DODGE DASH

CAR GAME

FIRST YEAR

BATCH = 2024-2028 & 2023-2027

Course Instructer: Ms.Samia Masood Awan

Group Members:

- *Aliza Kanwal CT -206*
- *Sameen Ali CT -207*
- *Maheen Jamali CR-23030*

TABLE OF CONTENTS

CHAPTER- 1

1-INTRODUCTION

1.1-Purpose

This is a simple console-based car game implemented in the C programming language. It seems to have been used in a project to satisfy the users with entertainment and an interactive process, moving through virtual obstacles and spikes to score points. The game features car selection and color customization, car battle with dynamic elements of gameplay, auditory effects, and user instructions**.**

1.2-Project's Objective

This code is a console-based car game written in C. The player could choose between a car of Sarge and McQueen, with the ability to choose the color between red, green, and blue. The game was designed to score by driving through obstacles and spikes around one's chosen car to obtain a higher score. The game has been made with sound effects, dynamic elements of gameplay, and an easy-to-use simple interface.

1.3-Key Concepts

Before diving into the code, let's understand some key concepts used in the game:

Game State:

The game can be in two states playing and over. When the game is over, the player cannot control the car anymore.

Score:

The player earns points by successfully navigating the car through obstacles.

Lives:

The player starts with a predetermined number of lives. Every time the player hits something, lives diminish

.

Walls:

The game has two walls left and right. The walls move down and require the player to not hit those walls.

Spikes:

The walls are covered with spikes that can be hazardous to the player's car in the event of collision. **Sarge**:

The player has his car as his representation, which is a Sarge. The Sarge car will move left to right in order to dodge the obstacles.

McQueen:

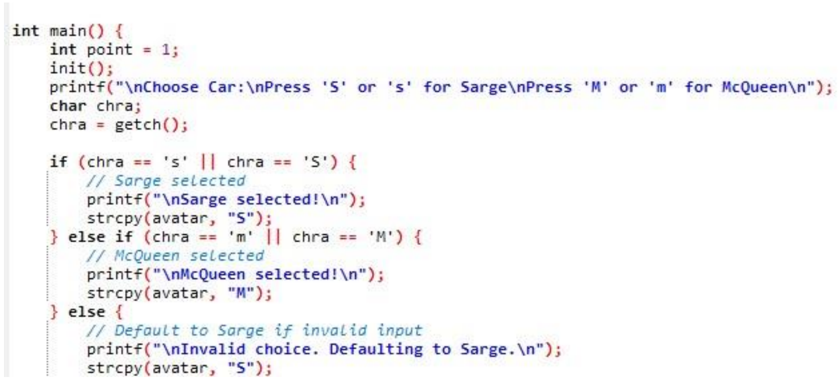
It controls a car represented by the character of McQueen. McQueen can move left and right to avoid the obstacles in the game.

CHAPTER 2

2-FEATURES

2.1-Car selection

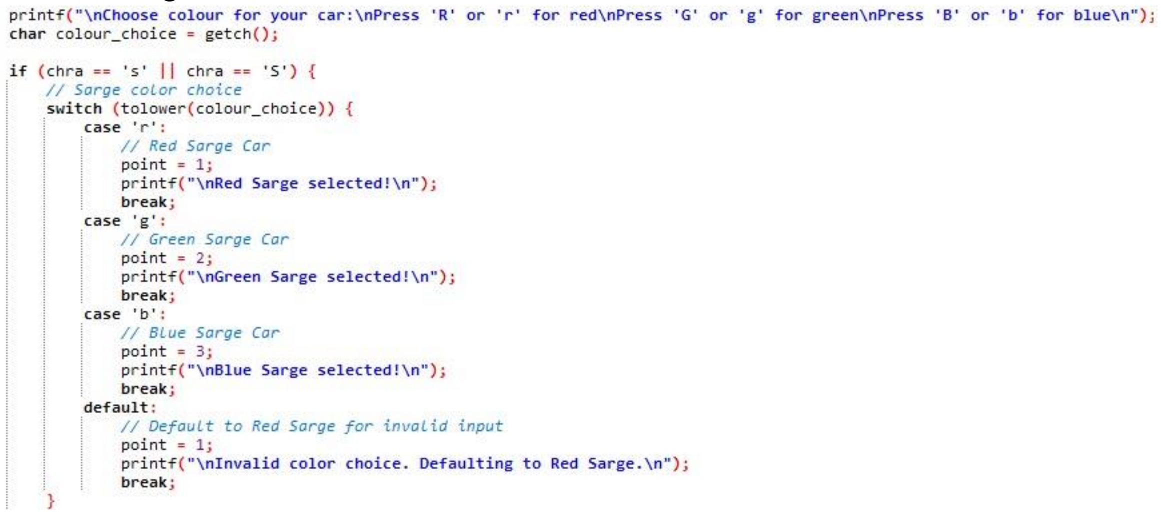
• Players can choose between a Sarge Car (represented by 'S') or a McQueen Car (represented by 'M').



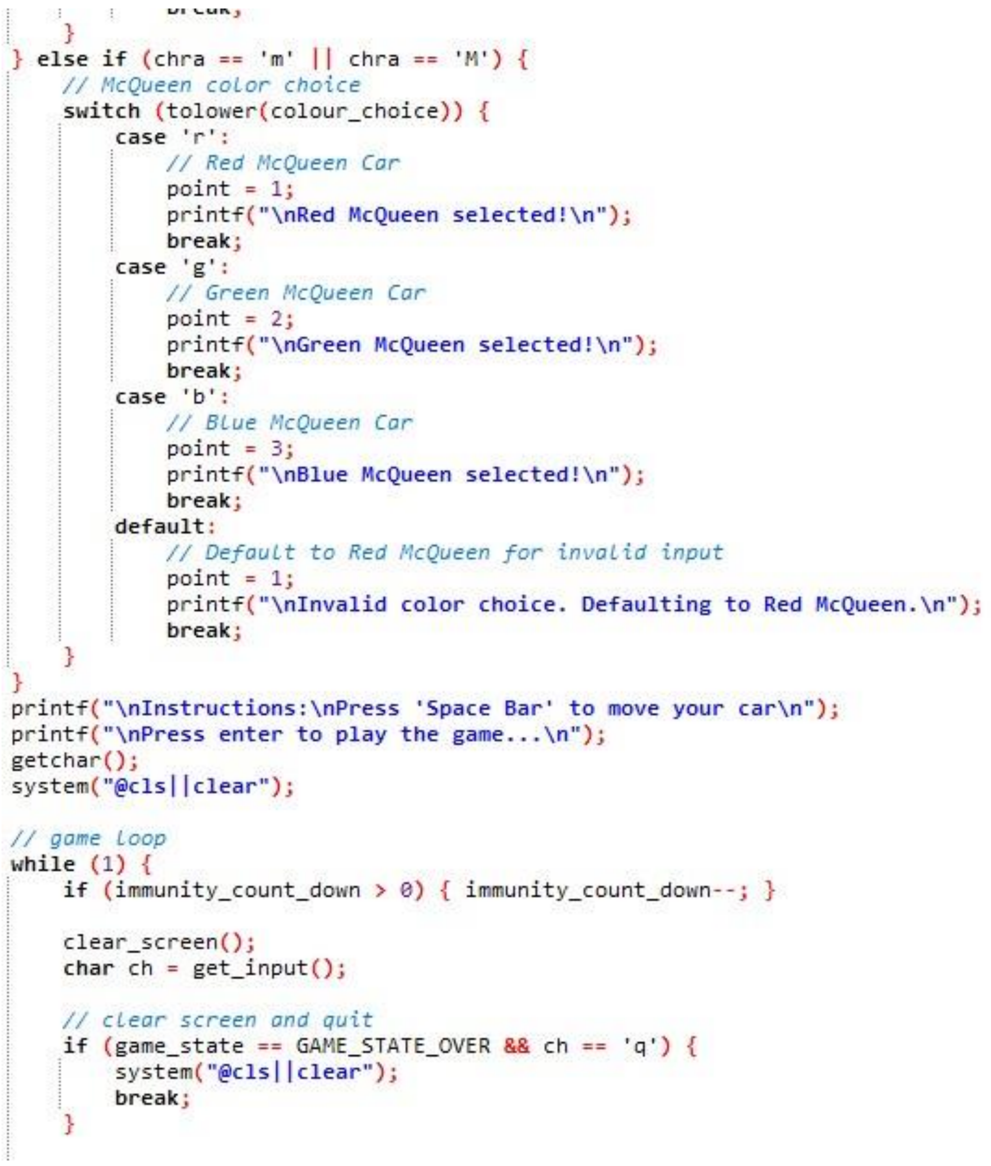
```c
int main() {
    int point = 1;
    init();
    printf("\nChoose Car:\nPress 'S' or 's' for Sarge\nPress 'M' or 'm' for McQueen\n");
    char chra;
    chra = getch();

    if (chra == 's' || chra == 'S') {
        // Sarge selected
        printf("\nSarge selected!\n");
        strcpy(avatar, "S");
    } else if (chra == 'm' || chra == 'M') {
        // McQueen selected
        printf("\nMcQueen selected!\n");
        strcpy(avatar, "M");
    } else {
        // Default to Sarge if invalid input
        printf("\nInvalid choice. Defaulting to Sarge.\n");
        strcpy(avatar, "S");
```

2-Colour customization

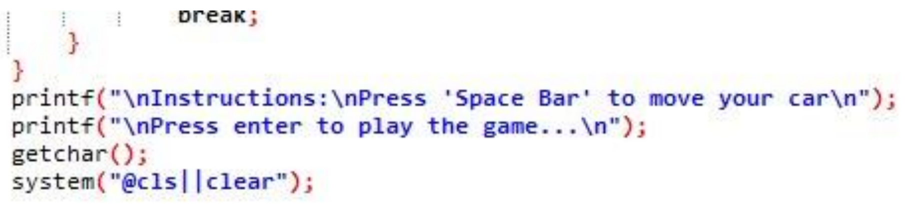
• Users can customize the color of their chosen of both Sarge car and McQueen car by selecting from red, green, or blue



```c
printf("\nChoose colour for your car:\nPress 'R' or 'r' for red\nPress 'G' or 'g' for green\nPress 'B' or 'b' for blue\n");
char colour_choice = getch();

if (chra == 's' || chra == 'S') {
    // Sarge color choice
    switch (tolower(colour_choice)) {
        case 'r':
            // Red Sarge Car
            point = 1;
            printf("\nRed Sarge selected!\n");
            break;
        case 'g':
            // Green Sarge Car
            point = 2;
            printf("\nGreen Sarge selected!\n");
            break;
        case 'b':
            // Blue Sarge Car
            point = 3;
            printf("\nBlue Sarge selected!\n");
            break;
        default:
            // Default to Red Sarge for invalid input
            point = 1;
            printf("\nInvalid color choice. Defaulting to Red Sarge.\n");
            break;
    }
```



```c
            break;
    }
} else if (chra == 'm' || chra == 'M') {
    // McQueen color choice
    switch (tolower(colour_choice)) {
        case 'r':
            // Red McQueen Car
            point = 1;
            printf("\nRed McQueen selected!\n");
            break;
        case 'g':
            // Green McQueen Car
            point = 2;
            printf("\nGreen McQueen selected!\n");
            break;
        case 'b':
            // Blue McQueen Car
            point = 3;
            printf("\nBlue McQueen selected!\n");
            break;
        default:
            // Default to Red McQueen for invalid input
            point = 1;
            printf("\nInvalid color choice. Defaulting to Red McQueen.\n");
            break;
    }
}
printf("\nInstructions:\nPress 'Space Bar' to move your car\n");
printf("\nPress enter to play the game...\n");
getchar();
system("@cls||clear");

// game loop
while (1) {
    if (immunity_count_down > 0) { immunity_count_down--; }

    clear_screen();
    char ch = get_input();

    // clear screen and quit
    if (game_state == GAME_STATE_OVER && ch == 'q') {
        system("@cls||clear");
        break;
    }

```

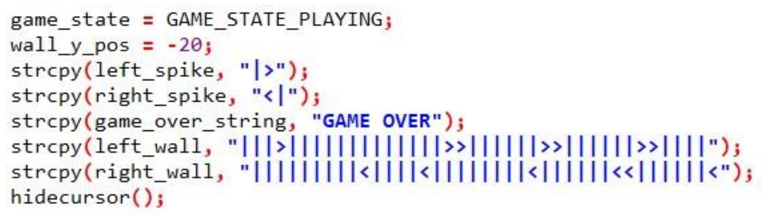


```c
            break;
    }
}
printf("\nInstructions:\nPress 'Space Bar' to move your car\n");
printf("\nPress enter to play the game...\n");
getchar();
system("@cls||clear");
```

2.3-Obstacles

•The player must carry their car through shifting walls with spikes that are shown by symbols ('>', '<|').

•When a collision happens with the spike, decrement of the life is triggered

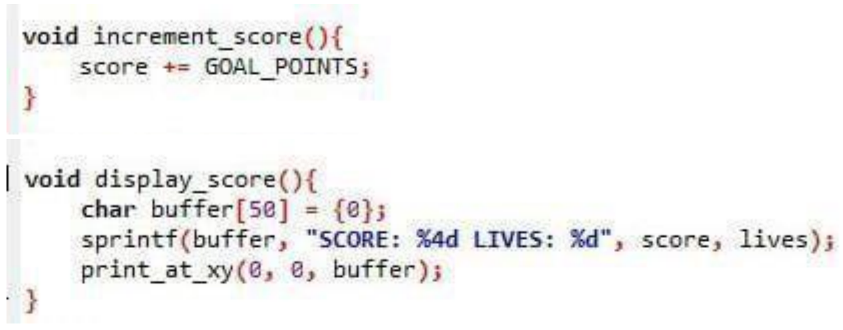


```
game_state = GAME_STATE_PLAYING;
wall_y_pos = -20;
strcpy(left_spike, "|>");
strcpy(right_spike, "<|");
strcpy(game_over_string, "GAME OVER");
strcpy(left_wall, "|||>|||||||||||||>>||||||>>||||||>>||||");
strcpy(right_wall, "|||||||||<||||<||||||||<||||||<<||||||<");
hidecursor();
```

2.4-Score

•Scores are awarded to players for completing the obstacles.

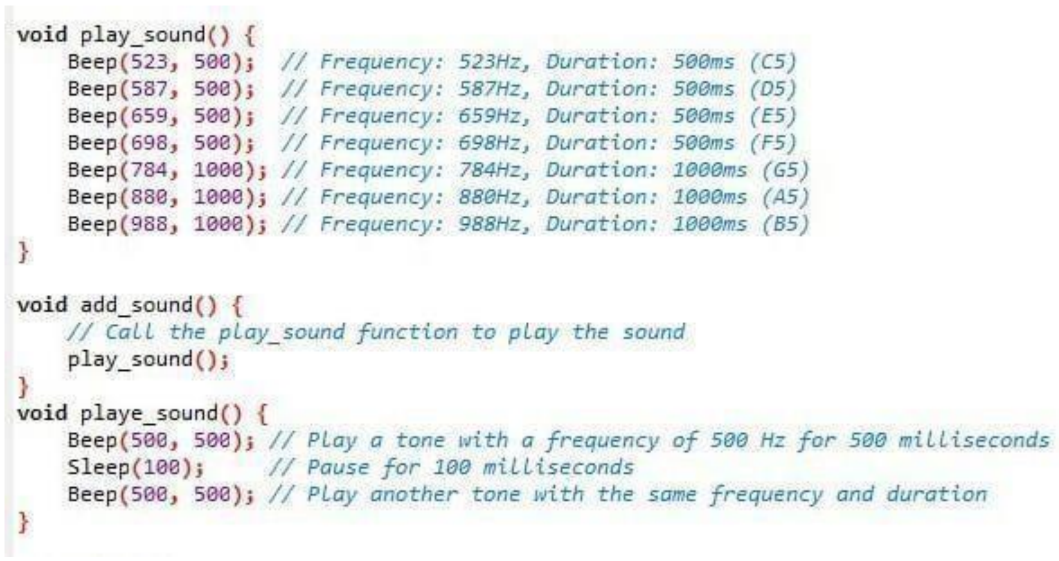
•The goal is to get the highest score.



```
void increment_score(){
    score += GOAL_POINTS;
}

void display_score(){
    char buffer[50] = {0};
    sprintf(buffer, "SCORE: %4d LIVES: %d", score, lives);
    print_at_xy(0, 0, buffer);
}
```

2.5-Sound effects

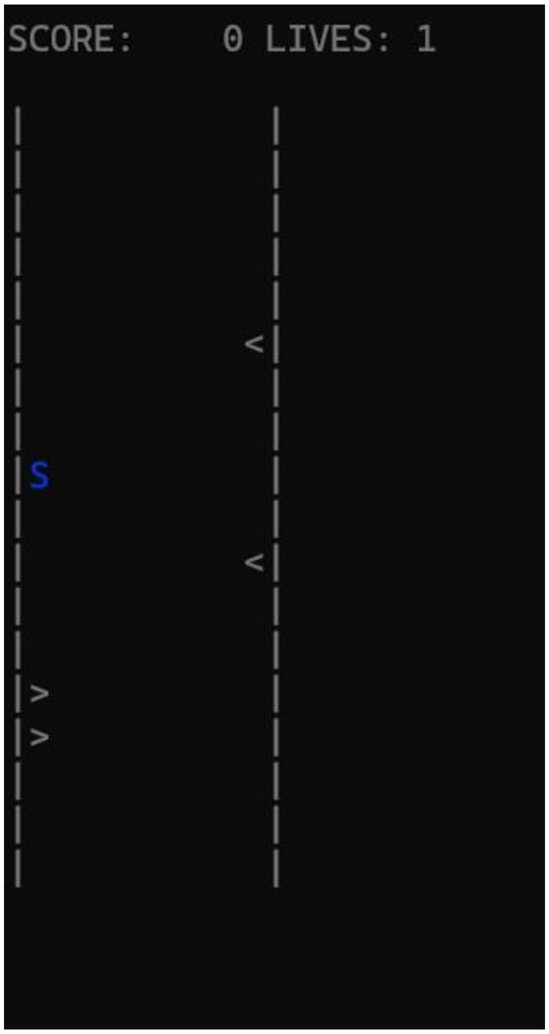
- The game includes sound effects triggered during specific events such as collisions.



```
void play_sound() {
    Beep(523, 500);  // Frequency: 523Hz, Duration: 500ms (C5)
    Beep(587, 500);  // Frequency: 587Hz, Duration: 500ms (D5)
    Beep(659, 500);  // Frequency: 659Hz, Duration: 500ms (E5)
    Beep(698, 500);  // Frequency: 698Hz, Duration: 500ms (F5)
    Beep(784, 1000); // Frequency: 784Hz, Duration: 1000ms (G5)
    Beep(880, 1000); // Frequency: 880Hz, Duration: 1000ms (A5)
    Beep(988, 1000); // Frequency: 988Hz, Duration: 1000ms (B5)
}

void add_sound() {
    // Call the play_sound function to play the sound
    play_sound();
}
void playe_sound() {
    Beep(500, 500); // Play a tone with a frequency of 500 Hz for 500 milliseconds
    Sleep(100);     // Pause for 100 milliseconds
    Beep(500, 500); // Play another tone with the same frequency and duration
}
```

2.6-Dynamic Gameplay

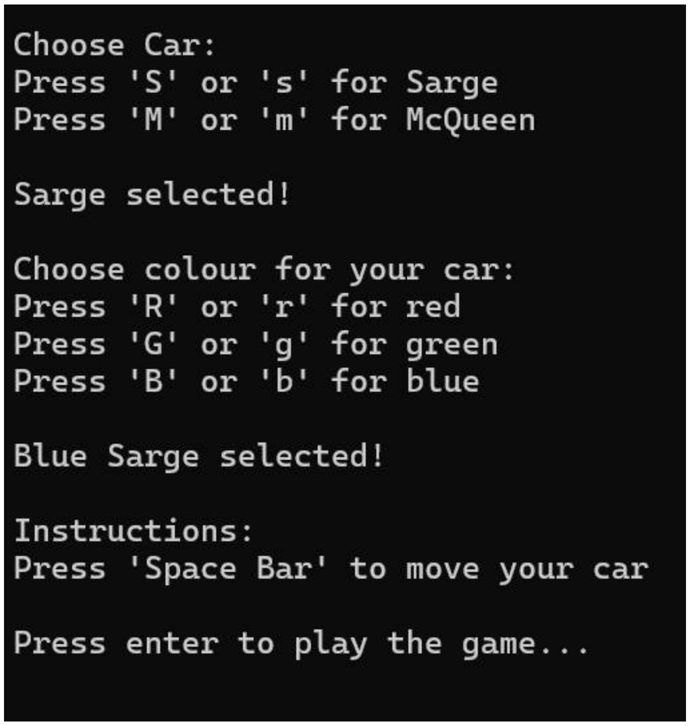
- Moving walls with spikes create a dynamic and challenging environment for the player.



```
SCORE:    0 LIVES: 1

|           |
|           |
|           |
|           |
|           |
|          <|
|           |
|           |
|S          |
|           |
|          <|
|           |
|           |
|>          |
|>          |
|           |
|           |
|           |
```

2.7-User instructions

- Clear instructions guide the player, including the use of the space bar to control the car



```
Choose Car:
Press 'S' or 's' for Sarge
Press 'M' or 'm' for McQueen

Sarge selected!

Choose colour for your car:
Press 'R' or 'r' for red
Press 'G' or 'g' for green
Press 'B' or 'b' for blue

Blue Sarge selected!

Instructions:
Press 'Space Bar' to move your car

Press enter to play the game...
```

CHAPTER-3

3-DESIGN AND ANALYSIS

3.1-Modularity

- The code is modular with well-defined functions for specific tasks, enhancing readability and maintainability.

3.2-Graphics representation

- Graphics are ASCII-based, utilizing symbols to represent cars, spikes, and walls.
- Colors are used to enhance the visual experience in supported terminals

3.3-Input heading

- The game is efficient in handling user input, particularly at a level that will respond to key presses for car control.

3.4-Game loop

- that is as smooth as possible
- A non-stop game loop helps in guaranteeing round-the-clock updates, ensuring an experience**.**

3.5-Error heading

• The code includes error handling for incorrect entries from a user, and instead defaults to a Sarge Car if an invalid choice is made.

3.6-User Interface

• A user interface is simple that shows score, lives and game-related messages.

CHAPTER-4

4-HEADER FILES

4.1-Purpose

Header files are made to create better organization, modularization, and readability. Header files encapsulate related functionality making it easier to understand and maintain different aspects of your code. **stdio.h:**

It represents standard input/output functions. It is being used in case some operations of input/output needed such as printing messages in the console using printf.

stdlib.h:

Standard Library. It includes functions dealing with memory allocation (malloc, free) and other generalpurposes. In our code probably used for general memory related operations.

windows.h:

Windows API functions. This library opens all the Windows specific functions to us like console manipulations, handling messages etc It is used in our code with respect to activities that are performed with a console.

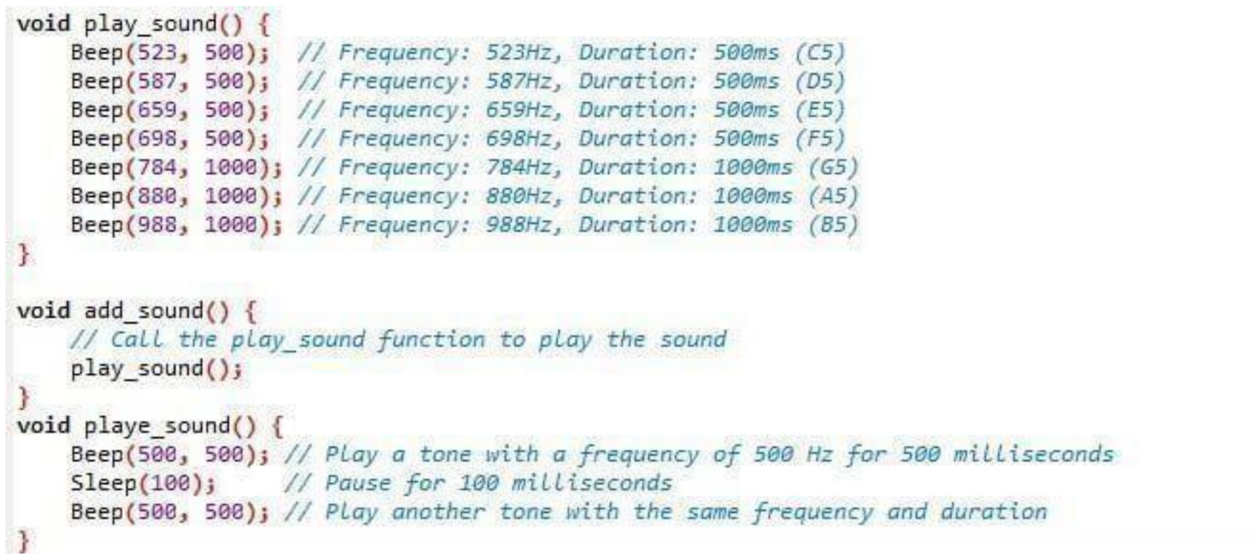
conio.h:

Console Input/Output. This library carries input/output operations using a console. In our code, the function getch is used to receive a character input from a console without echoing.

string.h:

String functions. It provides functions for manipulating strings, like strcpy used in our code for copying strings. **sound.h:**

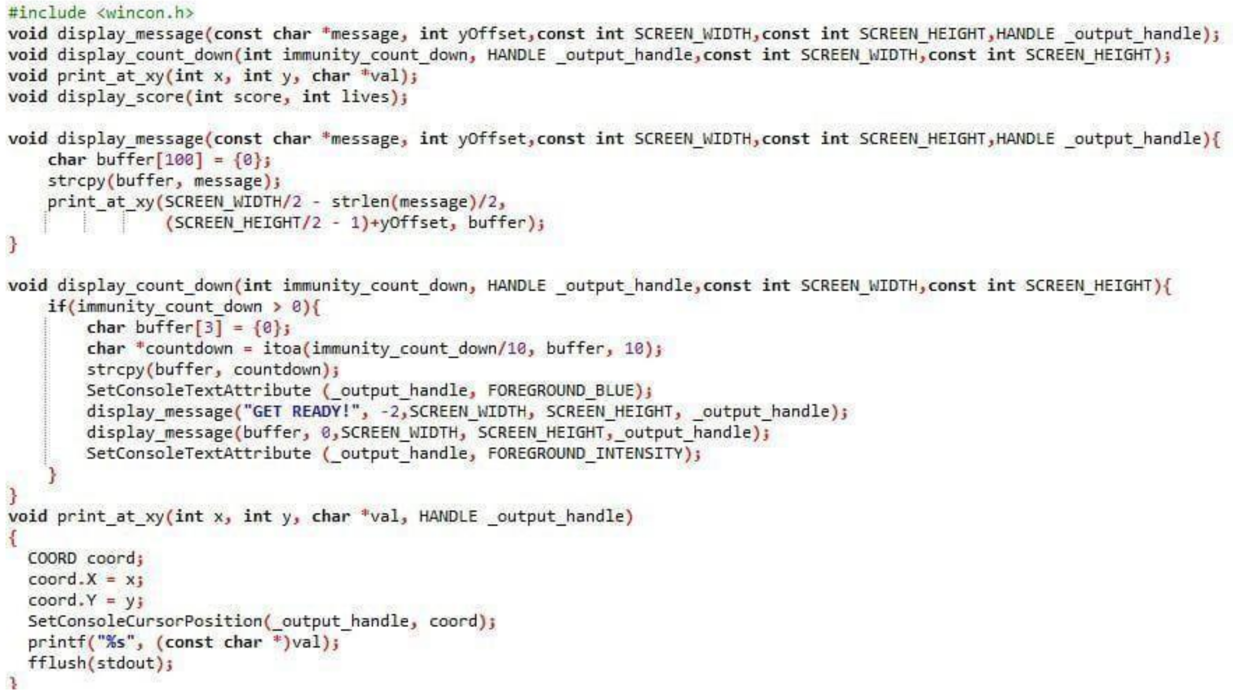
Contains function declarations related to sound effects in the game. This includes functions like play_sound and playe_sound. It helps separate sound-related functionality from the main code for clarity and maintainability.



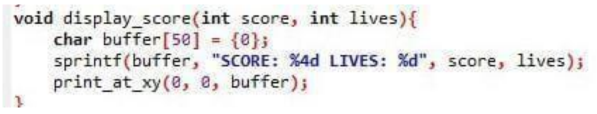
```c
void play_sound() {
    Beep(523, 500);  // Frequency: 523Hz, Duration: 500ms (C5)
    Beep(587, 500);  // Frequency: 587Hz, Duration: 500ms (D5)
    Beep(659, 500);  // Frequency: 659Hz, Duration: 500ms (E5)
    Beep(698, 500);  // Frequency: 698Hz, Duration: 500ms (F5)
    Beep(784, 1000); // Frequency: 784Hz, Duration: 1000ms (G5)
    Beep(880, 1000); // Frequency: 880Hz, Duration: 1000ms (A5)
    Beep(988, 1000); // Frequency: 988Hz, Duration: 1000ms (B5)
}

void add_sound() {
    // Call the play_sound function to play the sound
    play_sound();
}
void playe_sound() {
    Beep(500, 500); // Play a tone with a frequency of 500 Hz for 500 milliseconds
    Sleep(100);     // Pause for 100 milliseconds
    Beep(500, 500); // Play another tone with the same frequency and duration
}
```

display.h:

The one declares functions related to the screen. It includes display-related scores, messages, as well as countdowns. Functions include display score, display messages, and display count-down. This header file isolates the display-related logic, thus managing it would be easier.



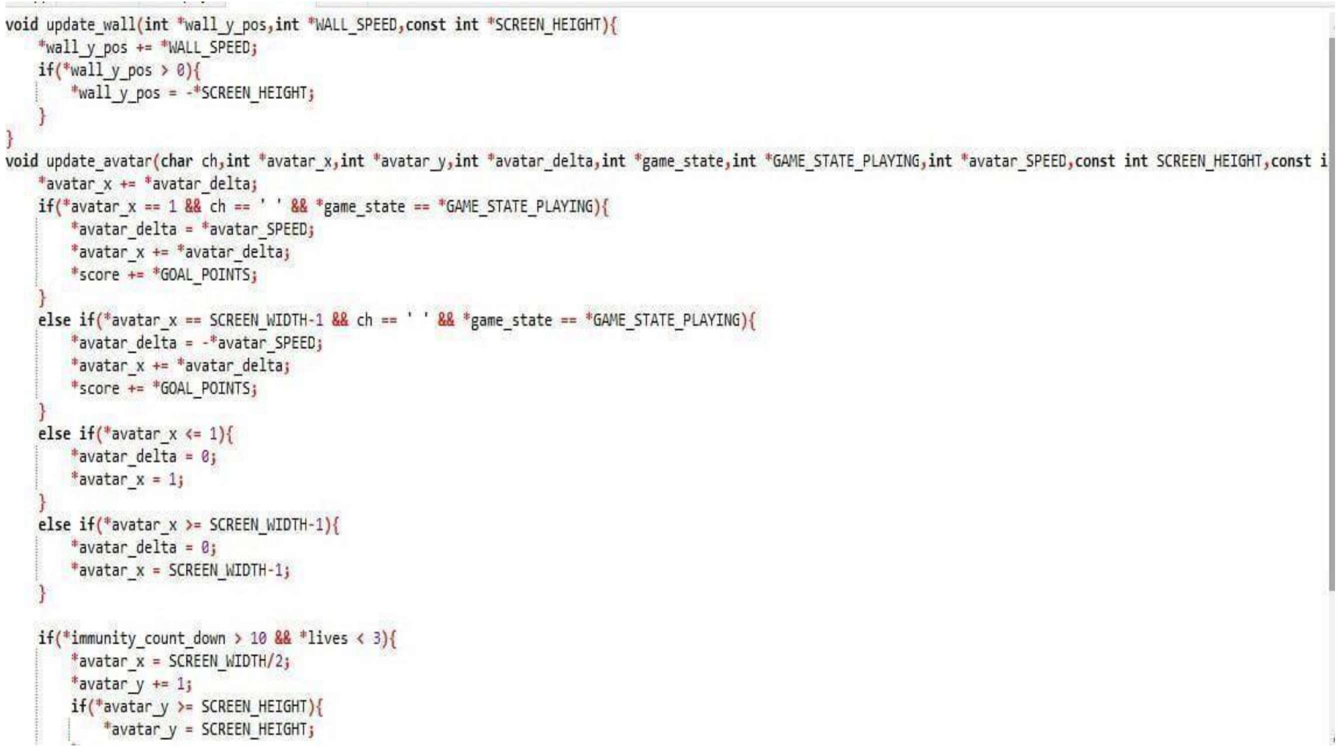
```c
#include <wincon.h>
void display_message(const char *message, int yOffset,const int SCREEN_WIDTH,const int SCREEN_HEIGHT,HANDLE _output_handle);
void display_count_down(int immunity_count_down, HANDLE _output_handle,const int SCREEN_WIDTH,const int SCREEN_HEIGHT);
void print_at_xy(int x, int y, char *val);
void display_score(int score, int lives);

void display_message(const char *message, int yOffset,const int SCREEN_WIDTH,const int SCREEN_HEIGHT,HANDLE _output_handle){
    char buffer[100] = {0};
    strcpy(buffer, message);
    print_at_xy(SCREEN_WIDTH/2 - strlen(message)/2,
                (SCREEN_HEIGHT/2 - 1)+yOffset, buffer);
}

void display_count_down(int immunity_count_down, HANDLE _output_handle,const int SCREEN_WIDTH,const int SCREEN_HEIGHT){
    if(immunity_count_down > 0){
        char buffer[3] = {0};
        char *countdown = itoa(immunity_count_down/10, buffer, 10);
        strcpy(buffer, countdown);
        SetConsoleTextAttribute (_output_handle, FOREGROUND_BLUE);
        display_message("GET READY!", -2,SCREEN_WIDTH, SCREEN_HEIGHT, _output_handle);
        display_message(buffer, 0,SCREEN_WIDTH, SCREEN_HEIGHT,_output_handle);
        SetConsoleTextAttribute (_output_handle, FOREGROUND_INTENSITY);
    }
}
void print_at_xy(int x, int y, char *val, HANDLE _output_handle)
{
  COORD coord;
  coord.X = x;
  coord.Y = y;
  SetConsoleCursorPosition(_output_handle, coord);
  printf("%s", (const char *)val);
  fflush(stdout);
}
```



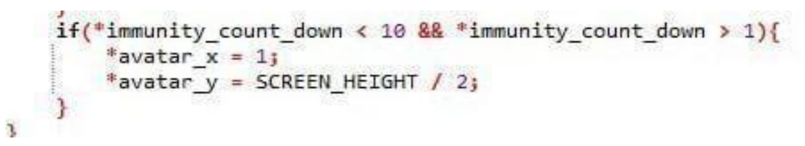
```c
 void display_score(int score, int lives){
     char buffer[50] = {0};
     sprintf(buffer, "SCORE: %4d LIVES: %d", score, lives);
     print_at_xy(0, 0, buffer);
 }
```

update.h:

Update the game state in the same declaration for holding functions responsible for updating walls and the player's avatar. Functions such as update_wall and update_avatar are declared here. That would clearly distinguish between updating game elements and other functionalities by segregation.



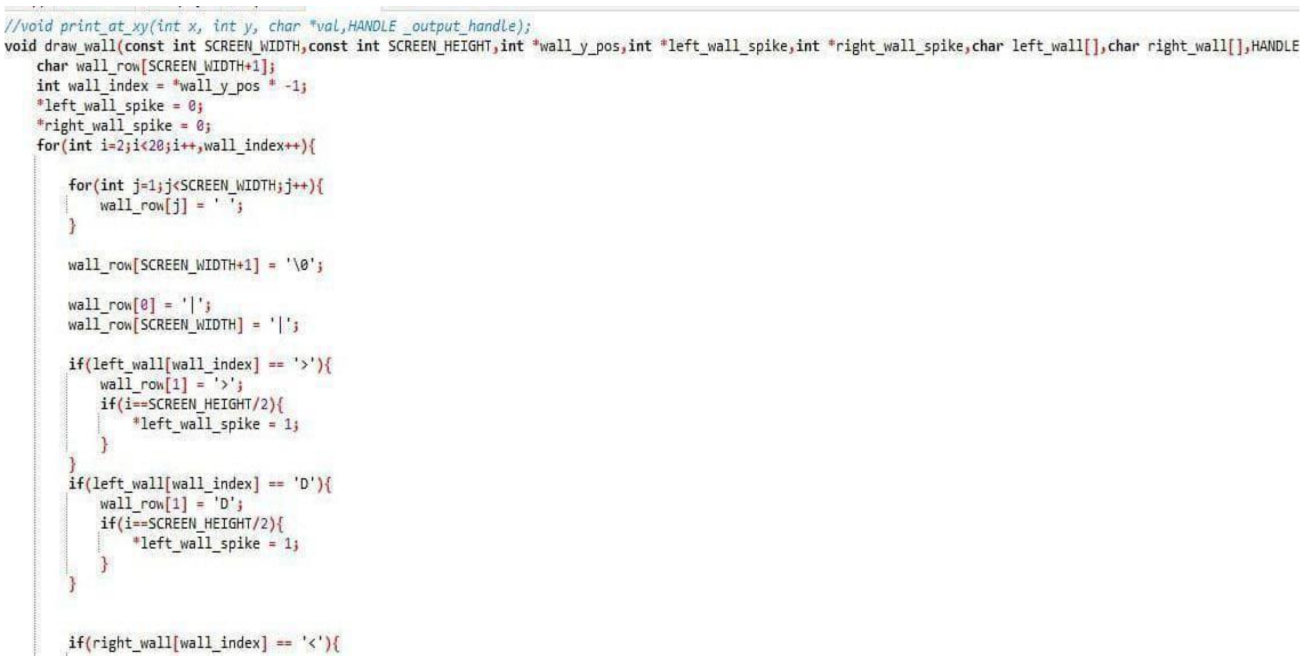
```c
void update_wall(int *wall_y_pos,int *WALL_SPEED,const int *SCREEN_HEIGHT){
    *wall_y_pos += *WALL_SPEED;
    if(*wall_y_pos > 0){
        *wall_y_pos = -*SCREEN_HEIGHT;
    }
}
void update_avatar(char ch,int *avatar_x,int *avatar_y,int *avatar_delta,int *game_state,int *GAME_STATE_PLAYING,int *avatar_SPEED,const int SCREEN_HEIGHT,const i
    *avatar_x += *avatar_delta;
    if(*avatar_x == 1 && ch == ' ' && *game_state == *GAME_STATE_PLAYING){
        *avatar_delta = *avatar_SPEED;
        *avatar_x += *avatar_delta;
        *score += *GOAL_POINTS;
    }
    else if(*avatar_x == SCREEN_WIDTH-1 && ch == ' ' && *game_state == *GAME_STATE_PLAYING){
        *avatar_delta = -*avatar_SPEED;
        *avatar_x += *avatar_delta;
        *score += *GOAL_POINTS;
    }
    else if(*avatar_x <= 1){
        *avatar_delta = 0;
        *avatar_x = 1;
    }
    else if(*avatar_x >= SCREEN_WIDTH-1){
        *avatar_delta = 0;
        *avatar_x = SCREEN_WIDTH-1;
    }

    if(*immunity_count_down > 10 && *lives < 3){
        *avatar_x = SCREEN_WIDTH/2;
        *avatar_y += 1;
        if(*avatar_y >= SCREEN_HEIGHT){
            *avatar_y = SCREEN_HEIGHT;
```



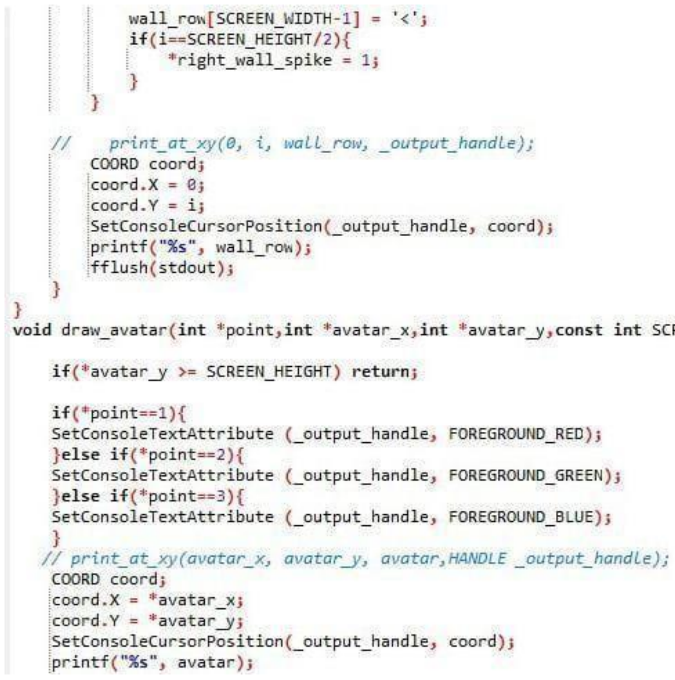
```c
    if(*immunity_count_down < 10 && *immunity_count_down > 1){
        *avatar_x = 1;
        *avatar_y = SCREEN_HEIGHT / 2;
    }
}
```

draw.h:

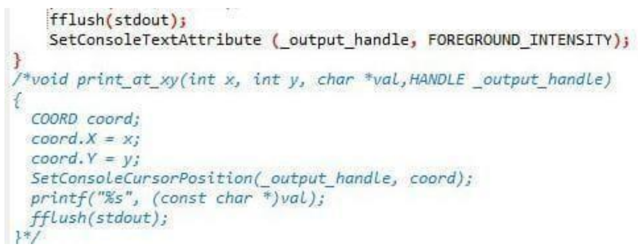
Contains declarations for functions related to drawing game elements onto the screen. It contains functions such as draw_wall, draw_avatar and draw_Sarge. This header file separates the drawing logic from other aspects of the game, hence it encourages modularity.



```c
//void print_at_xy(int x, int y, char *val,HANDLE _output_handle);
void draw_wall(const int SCREEN_WIDTH,const int SCREEN_HEIGHT,int *wall_y_pos,int *left_wall_spike,int *right_wall_spike,char left_wall[],char right_wall[],HANDLE
    char wall_row[SCREEN_WIDTH+1];
    int wall_index = *wall_y_pos * -1;
    *left_wall_spike = 0;
    *right_wall_spike = 0;
    for(int i=2;i<20;i++,wall_index++){

        for(int j=1;j<SCREEN_WIDTH;j++){
            wall_row[j] = ' ';
        }

        wall_row[SCREEN_WIDTH+1] = '\0';

        wall_row[0] = '|';
        wall_row[SCREEN_WIDTH] = '|';

        if(left_wall[wall_index] == '>'){
            wall_row[1] = '>';
            if(i==SCREEN_HEIGHT/2){
                *left_wall_spike = 1;
            }
        }
        if(left_wall[wall_index] == 'D'){
            wall_row[1] = 'D';
            if(i==SCREEN_HEIGHT/2){
                *left_wall_spike = 1;
            }
        }


        if(right_wall[wall_index] == '<'){
```



```c
            wall_row[SCREEN_WIDTH-1] = '<';
            if(i==SCREEN_HEIGHT/2){
                *right_wall_spike = 1;
            }
        }

    //     print_at_xy(0, i, wall_row, _output_handle);
        COORD coord;
        coord.X = 0;
        coord.Y = i;
        SetConsoleCursorPosition(_output_handle, coord);
        printf("%s", wall_row);
        fflush(stdout);
    }
}
void draw_avatar(int *point,int *avatar_x,int *avatar_y,const int SCREEN_HEIGHT,char avatar[],HANDLE _output_handle){

    if(*avatar_y >= SCREEN_HEIGHT) return;

    if(*point==1){
    SetConsoleTextAttribute (_output_handle, FOREGROUND_RED);
    }else if(*point==2){
    SetConsoleTextAttribute (_output_handle, FOREGROUND_GREEN);
    }else if(*point==3){
    SetConsoleTextAttribute (_output_handle, FOREGROUND_BLUE);
    }
   // print_at_xy(avatar_x, avatar_y, avatar,HANDLE _output_handle);
    COORD coord;
    coord.X = *avatar_x;
    coord.Y = *avatar_y;
    SetConsoleCursorPosition(_output_handle, coord);
    printf("%s", avatar);
```



```c
    fflush(stdout);
    SetConsoleTextAttribute (_output_handle, FOREGROUND_INTENSITY);
}
/*void print_at_xy(int x, int y, char *val,HANDLE _output_handle)
{
  COORD coord;
  coord.X = x;
  coord.Y = y;
  SetConsoleCursorPosition(_output_handle, coord);
  printf("%s", (const char *)val);
  fflush(stdout);
}*/
```

CHAPTER-5

5-CODE DESIGN

The code consists of various functions dealing with the different game aspects. Here is a general overview of the main functions applied in the code:

- **5.1- Functions Overview**
- **play_sound():**
  This function plays a sequence of beeps that have varying frequencies and durations that create a sound effect.
- **add_sound() :**
  This function calls the play_sound() function to play the sound effect.
- **player_sound():**
  This function plays a tone with a frequency of 500 Hz for 500 milliseconds, followed by a pause of 100 milliseconds, and then plays the same tone again.
- **print_at_xy(int x, int y, char *val):**
  Prints the specified character or string at the given X and Y coordinates of the console screen. It is useful to display characters, messages, or graphics at specified locations.
- **display_score():**
  It will print the score and life details on the console. This will show the player the game's status, like his/her score and other relevant information.
- **init():**
  It sets up the game with initial values in variables like score, lives, game state, wall speed, avatar position, and also spike positions. The command to hide the cursor is given here too.
- **zero_lives():**
  This function checks if the player has zero lives left and returns 1 if true, meaning the game is over.
- **set_game_state_over():**
  It sets the game state to "GAME_STATE_OVER," meaning the game is over.

  get_input():

  This function checks whether a key is pressed and returns the key when pressed.

  update_player():

  This function updates the player's position according to the received input**.**
- **update_wall():**

Updates the moving wall's position. This is necessary to shift the obstacles and give a sense of movement and challenge to the player.

- **increment_score()**
  : This function increases the player's score by the value of GOAL_POINTS**.**
- **decrement_lives():**
  This function reduces the player's lives by 1. draw
- **(int point):**
  Combines multiple drawing functions to display game elements like walls and a player's car on the console screen. This also considers the point parameter in the function, probably representing the color or type of the player's car
- **draw_wall()**
  : It calls other functions to draw the wall, avatar, and displays the score and countdown.

  draw_avatar(int point):

  Draws the player's car on the console screen. The point parameter may indicate the colour or type of the car.
- **draw_McQueen(int point):**
  Draws the McQueen car on the console screen. This function likely provides an alternative representation of the player's car, depending on the game's logic.
- **clean_up():**
  This function displays a "Thanks for playing" message.
- **clear_screen():**
  This function clears the screen by printing empty spaces in the first three rows. .
- **display_message(const char *, int yOffset):**
  Displays a message on the console screen, with options to customize the message content and vertical position (yOffset).

- **update_avatar(char ch):**
  Updates the position and behavior of the player's car based on user input (character ch). Handles logic related to car movement and collision detection.
- **update_McQueen(char ch):**
  Similar to update_avatar, updates the position and behavior of the McQueen car based on user input. Handles logic related to McQueen car movement and collision detection.
- **collides_with_spike():**
  Checks whether the player's car is colliding with spikes. This function is critical for determining if the player has encountered an obstacle.
- **display_count_down():**
  Displays a countdown message on the screen. Likely used to inform the player during a countdown sequence, adding an element of anticipation to the game.

CHAPTER-6

6-LIMITATIONS

Platform Dependency:

The game relies on Windows specific functions and libraries (windows.h, conio.h). This makes it platform dependent and may not run seamlessly on non-Windows systems.

Graphics and User Interface:

The game currently uses a console-based interface which might limit graphical elements and overall user experience. Implementing a graphical user interface (GUI) or utilizing a game development framework could improve visual appeal.

Gameplay Depth:

The gameplay appears relatively simple involving avoiding obstacles and earning points. Consider adding more features, levels or challenges to enhance the depth and longevity of the gaming experience.

CHAPTER 7:

7-CONCLUSION

So, in conclusion, we have devised the code that successfully brings the concept of a console-based car game to life, providing entertainment for its users, while maintaining a very well-organized and efficient codebase. we explored a car game code written in the C programming language. We discussed the key concepts used in the game and examined the code structure. It will let you understand the game development and how to implement game logic using C.