TRIBHUVAN UNIVERSITY

(INSTITUTE OF ENGINEERING)

THAPATHALI CAMPUS

A PROJECT REPORT ON

DINO JUMP

A course Project Submitted to the Department of Electronics and Computer Engineering as a prerequisite for the partial fulfilment of our internal evaluation for the Practical course on Computer Programming

Submitted by:

Agrima Shahi (THA080BEI004)

Samiksha Dhakal (THA080BEI038)

Shristi Mallik (THA080BEI044)

Submitted to:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

May ,2024

ACKNOWLEDGEMENT

We would like to express our sincere gratitude and appreciation to everyone who contributed, directly or indirectly, to the initiation and completion of our project. We are thankful to Thapathali Campus, IOE for providing us with the opportunity to put our knowledge and skills in C programming to the test. This challenge allowed us to come up with new ideas and methods for designing, problem-analyzing and solving in the field of game development. As a result, we collaborated on a project called "Dino Game," which greatly enhanced our learning and innovation in this area and helped with a variety of supplementary skills in coding. This project has led us to be more enthusiastic and motivated in our pursuit of research goals. We would like to express our gratitude for the exceptional dedication and contributions made by each team member, and ensure that they are all equally appreciated. We express our gratitude to Er. Saroj Shakya, our supervisor, for his valuable assistance and support throughout the project. Finally, upon the completion of our project, we would like to express our gratitude to the authors and programming libraries we referenced.

Agrima Shahi (THA080BEI004)

Samiksha Dhakal (THA080BEI038)

Shristi Mallik (THA080BEI044)

ABSTRACT

The C programming language is widely known for its speed and portability, which makes it a popular choice for game development. However, its lack of built-in support for modern game development features, such as object-oriented programming and automatic memory management, can make it more challenging compared to higher-level languages like C++ or scripting languages like Python. In this project, a simplified version of the Chrome Dino game was created using C. The code is modularized, utilizing functions that enable organization, readability, and maintenance. Conditional statements are used to regulate the game's flow, controlling collisions, user input, obstacle movement, and game-ending scenarios. Functions and conditional statements are fundamental in game development, enhancing user engagement and game behavior. The Dino game implementation provides interactive gaming experiences within a console environment, offering valuable insights into software design and game development techniques.

Keywords: game development, functions, conditional statements, interactive gaming experiences, console

Table of Contents

List of Figures

List of Table

List of Abbreviations

URL	Uniform Resource Locator
HTML	Hypertext Markup Language
T-rex	Tyrannosaurus rex
SDL	Simple DirectMedia Layer
2D	2 Dimensional
VS Code	Virtual Studio Code
PC	Personal Computer
ASCII	American Standard Code for Information Interchange
CSS	Cascading Style Sheets

1. INTRODUCTION

Welcome to the exciting world of "Dino Jump"! Imagine controlling a dinosaur that leaps through a prehistoric landscape. According to the designer of Dino game, Sebastian Gabriel, who is a Google Chrome Designer, “When you do not have Wi-Fi, then you return to the prehistoric era”. So, he introduced this game for google no-connection page, that brought this imagination to reality by the use of cacti and desert and styling it by adhering to pixel art. In this project we have recreated this game using basic C programming, making it perfect for beginners who want to explore game development. "Dino Jump" offers classic arcade-style fun with easy-to-understand gameplay. Players can guide their dinosaurs through obstacles and challenges with just a few keystrokes. It's a game that's suitable for all ages and experience levels. In this report, we'll explore the process of creating "Dino Jump," from its humble beginnings to its engaging gameplay. We'll take a look at how it was made, what makes it special, and why it's a great learning experience for those who aspire to become game developers.

1.1 Background Introduction

The Google team developed the Dinosaur Game in September 2014 to provide entertainment to users even when their internet connection was down. This simple yet enjoyable game was designed to bring joy to users. One of the appealing aspects of the Chrome Dinosaur Game is the inclusion of hidden Easter eggs and secrets. Among these, the ability to transform the T-Rex into various characters by inputting special codes in the developer console is particularly popular. Players can choose to play as a pterodactyl or a cute corgi, adding a playful and imaginative touch to the game. The widespread popularity of the Chrome Dinosaur Game highlights the appeal of simplicity and the human need for quick and easily accessible entertainment. It caters to a wide range of age groups, from avid gamers to those seeking a casual way to pass the time. The game's influence goes beyond the digital realm, with its iconic T-Rex symbolizing the imaginative nature of the internet. The Chrome Dino Game exemplifies how Google transformed a negative situation, such as losing internet connection, into a positive one by offering a fun and engaging game. This desert-dwelling dinosaur has captured the hearts of internet users worldwide.

1.2 Motivation

The Dino game was originally created by a designer at Google Chrome. It was launched on Google's no-connection page, and different game developers built versions of it using mainly HTML5, CSS3, and JavaScript. While most Dino Projects found on websites use C++ to write the code for the game, we decided to challenge ourselves by writing the source code in a completely different programming language: C Programming. Our inspiration for creating "Dino Jump" came from our shared passion for game development and our desire to explore the possibilities within the constraints of C programming. As a team, we were motivated by the challenge of crafting a compelling gaming experience using simple tools and techniques. Our goal was to demonstrate that even a simple programming language like C could be used to develop enjoyable games. To achieve this, we leveraged ASCII art and basic gameplay mechanics, showcasing the potential for innovation within limited resources. The allure of revisiting the classic arcade-style gameplay, reminiscent of childhood favorites, was also a source of motivation for us.

1.3 Problem Definition

The project aims to create an entertaining and exciting game, "Dino Jump," from scratch, utilizing the C programming language. The game's objective is to guide a dinosaur avatar across a sequence of platforms without falling. The primary goals of the project are to design simple player controls, visually appealing ASCII art for the dinosaur character and platform elements, a scoring system that rewards skillful gameplay, and improved overall game mechanics to ensure players have an immersive and enjoyable experience. The project team intends to produce an engaging game that would entertain players of all ages and skill levels and showcase the creative potential of C programming in the field of game production. By giving comprehensive thought to these factors, the team hopes to develop a game that would be engaging, entertaining, and demonstrate the creativity and expertise of the team members in game development.

1.4 Objectives

The main objectives of our project are listed below:

- To improve programming skills in C language throughout the game development journey with practical and engaging concepts.

- To foster skill development and knowledge sharing among team members, and benefit from the collaborative nature of the project.

- To write, execute and present an entire game using a C programming language using basic concepts of the C language in game development.

- To contribute to the collective knowledge base of the programming community and inspire others to pursue game development and programming as an educational resource.

- To enhance personal development, foster resilience, and cultivate problem-solving abilities beyond game development by pushing limits and enhancing capacities.

2. LITERATURE REVIEW

Now, in today's society, the landscape of gaming has evolved to encompass a diverse array of genres and platforms, catering to the varied interests and preferences of players worldwide. The resurgence of interest in retro and indie games reflects a longing for simplicity and nostalgia, coupled with a desire for engaging gameplay experiences. Examining prominent titles within this niche sheds light on the enduring appeal and evolving trends within the gaming industry.

Canabalt: Canabalt, for instance, pioneered the endless runner genre and hooked players with its minimalist design and addictive gameplay. Players control a character fleeing from an unspecified threat, jumping from building to building in a grayscale cityscape. The game's focus on reflexes and timing, coupled with its procedurally generated levels, ensures a dynamic and challenging experience with each play through.

Temple Run: Temple Run took the endless runner game to new heights with its stunning 3D environments and easy-to-use swipe-based controls. The game involves playing as an adventurer exploring treacherous temple ruins while being chased by demonic monkeys. The gameplay requires precision and quick reflexes as players use gestures for jumping, sliding, and turning. Power-ups and collectibles add depth to the gameplay, encouraging players to strategize and optimize their runs for maximum distance.

Dino Run: Dino Run offers a throwback to classic arcade gaming. The game features pixelated 2D visuals and classic side-scrolling gameplay, where players control a dinosaur sprinting through prehistoric landscapes while dodging obstacles and collecting power-ups. Dino Run offers multiple levels, varied challenges, and a retro-inspired aesthetic that evokes a sense of nostalgia while providing a fresh and engaging experience.

Jetpack Joyride: Jetpack Joyride takes players on an exciting adventure with Barry Steak fries, who uses a stolen jetpack to fly through various obstacles. The game features vibrant graphics, a range of power-ups, and an addictive gameplay loop that combines humor and excitement. Players must collect coins, complete missions, and avoid obstacles to progress through the game, making each run fresh and rewarding.

Subway Surfers: Subway Surfers is an exciting mobile game that takes place in busy subway systems all over the world. The goal of the game is to outrun a grumpy inspector and his dog while avoiding obstacles and oncoming trains. The game features vibrant and colorful graphics, dynamic gameplay, and regular updates to keep players engaged and entertained. Players can customize their characters, unlock rewards, and participate in seasonal events, ensuring that the game remains fresh and appealing to a wide audience of all ages. As a result, Subway Surfers has built a dedicated community of players from around the world.

3. METHODOLOGY AND SYSTEM DESCRIPTION

3.1 System Description

Our game is a C-based implementation designed for developing a simplified 2D version of the Chrome Dino game. It emphasizes modularity, employing functions to organize code segments for enhanced readability and easier maintenance. Utilizing conditional statements, the system regulates game flow, managing collisions, user input, obstacle movement, and game-ending scenarios. The resulting game offers interactive experiences within a console environment, showcasing fundamental game development techniques and software design principles.

3.1.1 Game Overview

Dino jump game is a simple side-scrolling obstacle avoidance game where the player controls a dinosaur that must jump over obstacles to avoid colliding with them. The player's score increases as they successfully navigate through obstacles. As the score increases, the game speed gradually increases, making it more challenging. The game ends when the player collides with an obstacle, and they have the option to replay by pressing the 'space' key. The main objective is to keep the dinosaur from colliding with the obstacles by timing its jumps effectively.

3.1.2 Development Process

3.1.2.1 Initialization

The code initializes the game environment by configuring the console window dimensions, displaying game information, initializing the positions of the dinosaur and obstacle, and prompting the user to start the game. Once the user begins by pressing the space key, the game loop starts, allowing the user to interact with the environment. These steps prepare the game for player engagement.

3.1.2.2 Character movement

The 'ds' function in the code animates the dinosaur in the game. It takes an 'jump' integer parameter for the current state of the dinosaur's movement. The function adjusts the dino's vertical position within limits to simulate jumping and falling. It uses the 'gotoxy' function to print the dino's ASCII art representation at the updated location. The 'delay' function regulates the animation speed. The 'ds' function is a crucial component of the game's immersive experience.

3.1.2.3 Obstacle Generation

In our code, obstacle generation is not implemented dynamically. Instead, the code manually places a fixed obstacle at a specific position on the screen and moves it horizontally. As the obstacle reaches the end of the screen, it is cleared, and a new obstacle is placed at the starting position, creating the illusion of continuous obstacles.

3.1.2.4 Collision Detection

Collision detection in the code compares the bounding boxes of the dinosaur and obstacle, checking for overlap to detect collisions. When a collision occurs, the game enters a "game over" state, displaying the final score and prompting the user to replay by pressing the space key. This mechanism ensures realistic gameplay interaction between the dinosaur and obstacles.

3.1.2.5 User Input Handling

User input handling in the program is facilitated by a continuous loop within the ‘main()’ function. Upon pressing the space key, the dinosaur executes a jump, while pressing 'X' terminates the game. This intuitive mechanism empowers players to interact with and control the game's dynamics, enriching their gaming experience.

3.1.2.6 Game Loop

The game loop in the program is implemented using a combination of continuous loops and conditional statements within the ‘main()’ function. Initially, the program waits for user input to start the game. Once the game begins, it enters a loop that continuously updates and renders the game state. Within this loop, the ‘ds()’ function handles the animation of the dinosaur, while the ‘obj()’ function manages obstacle generation and movement. The loop continues until the game is terminated by the user pressing the 'X' key, ensuring uninterrupted gameplay. This game loop structure facilitates the continuous rendering and updating of the game environment, providing a seamless and engaging gaming experience.

In conclusion, the game development process for this project demonstrates the successful implementation of essential game mechanics, including character movement, collision detection, and user input handling. Through careful coding and iterative refinement, a simple yet engaging 2D game experience is achieved within a console environment. This process highlights the importance of systematic design and development methodologies in creating enjoyable gaming experiences.

3.1.3 Graphics and Sound

This program does not utilize graphics or sound. It is a text-based game that relies solely on console output for visual representation and keyboard input for user interaction. Graphics, such as images for the dinosaur and obstacles, are represented using ASCII art within the console. Similarly, sound effects are not incorporated into the game. Instead, game events are communicated through text messages displayed on the console. Thus, the program focuses on providing a simple yet engaging gameplay experience without the need for graphical or auditory enhancements.

3.1.4 Testing and Debugging

Testing and debugging were pivotal in ensuring the functionality and stability of our game code. We conducted thorough manual and automated tests, identifying and addressing issues systematically. Utilizing debugging techniques like print statements and step-by-step execution, we isolated and fixed errors, ultimately enhancing the game's quality for a smoother and more enjoyable player experience

3.1.3 System Architecture

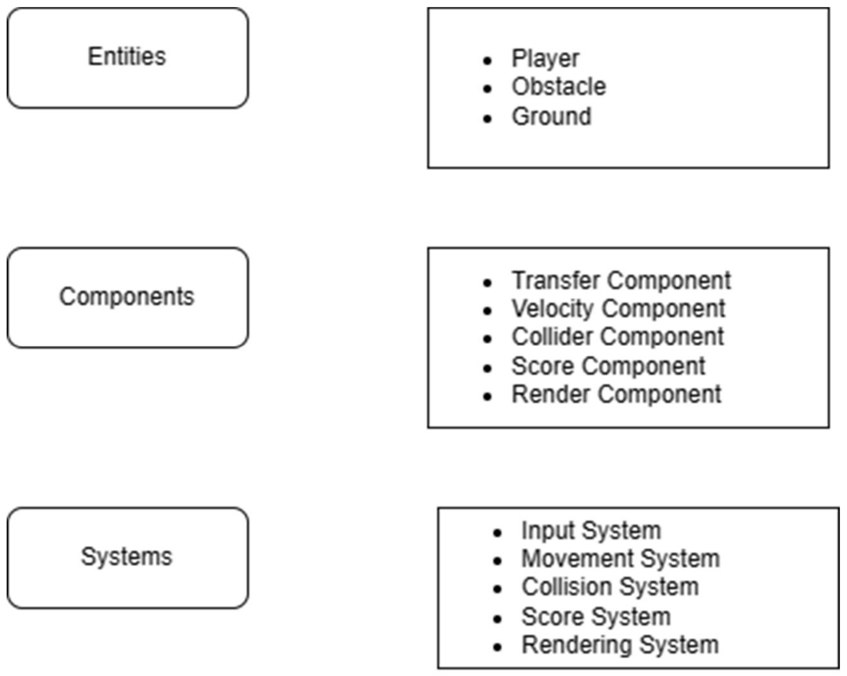


Figure 1 System Architecture

Here the Entity-Component-System Architecture is shown to bind all the assets and objects to make them work together.

3.2.1 Entities

The main entities include:

- Player (Dinosaur)
- Obstacles
- Ground

3.2.2 Components

Each entity is composed of various components that define its behavior and properties:

- Transform Component: Contains position, rotation, and scale data for entities.
- Velocity Component: Stores the velocity of entities, used for movement.
- Collider Component: Represents the collision boundaries of entities for collision detection.
- Score Component: Tracks the score for the player. Render Component: Contains information for rendering entities on the screen.

3.2.3 System

Systems process entities by iterating over their components and performing specific tasks:

- Input System: Handles user input for controlling the dinosaur's jump action.
- Movement System: Updates the position of entities based on their velocity.
- Collision System: Detects collisions between entities using their collider components.
- Score System: Manages the scoring mechanism based on game progress.
- Rendering System: Renders entities on the screen based on their render components.

3.1.4 Block Diagram

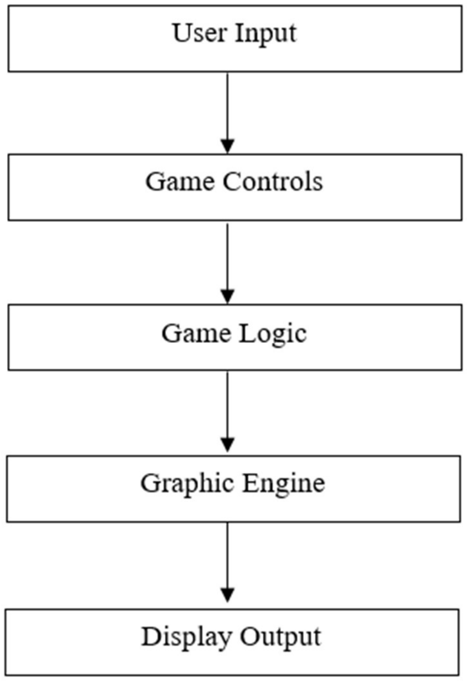


Figure 2 Block Diagram of Dino Game

3.1.5 Program Flowchart

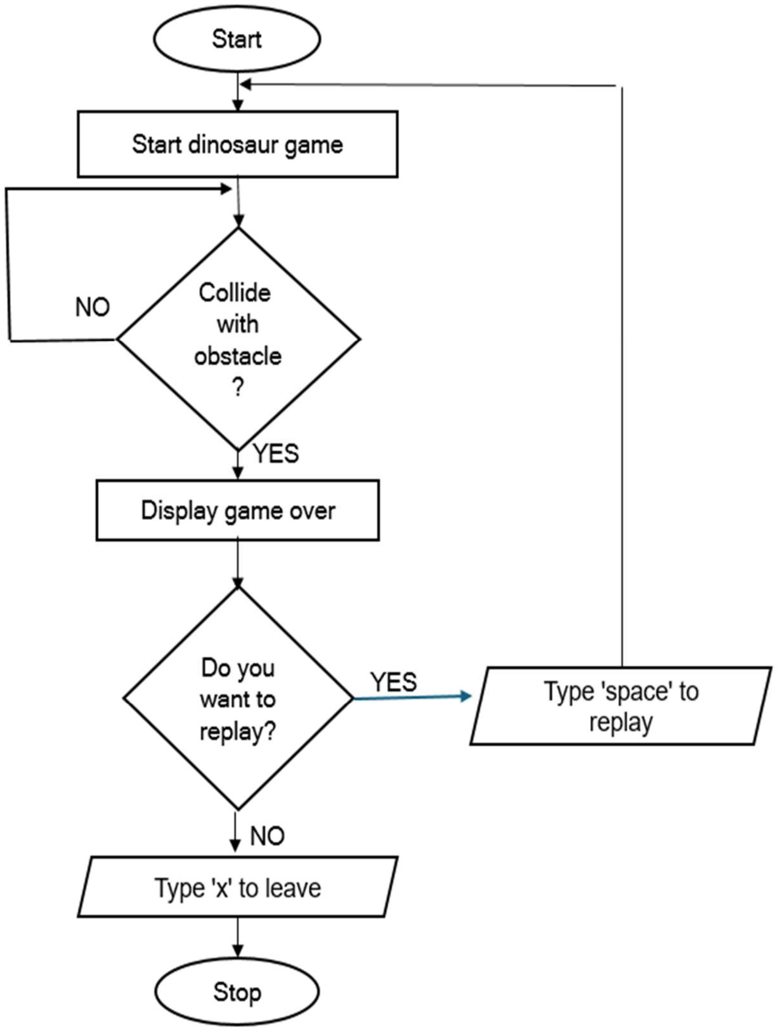


Figure 3 Flowchart of Dino Game

3.2 Methodology

The methodology in our code emphasizes efficient console-based game development through structured organization, utilizing header files for essential functionalities and functions for modularization.

3.2.1 Header Files

- <stdio.h>: This header file is used for standard input/output operations. In our code, it is primarily utilized for printing messages and formatting output to the console. For example, the printf() function is used to display messages such as "Press X to Exit, Press Space to Jump" and "Game Over!".

- <windows.h>: This header file provides functions for console manipulation in Windows environments. In our code, the SetConsoleCursorPosition() function from this header is used to set the cursor position on the console window. This function is essential for positioning text and other graphical elements within the console window accurately.

- <conio.h>: The <conio.h> header file facilitates console input and output operations. In our code, functions such as kbhit() and getch() are utilized from this header for user input handling. These functions allow the program to detect keyboard key presses and retrieve individual keystrokes from the user without echoing them to the screen, enabling interactive gameplay.

- <time.h>: This header file provides functions and data types for working with time-related information. In our code, the clock() function from this header is used to measure processor time for implementing delays in animation. By calculating the time difference between successive frames, we can control the speed of animations such as the dinosaur's movement.

3.3 Functions

The Functions are integral to the functionality of our game, aiding in the modularization of code segments for improved organization, readability, and maintainability. Each function in the codebase is dedicated to specific tasks such as initializing the game state, rendering graphics, handling user input, and updating game logic. This modular approach enhances code reuse, simplifies debugging, and facilitates future modifications or expansions of the game's features. Noteworthy functions in our code include:

- **getup():** Responsible for initializing game variables such as jump flag, obstacle position, and dinosaur position, setting up the game environment.

- **delay()**: The delay function in the code introduces a pause in program execution for a specified duration, controlling animation speed.

- **ds():** Handles the rendering of the dinosaur character on the console screen, adjusting its position based on user input and game state.

- **obj():** Manages the generation and movement of obstacles, as well as collision detection with the dinosaur, updating the game state accordingly.

- **main():** Serves as the main control loop of the game, orchestrating the execution of various functions based on user input and game state to maintain the game's flow and interaction.

3.4 Control Statements

In our game, control statements are extensively utilized to manage the game's flow, handle user input, and update game state. Here's how they are employed:

Conditional Statements (if, else if, else):

- Used for collision detection to determine if the dinosaur collides with an obstacle, triggering a game over state.

- Control the display of messages such as "Game Over!" and "Press space key to replay!" based on game conditions.
- Manage the execution of different code blocks based on the state of the game, such as handling user input and updating game variables.

Looping Statements (while, for):

- Employed in the main game loop to continuously update the game state and render frames until the game ends.
- Used to handle user input within a loop, waiting for specific keys to be pressed before proceeding.
- Control the animation of the dinosaur and movement of obstacles, iterating over frames to create smooth gameplay.

break Statement:

- Allows for the early termination of loops, such as when the player restarts the game or exits the game entirely.
- Used to exit the loop waiting for user input once a key press is detected, allowing the game to resume execution.

return Statement:

- Enables exiting functions early, such as terminating the ‘main()’ function when the player chooses to exit the game.
- These control statements collectively govern the behavior of our game, ensuring that it responds appropriately to user input, updates the game state accurately, and provides an engaging gaming experience.

4. CONCLUSION

The development of the dinosaur game presented in this report has been a challenging yet rewarding endeavor for our team of engineering students. Throughout the process, we faced various technical and design challenges, which we overcame through collaborative efforts, innovative thinking, and diligent problem-solving.

The game showcases our proficiency in programming languages such as C, as well as our ability to apply fundamental concepts of game development, including graphics rendering, user input handling, and collision detection. We implemented features such as dynamic obstacle generation, scoring mechanics, and responsive user controls to enhance the gameplay experience.

Our project highlights the importance of teamwork, time management, and continuous iteration in software development. By conducting thorough testing and incorporating user feedback, we were able to refine and improve the game's functionality and user interface.

Moving forward, we recognize the potential for further enhancements and feature additions, such as multiplayer capabilities, additional levels, and enhanced graphics. These future developments can build upon the strong foundation laid by our current implementation and contribute to a more immersive and engaging gaming experience.

Overall, the dinosaur game project demonstrates not only our technical skills and creativity but also our passion for game development and our commitment to delivering high-quality software solutions. We look forward to applying the knowledge and experience gained from this project to future endeavors in the field of software engineering.

4.1 Limitations and Enhancement

While the Dino game is a simple yet exciting game that can keep one engaged, it has several limitations.

- **Limited Gameplay Variety:** The core gameplay mechanics of the dinosaur game, such as jumping over obstacles, may become repetitive over time, leading to reduced player engagement.

- **Graphical Complexity:** Due to the simplistic nature of the game's graphics and animations, players may eventually seek more visually stimulating games with advanced graphics and effects.

- **Scalability:** The game may face scalability challenges, particularly in terms of accommodating a large number of concurrent players or expanding the game with additional features and content.

- **Device Compatibility:** Compatibility issues may arise on certain devices or platforms, affecting the game's accessibility and user experience for players using different devices.

- **Limited Social Features:** The game may lack social integration features, such as multiplayer modes, leaderboards, or social media sharing options, which could limit its ability to foster a sense of community among players.

- **Monotonous Progression:** The progression system in the game, such as obstacle generation, scoring and level advancement, may not offer sufficient depth or rewards to motivate long-term player engagement.

- **Technical Constraints:** The game's performance may be constrained by technical limitations, such as hardware capabilities, network connectivity issues, or compatibility with older software versions.

- **Lack of Storyline:** Unlike narrative-driven games, the dinosaur game may lack a compelling storyline or thematic depth, which could limit its appeal to players seeking immersive storytelling experiences.

- **Limited Replay Value:** Once players have mastered the gameplay mechanics and completed the available content, the game may offer limited replay value, leading to decreased player retention over time.

Addressing these limitations through updates, enhancements, and additional features could help broaden its appeal and longevity.

4.2 Future Enhancements

Here are some potential future enhancements for a dinosaur game that we could consider:

- **Multiplayer Mode:** Introducing a multiplayer mode where players can compete against each other in real-time or collaborate to achieve common goals, adding a social and competitive aspect to the game.

- **Additional Game Modes:** Expanding the game with different game modes such as time trials, challenge modes with specific objectives, or endless mode variations with dynamic environmental changes.

- **Power-Ups and Abilities:** Implementing power-ups and special abilities that players can collect or unlock, such as speed boosts, temporary invincibility, or abilities to smash obstacles.

- **Customization Options:** Allowing players to customize their dinosaurs with different skins, accessories, and upgrades, giving them a sense of personalization and progression.

- **Random obstacles generation and Interactive Environments:** Introducing randomness in the appearance of obstacles and interactive elements in the game environment, such as destructible obstacles, moving platforms, or weather effects that impact gameplay dynamics.

- **Storyline and Quests:** Developing a storyline with quests and missions that provide context and purpose to the player's actions, offering a more immersive and engaging experience.

- **Dynamic Difficulty Scaling:** Implementing a dynamic difficulty system that adjusts the game's challenge level based on player performance, ensuring an optimal balance of difficulty and enjoyment.

- **Achievements and Challenges:** Including achievement systems, challenges, and daily/weekly quests that reward players for completing specific objectives and encourage replay ability.

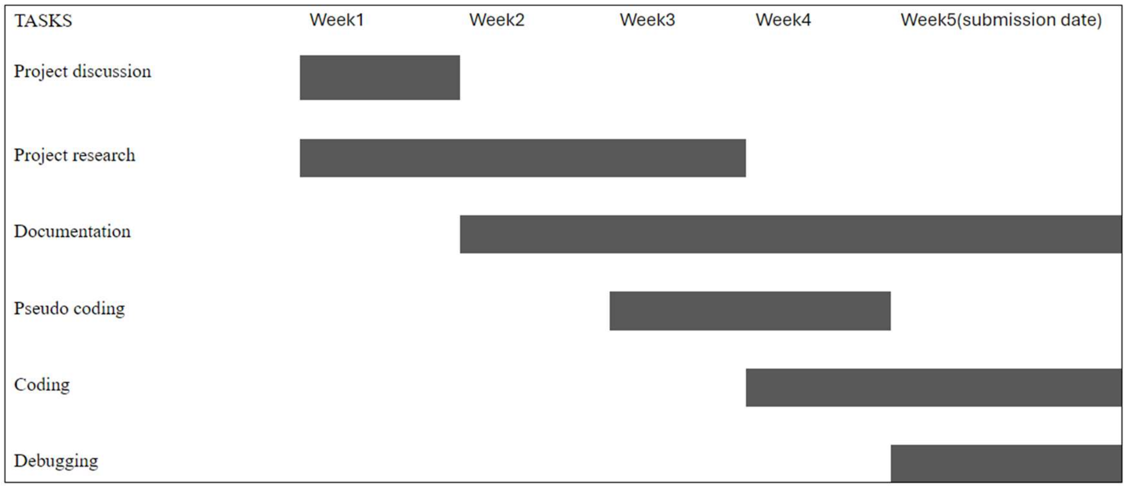
- **Social Features:** Integrating social features such as leaderboards, achievements sharing, and multiplayer challenges to foster community interaction and friendly competition among players.

- **Advanced Graphics and Effects:** Enhancing the game's visual quality with improved graphics, animations, particle effects, and environmental details to create a more immersive and visually appealing experience.

By incorporating these future enhancements, we can elevate the gameplay experience, attract new players, and retain existing players by offering fresh content, challenges, and social interactions within the dinosaur game.

5. TIME ESTIMATION

Table 6- 1 Time Estimation



5. SOURCE CODE

```c
#include <windows.h>

#include <conio.h>
#include <time.h>

int jump = 0; // Flag to indicate if the dinosaur is jumping
int t = 0;    // Vertical position of the dinosaur
int speed = 40; // Speed of the game, used in delay
int score = 0; // Variable to store the score

// Function to set cursor position
void gotoxy(int x, int y) {
    COORD coord;
    coord.X = x;
    coord.Y = y;
        SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),
coord);
}

// Function to introduce delay
void delay(unsigned int mseconds) {
    clock_t goal = mseconds + clock();
    while (goal > clock());
}

// Function to display game setup
void getup() {
    system("cls");
    gotoxy(10, 2);
    printf("                  Press Space to Jump and X to exit");
    gotoxy(63, 2);
    printf("SCORE:   %d", score);
    gotoxy(1, 25);
    for (int x = 0; x < 80; x++)
        printf("_");
}
```

```c
// Function to draw the dinosaur
void ds(int jump) {
    static const int min_height = 0;
    static const int max_height = 15; // Maximum jump height

    // Update vertical position based on jump state
    if (jump == 0)
        t = min_height;
    else if (jump == 2) {
        t = t - 1; // Descending
    }
    else {
        t = t + 1; // Ascending
    }

    // Limit the vertical position within the boundaries
    t = (t < min_height) ? min_height : t;
    t = (t > max_height) ? max_height : t;

    // Clear previous dinosaur position
    for (int i = min_height; i <= max_height; i++) {
        gotoxy(7, 17 - i);
        printf("                     \n");
    }

    // Clear the area below the dinosaur
    for (int i = 0; i < 6; i++) {
        gotoxy(7, 23 - t - i);
        printf("                    \n");
    }

    // Print dinosaur at adjusted position
    gotoxy(7, 18 - t);
    printf("            __ \n");
    gotoxy(7, 19 - t);
    printf("           / _)\n");
    gotoxy(7, 20 - t);
    printf("    .----/ /  \n");
    gotoxy(7, 21 - t);
    printf(" __/       /   \n");
    gotoxy(7, 22 - t);
    printf("|__.|_|-|_|     \n");
    if (jump != 0) {
        printf("                \n");
    }

    delay(speed); // Delay to control the speed of animation
}

// Function to draw obstacles and handle collisions
void obj() {
    static int x = 0;
    char ch;
    int gameover = 0;

    // Define bounding boxes for the dinosaur and the obstacle
    int dinoLeft = 7; // Left edge of dinosaur bounding box
    int dinoRight = 17; // Right edge of dinosaur bounding box
    int dinoTop = 18 - t; // Top edge of dinosaur bounding box
    int dinoBottom = 23 - t; // Bottom edge of dinosaur bounding
box
    int obstacleLeft = 74 - x; // Left edge of obstacle bounding
box
    int obstacleRight = 77 - x; // Right edge of obstacle bounding
box
    int obstacleTop = 20; // Top edge of obstacle bounding box
    int obstacleBottom = 23; // Bottom edge of obstacle bounding
box

    // Improved collision detection for precise touch
    if (obstacleRight >= dinoLeft && obstacleLeft <= dinoRight &&
dinoBottom >= obstacleTop && dinoTop <= obstacleBottom) {
        gameover = 1;
        gotoxy(40, 15);
        printf("Game Over!\n");
        gotoxy(40, 17);
        printf("Your Score: %d\n", score);
        gotoxy(10, 14);
```

```c
        printf("Press space key to replay!");
        while (1) {
            if (kbhit()) {
                ch = getch();
                if (ch == ' ') {
                    system("cls");
                    score = 0; // Reset score
                    x = 0;
                    t = 0;
                    jump = 0;
                    getup();
                    return;
                }
            }
        }
    }

    // Draw the obstacle
    gotoxy(74 - x, 20);
    printf("# ");
    gotoxy(74 - x, 21);
    printf("# ");
    gotoxy(74 - x, 22);
    printf("# ");
    gotoxy(74 - x, 23);
    printf("# ");
    x++;

    // If the obstacle has passed, increment the score
    if (x == 73) {
        x = 0;
        score++; // Increment score when obstacle passes

        // Display the updated score
        gotoxy(70, 2);
        printf("       "); // Clear previous score
        gotoxy(70, 2);
        printf("%d", score);
        if (speed > 25) speed--; // Increase speed gradually
    }
}

int main() {
    system("mode con: lines=29 cols=82");
    char ch;
    getup(); // Display game setup
    ds(0); // Draw the dinosaur
    obj(); // Draw the obstacle

    // Display start message and wait for space key
    gotoxy(30, 10);
    printf("Press space key to start the game...");

    while (1) {
        if (kbhit()) {
            ch = getch();
            if (ch == ' ')
                break;
        }
    }

    getup(); // Clear setup message
    while (1) {
        while (!kbhit()) {
            ds(0); // Draw dinosaur
            obj(); // Draw obstacle
        }
        ch = getch();
        if (ch == ' ') {
            // Perform jump animation
              for (int i = 0; i < 15; i++) { // Increased jump
iterations for a longer jump
                jump = 1;
                ds(1);
                obj();
            }
            // Perform descent animation
            for (int i = 0; i < 15; i++) {
                jump = 2;
                ds(2);
                obj();
            }
        } else if (ch == 'x' || ch == 'X') {
            return 0; // Exit the game if 'X' is pressed
        }
    }
    return 0;
}
```

8. REFERENCES

[1] Dinorunner, “Dinosaur T-Rex Game [Online]. Available: https://dinorunner.com/#:~:text=As%20mentioned%20above%2C%20the%20Chrome%20Dino%20Game%20is,the%20game%20is%20launched%20on%20a%20cell%20phone. [Accessed 28 April 2024]

[2] G. Bateson, “History of the Dinosaur Game” [Online]. Available: The Origins of Chrome Dino Game – From Past to Present (coolmathgames.com) [Accessed 29 March 2024]

[3] Tvtropes, “Video Games/Canabalt” [Onlime]. Available: https://tvtropes.org/pmwiki/pmwiki.php/VideoGame/Canabalt [Accessed 28 March 2024}

[4] T. Ruchandani, T. Tarihalkar, W. He and M. Shunde, “F19 T-Rex Run” [Online]. Available: http://socialledge.com/sjsu/index.php/F19:_T-Rex_Run! [Accessed 30 March 2024]

[5] Geeksforgeeks, "Dino Game in C" [Online]. Available: https://www.geeksforgeeks.org/dino-game-in-c [Accessed 30 March 2024].

[6] J. Bhatia, “6 Benefits of Using C and C++ for Game Development” [Online]. Available: https://pwskills.com/blog/c-for-game-development/#:~:text=C%20is%20a%20viable%20choice,those%20who%20prefer%20manual%20control. [Accessed 25 March 2024]