }
}else if (completed[1]==false){ //checks if the LCB card is not selected
    load_text(renderer,font,color,"Select your LCB",textX,textY,textW,textH);
    if (team2_ongoing==false){
        display_options(renderer,cardTexture_lcb);
    }else if(team2_ongoing==true){
        display_options(renderer,cardTexture_lcb2);
    }

}else if (completed[2]==false){ //checks if the RCB card is not selected
    load_text(renderer,font,color,"Select your RCB",textX,textY,textW,textH);
    if (team2_ongoing==false){
        display_options(renderer,cardTexture_rcb);
    }else if(team2_ongoing==true){
        display_options(renderer,cardTexture_rcb2);
    }
}else if (completed[3]==false){ //checks if the LB card is not selected
    load_text(renderer,font,color,"Select your LB",textX,textY,textW,textH);
    if (team2_ongoing==false){
        display_options(renderer,cardTexture_lb);
    }else if(team2_ongoing==true){
        display_options(renderer,cardTexture_lb2);
    }
```

Figure 33. renders texts and options

In this part of the main() function, we first declare and assign the texts coordinates and size, followed by nested if statements. The first if condition checks if the user selected a player, if not it renders the text with the position needed to be selected, with another if condition which checks which team is currently ongoing, and depending on that, it will output the right set of options for the user to select. This is repeated for all the positions.

```
        //rendering the menu image
        if (open_EscMenu==true) {
            SDL_Rect menurect = {0, 0, WINDOW_WIDTH, WINDOW_HEIGHT};
            SDL_RenderCopy(renderer, menu_texture, NULL, &menurect);

            /*positions the slider image, with the width to depend on the users prefrence.
              But initially set at half*/
            SDL_Rect SliderRect = {651, 282, (VolSliderAdd-2), 16};
            SDL_RenderCopy(renderer, slide_tex, NULL, &SliderRect);//renders the slider image
            load_volume(renderer, font, color, volume); //renders the volume percentage
        }

        //rendering the submit button
        if (submit==true && open_EscMenu== false){
            SDL_Rect SubmitBtnRect = {1275, 650, 200, 100};
            SDL_RenderCopy(renderer, SubmitBtn_tex, NULL, &SubmitBtnRect);
        }
        //rendering the main menu
        if (leave_MainMenu==false){ //checks if the main menu is open
        SDL_Rect mainrect = {0, 0, WINDOW_WIDTH, WINDOW_HEIGHT};
        SDL_RenderCopy(renderer, main_menu_texture, NULL, &mainrect);
        }else if(leave_MainMenu==true){ //checks if the main menu is closed
            SDL_Rect inforect = {0, 0, WINDOW_WIDTH, WINDOW_HEIGHT};
            SDL_RenderCopy(renderer, info_texture, NULL, &inforect); //renders the info page once
        }
```

Figure 34. Renders images

In this part it renders certain images like the main menu, escape menu, info screen, submit button and the volume slider. If keeps on checking if it should be opened or not, for example the submit button needs to have the submit variable to be true and the open_EscMenu closed. In the escape menu the volume slider is located, where the image will move depending on the user's volume selection.

```
        if (winning_screen==true){ //checks if the winning screen should be opened
            SDL_Rect winrect = {0, 0, WINDOW_WIDTH, WINDOW_HEIGHT};
            SDL_RenderCopy(renderer, win_tex, NULL, &winrect); //renders the winning screen

            if (score>score_2){ //checks if team 1 score is bigger than team 2
                while (play_cheer<1){ //allows the sound to play once
                    Mix_Volume(-1, MIX_MAX_VOLUME / 2); //sets the sound effects volume to half
                    Mix_PlayChannel(-1, CheeringSound, 0); //plays the cheering sound effect
                    play_cheer++; //increments play_cheer so that it exits the loop
                }
                //renders the winner in text
                load_text(renderer,font,color,"Team 1 won!!",575,100,200,100);
            }else if (score<score_2){ //checks if team 2 score is bigger than team 1
                while (play_cheer<1){
                    Mix_Volume(-1, MIX_MAX_VOLUME / 2);
                    Mix_PlayChannel(-1, CheeringSound, 0);
                    play_cheer++;
                }
                load_text(renderer,font,color,"Team 2 won!!",575,100,200,100);
            }else{ //if its a tie
                // renders a text mentioning its a tie
                load_text(renderer,font,color,"It was a tie!!",575,100,200,100);
            }
        }
        SDL_RenderPresent(renderer);

    }
```

Figure 34. Renders the winner screen with the winner

```
    //freeing, destroying and quiting arrays,textures, render, SDL, etc...
    Mix_FreeChunk(SelectSound);
    Mix_FreeMusic(main_music);
    Mix_CloseAudio();
    free(players);
    free_and_destroy(displayGK,filename_gk,cardTexture_gk);
    free_and_destroy(displayLCB,filename_lcb,cardTexture_lcb);
    free_and_destroy(displayRCB,filename_rcb,cardTexture_rcb);
    free_and_destroy(displayLB,filename_lb,cardTexture_lb);
    free_and_destroy(displayRB,filename_rb,cardTexture_rb);
    free_and_destroy(displayCDM,filename_cdm,cardTexture_cdm);
    free_and_destroy(displayLCM,filename_lcm,cardTexture_lcm);
    free_and_destroy(displayRCM,filename_rcm,cardTexture_rcm);
    free_and_destroy(displayST,filename_st,cardTexture_st);
    free_and_destroy(displayLW,filename_lw,cardTexture_lw);
    free_and_destroy(displayRW,filename_rw,cardTexture_rw);
    free_and_destroy(displayGK2,filename_gk2,cardTexture_gk2);
    free_and_destroy(displayLCB2,filename_lcb2,cardTexture_lcb2);
    free_and_destroy(displayRCB2,filename_rcb2,cardTexture_rcb2);
    free_and_destroy(displayLB2,filename_lb2,cardTexture_lb2);
    free_and_destroy(displayRB2,filename_rb2,cardTexture_rb2);
    free_and_destroy(displayCDM2,filename_cdm2,cardTexture_cdm2);
    free_and_destroy(displayLCM2,filename_lcm2,cardTexture_lcm2);
    free_and_destroy(displayRCM2,filename_rcm2,cardTexture_rcm2);
    free_and_destroy(displayST2,filename_st2,cardTexture_st2);
    free_and_destroy(displayLW2,filename_lw2,cardTexture_lw2);
    free_and_destroy(displayRW2,filename_rw2,cardTexture_rw2);
    SDL_DestroyTexture(main_menu_texture);
    SDL_DestroyTexture(menu_texture);
    SDL_DestroyTexture(info_texture);
    SDL_DestroyTexture(slide_tex);
    SDL_DestroyTexture(SubmitBtn_tex);
    SDL_DestroyTexture(ref_tex);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    TTF_Quit();
    IMG_Quit();
    SDL_Quit();

    return 0;
}
```

Figure 35. freeing, destroying and quitting arrays, textures, render, SDL, etc...

Conclusion

In this report we have gone through a detailed C program for creating an engaging, interactive and SDL based football team selection game. It covers essential functions such as initialization, resource cleanup, game state updates, user input handling, loading game materials, and rendering of game objects. When it comes to operations like text rendering, player card display/rendering, game state management, and user input, the program involves an organized approach with different functions. Additionally, it uses audio enhancements to improve the gaming experience.