TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

Proposal

On

Dino Game using C

Submitted by:

Agrima Shahi(THA080BEI004)

Samiksha Dhakal (THA080BEI038)

Shristi Mallik (THA080BEI044)

Submitted to:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

March, 2024

ACKNOWLEDGEMENT

We express our heartfelt thanks to the Department of Electronics and Computer Engineering for presenting us with an opportunity to test our learning and knowledge through a collaborative project, namely the "Dino game." We would like to acknowledge and appreciate the contributions of each team member equally.

We would also like to express our gratitude to our supervisor, Er. Saroj Shakya, for his invaluable assistance and support throughout the project's duration. We also extend our appreciation to all individuals who were directly or indirectly involved in the development of this project.

Finally, we would like to express our gratitude to the authors of the papers and programming libraries that we referenced in our project.

Agrima Shahi (THA080BEI004)

Samiksha Dhakal (THA080BEI038)

Shristi Malik (THA080BEI044)

ABSTRACT

C is a viable choice for game development due to its speed and portability. This project is a simple implementation of the Chrome Dino game using C language. The game is designed to run in a console environment and utilizes functions to modularize the code, making it easier to organize, read, and maintain. Each function serves a specific purpose, such as initialization, rendering the game state, handling user input, and updating the game logic. This modular approach enhances the code's reusability, simplifies debugging, and enables easy modification or expansion of the game's features. Conditional statements are used extensively to control the game's flow based on various conditions, such as collision detection between the dinosaur and obstacles, user input handling, obstacle movement, and game-ending scenarios. These statements ensure that the game responds appropriately to user actions and game events, providing an interactive and immersive gaming experience. The project highlights how functions and conditional statements are essential in game development. They structure code, manage game logic, and enhance user engagement. These programming constructs simplify complex processes, control game behavior, and improve code readability, maintenance, and scalability. The implementation of the Dino game illustrates how they create interactive gaming experiences within the limitations of a console environment, providing insights into software design and game development techniques.

Keywords: Conditional Statements, Console Environment, Functions, Game Development, Implementation

Table of Contents

List of Figures

List of Tables

List of Abbreviations

URL	Uniform Resource Locator
HTML	Hypertext Markup Language
T-rex	Tyrannosaurus rex
SDL	Simple DirectMedia Layer
2D	2 Dimensional
UI	User Interface
WebGL	Web Graphics Library
IDE	Integrated Development Environment
VS Code	Virtual Studio Code
GCC	GNU Compiler Collection
RAM	Random Access Memory
OS	Operating System
HDD	Hard Disk Drive
PC	Personal Computer

1. INTRODUCTION

Chrome Dino (also referred to as T-rex Runner) is a game that appears in Google Chrome in offline mode. This can be accessed at the URL, chrome://dino/. The enemies in this game are the obstacles in the form of cacti and birds. And the objective of the player/agent is to stay safe from these obstacles. The player runs on its own and the only state of possible action is to jump or do nothing. The score increases for every step taken while the player is alive. In Chrome Dino, there is an indefinite number of states that are possible (since it is an infinite canvas of obstacles at different distances).

This game is a timeless creation by Google that has captured the hearts of millions of users worldwide. It is known for its simplistic yet addictive gameplay and charming pixel art aesthetic, making it a beloved classic in browser-based entertainment. Our goal in this endeavor was to recreate the essence of the Chrome Dino game using the C programming language. While the original game is coded in JavaScript and HTML5, we challenged ourselves to implement its core mechanics and features using the power and flexibility of C. In this document, we will explore the design and implementation specifics of our C-based Chrome Dino game, highlighting the challenges we faced, the strategies we utilized to overcome them, and the valuable lessons we gained throughout the process. Furthermore, we will offer insights into the game's architecture, the fundamental algorithms guiding its gameplay, and the user interface components that enhance the overall experience. [1]

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 whimsical 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. [2]

1.2 Motivation

The Chrome Dino game is a timeless classic that has captured the hearts of internet users worldwide. By opting for C as the programming language for this endeavor, we aimed to deepen our grasp of key programming principles and delve into the complexities of game development. As a novice programmer, the prospect of creating a game using C is not only thrilling but also immensely rewarding. Embarking on this journey has allowed us to explore a wide array of programming concepts in a hands-on manner, from data structures to graphics manipulation. The challenge of recreating the Google Dino Game in C has been a stimulating technical feat, requiring problem-solving skills and creativity to implement features like collision detection and character animation. Drawing inspiration from the original game's simplicity and allure, we aspire to capture its essence while infusing our unique flair through customization and enhancements. This project has been a testament to the versatility of C and has pushed the boundaries of what can be achieved with this venerable programming language.

1.3 Objectives

The main objectives of our project are listed below:

- To enhance programming skills, particularly in the C language, by applying concepts learned in a practical and engaging context.

- To write, execute, and present an entire game using a C programming language with the open SDL library and show the high performance and efficiency of using the C language in game development.

2. LITERATURE REVIEW

Now, games have become an integral part of human society. Society is indeed developing but along with that the desire for computer/mobile games as well. Along with that, the interest in retro/indie games can be seen increasing rapidly. The indie vibe, recreation, and casual games are desired by many people to get together and play for enjoyment. In the past, many games were developed with that in mind. There are a lot of modern games with vintage aesthetics developed for different platforms.

2.1 Canabalt

Canabalt is an endless runner game where players control a character fleeing from an unknown threat, leaping from building to building across a grayscale cityscape. The game's minimalist design and focus on reflexes and timing make it comparable to the Dino Game.[3]

2.2 Temple Run

Temple Run is a 3D endless runner game where players control an explorer fleeing from demonic monkeys through temple ruins. The game features swiping gestures to jump, slide, and turn, as well as collecting coins and power-ups to progress further. Temple Run may experience performance issues such as lag, stuttering, or crashes. These technical limitations can detract from the overall gaming experience and may frustrate players who encounter them frequently. The overall level of customization is relatively limited compared to other games.[4]

2.3 Dino Run

Dino Run is a classic pixelated 2D side-scrolling game where you control a dinosaur running through various obstacles. It features multiple levels, power-ups, and challenges to keep the gameplay engaging. Available on platforms like Steam and web browsers. Given that this game is already accessible on Chrome offline, we have the opportunity to play it on Chrome without the necessity of downloading it from the Play Store.

2.3.1 Fun

Based on the game reviews, the majority of players are satisfied with the game's features. The game keeps track of the player's score based on the distance traveled. This allows players to challenge themselves to beat their high scores or compete with friends to see who can run the farthest. The game features various obstacles that the player must dodge, including cacti, pterodactyls, and birds. These obstacles appear at random intervals and increase in speed and frequency as the game progresses, adding to the challenge. Some versions of the game have a day/night mode where the background changes from a bright daytime desert to a darker nighttime setting. This variation adds visual interest and diversity to the gameplay experience. One of the defining features of the Chrome Dino Game is its ability to be played offline, making it easily accessible to all players.

2.3.2 Challenges

Debugging C code can be challenging, especially in complex projects with multiple interconnected components. Since C is an older programming language, it does not natively support certain contemporary programming practices, such as object-oriented programming paradigms. Additionally, it is crucial to create clear and concise documentation and comments in C code to ensure readability and facilitate collaboration.

3. PROPOSED SYSTEM ARCHITECTURE

3.1 System Architecture

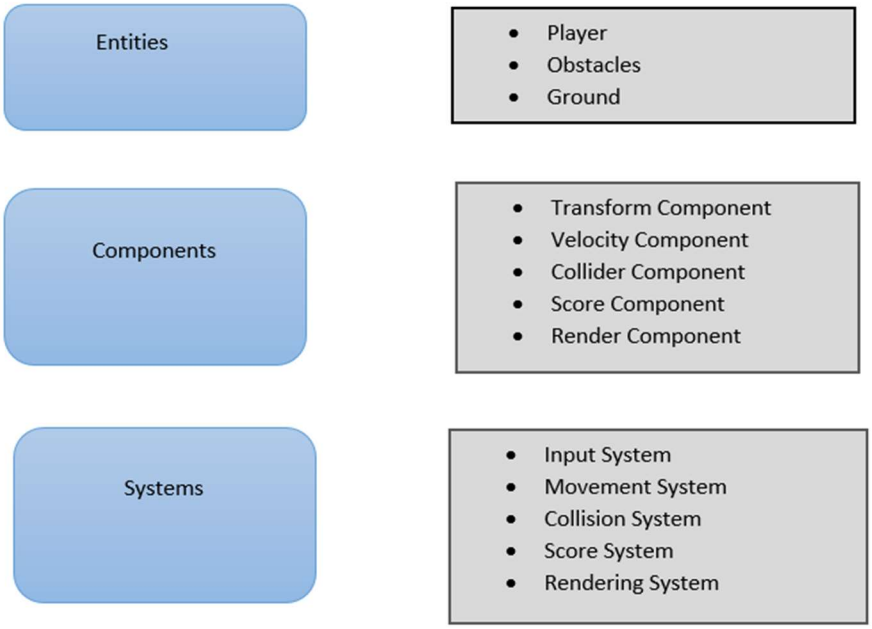

Figure 3- 1 System Architecture

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

3.1.1 Entities

The main entities include:

- Player (Dinosaur)
- Obstacles
- Ground

3.1.2 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.

3.1.3 Systems

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.

3.2 Block Diagram

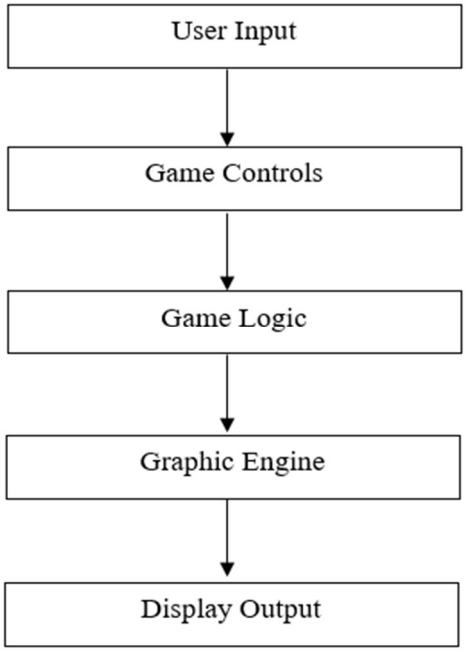

Figure 3- 2 Block Diagram of Dino Game

3.3 Program Flowchart

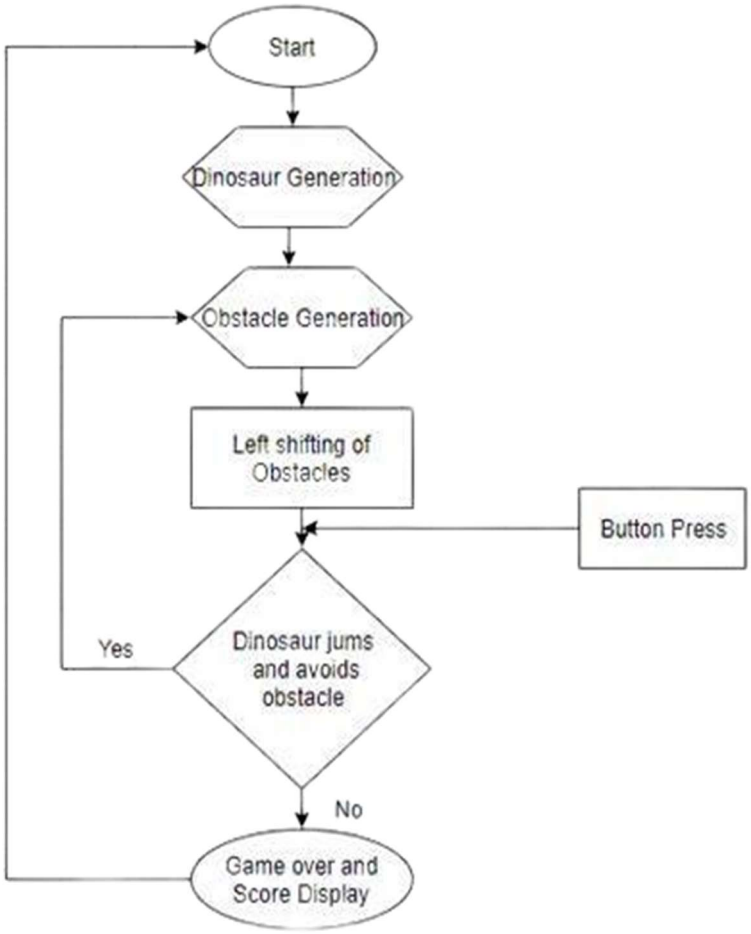

Figure 3- 3 Flowchart of Dino Game [4]

3.4 Tools and Environment

We will utilize a combination of IDEs, libraries, and tools to facilitate efficient development, debugging, and testing.

3.4.1 Language

C language

To code this game, only the C programming language is being used.

3.4.2 IDEs

VS Code

A lightweight, open-source code editor with support for C/C++ development through extensions like C/C++ for Visual Studio Code. It provides features such as syntax highlighting, code completion, debugging capabilities, and integration with version control systems.

Code Blocks

A free, open-source, cross-platform IDE specifically designed for C/C++ development. It offers a user-friendly interface, project management tools, integrated debugging, and support for multiple compilers.

3.4.3 Compiler

GCC

A widely used compiler suite for C and C++ programming languages. It is known for its portability, performance, and adherence to language standards. We will use GCC to compile and build the source code of the Chrome Dino game on various platforms, including Windows, Linux, and Mac OS.

3.4.4 Libraries

SDL

A cross-platform multimedia library that provides low-level access to audio, keyboard, mouse, and graphics hardware. SDL simplifies the process of creating games and multimedia applications by abstracting platform-specific details and providing a consistent API. We will leverage SDL for handling graphics rendering, input handling, and audio playback in the Chrome Dino game.

SDL_image

An extension library for SDL that enables loading and manipulating various image formats (e.g., PNG, JPEG) for use in SDL applications. We will use SDL_image to handle loading and displaying pixel art graphics assets in the Chrome Dino game.

SDL_ttf

Another extension library for SDL that facilitates rendering TrueType fonts in SDL applications. We will utilize SDL_ttf to display text elements, such as the player's score and game-over message, in the Chrome Dino game.

3.4.5 Graphics Editor

Aseprite

A pixel art and animation tool specifically designed for creating pixel art graphics. Aseprite offers features such as layering, animation timelines, and export options tailored for game development. We will use Aseprite to design and create pixel art assets for the Chrome Dino game, including the dinosaur character, obstacles, and background elements.

4. METHODOLOGY

4.1 Header Files

- <stdio.h>: Standard Input/output functions like printf.
- <conio.h>: Console Input/output functions, mainly used for getch() and kbhit().
- <time.h>: Provides functions to work with time, used for the delay function.
- <windows.h>: Provides functions for interacting with Windows OS, used for console manipulation. [5]

4.2 Functions

Functions play a crucial role in this game as they help to modularize the code, making it more organized, readable, and easier to maintain. Each function is responsible for handling a specific aspect of the game, such as initialization, drawing the game state, handling user input, and updating the game logic. This modular approach allows for better code reuse, simplifies debugging, and enables easier modification or expansion of the game's features. Furthermore, functions facilitate the abstraction of complex processes, which can enhance code readability and comprehension, especially in larger projects like game development. Some functions used are:

- setup(): This function initializes the game variables such as height, width, game over flag, jump flag, obstacle position, and dinosaur position.
- draw(): This function is responsible for clearing the console screen and printing the game's current state, including the dinosaur position, obstacle position, and score.
- input(): This function checks for keyboard input. If the spacebar is pressed, it sets the jump flag to 1.
- update(): This function updates the game state based on user input and game rules. It moves the dinosaur up if the jump flag is set, moves the obstacle toward the dinosaur, increments the score if the obstacle passes the dinosaur, and checks for collision conditions to determine if the game is over.

4.3 Conditional Statements

Conditional statements are an essential part of any game as they define the game's flow based on different conditions. In this game, conditional statements are used to check for collisions between the dinosaur and obstacles, manage user input to trigger actions like jumping, reset the position of obstacles when they reach the screen's end, and stop the game when specific situations occur, such as collision or going beyond the boundaries. These conditional statements are responsible for managing the game's behavior, guaranteeing that it responds appropriately to user input and game events, thereby providing an interactive and enjoyable gaming experience. Some conditional statements used are:

- if else
- if else ladder

4.4 Parts of the Program

4.4.1 Initialization

a. Initialize Game Environment:

- Set up the game canvas where all the graphics will be displayed.
- Define the dimensions of the canvas based on the screen size or a predefined resolution.
- Initialize the rendering context for the canvas (e.g., 2D or WebGL).

b. Load Game Assets:

- Load all necessary game assets such as images, sprite sheets, and audio files.
- This includes assets for the dinosaur character, obstacles, background, and any UI elements.

c. Create a Dinosaur or Some Other Character:

- Initialize the dinosaur character object with properties such as position, size, and velocity.
- Load the sprite or image for the dinosaur character.

- Set the initial position of the dinosaur on the canvas.

d. Render Initial Graphics:

- Draw the initial game scene, including the background, dinosaur character, and any UI elements.
- Display any introductory messages or instructions to the player.

e. Initialize Audio:

- Set up audio assets and load them into memory.
- Initialize audio playback functionality for game events such as jumping and collisions.

f. Handle Initialization Errors:

- Implement error handling mechanisms to handle any issues that may arise during initialization, such as asset loading failures or unsupported browser features.
- Provide feedback to the user if initialization fails, indicating that the game cannot be played.

4.4.2 Game Loop

a. Initialize game variables and structures:

- Set up variables for the player's position, score, and game over flag.'
- Define structures for the player character, obstacles, and any other game entities.

b. Start the game loop:

- Enter a loop that continues until the game-over condition is met.

4.4.3 User Input

a. Check for User Input:

- Inside the game loop, continuously monitor for user input using functions like kbhit() and getch().

b. Handle User Input:

- If a key press is detected, capture the key that was pressed.

c. Process the Input:

- Determine the action corresponding to the key press.
- Commonly, in the Chrome Dino game, the only action required from the user is to make the dinosaur jump. The spacebar key is typically used for this action.

d. Update Dinosaur State:

- Adjust the state of the dinosaur character based on the user input.
- For example, if the spacebar is pressed, modify the dinosaur's position or velocity to initiate a jump.

e. Handle Continuous Input:

- Depending on the game design, you may need to handle continuous input for actions like holding down a key for continuous jumping.
- Implement mechanisms to ensure that the game responds appropriately to continuous input without causing unintended behavior or making the game too easy.

4.4.4 Obstacle Generation

a. Define Obstacle Parameters:

- Determine the characteristics of obstacles, such as their types, sizes, and spawn intervals.

c. Spawn Initialization:

- Set up initial parameters for spawning obstacles, such as spawn position and frequency.

d. Spawn Obstacle:

- At regular intervals or based on specific conditions (e.g., time elapsed, distance traveled), generate a new obstacle.
- Determine the type and size of the obstacle to be spawned randomly or based on predefined patterns.
- Set the initial position of the obstacle off-screen to the right, ensuring it will move toward the player.

e. Move Obstacles:

- Continuously update the position of all spawned obstacles, moving them toward the player character at a constant speed.
- Once an obstacle moves off-screen to the left, remove it from the game to prevent excessive memory usage.

f. Obstacle Collision Check:

- Periodically check for collisions between the player character and obstacles to detect if the player has collided with any obstacle.

g. Adjust Spawn Rate:

- Optionally, adjust the spawn rate of obstacles based on game difficulty or player progress to increase the challenge over time.
- This could involve increasing the frequency of obstacle spawns or introducing new types of obstacles as the game progresses.

h. Repeat Process:

- Continuously repeat the obstacle generation process as long as the game is in progress, ensuring a constant stream of obstacles to challenge the player.

4.4.5 Dinosaur Jump

a. Listen for User Input:

- Continuously monitor user input to detect when the player presses the jump key. In the Chrome Dino game, this is typically the spacebar key.

b. Detect Jump Input:

- Check for the specific key press corresponding to the jump action. If the jump key is pressed, proceed to initiate the jump.

c. Initiate Jump:

- When the jump key is pressed, set the dinosaur's upward velocity to give it an initial boost upward.
- This velocity will counteract gravity and cause the dinosaur to begin ascending.

d. Apply Gravity:

- While the dinosaur is in the air, continuously apply gravity to bring it back down.
- Decrease the upward velocity gradually over time until it reaches zero, then start increasing it in the downward direction.

e. Update Dinosaur Position:

- During each frame of the game, update the position of the dinosaur based on its velocity.
- Add the velocity to the current position to move the dinosaur up or down accordingly.

f. Handle Ground Collision:

- Check if the dinosaur's position intersects with the ground level.
- If it does, set the dinosaur's position to be exactly on the ground level and reset its velocity to zero to prepare for the next jump.

4.4.6 Collision Detection

a. Define Game Objects:

- Identify the game objects that can collide with each other. In the Chrome Dino game, this typically includes the player character (dinosaur) and obstacles.

b. Determine Collision Shapes:

- Determine the shapes of the game objects for collision detection purposes. In many cases, bounding boxes or bounding rectangles are used for simplicity.

c. Calculate Object Bounds:

- Calculate the bounding box coordinates for each game object. This includes the top-left corner (x, y) coordinates and the width and height of the bounding box.

d. Check for Overlapping Bounds:

- Compare the bounding boxes of the two-game objects to check if they overlap. This can be done by checking if the x and y coordinates of one object fall within the bounds of the other object.

e. Handle Collision:

- If the bounding boxes overlap, a collision has occurred. Implement logic to handle the collision appropriately. In the Chrome Dino game, this typically involves ending the game.

4.4.7 Difficulty Progression

a. Define Difficulty Parameters:

- Identify the aspects of the game that will increase in difficulty over time. This could include the speed of the game, frequency of obstacles, or complexity of obstacles.

b. Set Initial Difficulty Level:

- Define the initial values for the difficulty parameters. These values will determine the starting difficulty level of the game.

c. Increment Difficulty Over Time:

- Implement a mechanism to gradually increase the difficulty as the game progresses. This could involve increasing the speed of the game, spawning obstacles more frequently, or introducing more complex obstacle patterns.

d. Define Difficulty Increments:

- Determine how much to increase each difficulty parameter over time. This could be a fixed increment or could vary based on the player's performance or game progression.

e. Trigger Difficulty Increases:

- Decide when and how often to trigger difficulty increases. This could be based on a timer, the player's score, or other game events.

f. Update Game Elements According to Difficulty:

- Adjust game elements such as player movement speed, obstacle speed, or obstacle spawn rate based on the current difficulty level.

g. Balance Difficulty Progression:

- Ensure that the difficulty progression is balanced to provide a challenging but fair experience for players of different skill levels.
- Communicate Difficulty Changes to Player:
- Optionally, provide feedback to the player when the difficulty increases. This could be done through visual cues, such as speeding up the game or introducing new obstacles, or through textual feedback, such as displaying a message indicating that the game has become more challenging.

4.4.8 Scorekeeping

a. Define Score Variable:

- Create a variable to store the player's score. This variable will be incremented based on the player's progress in the game.

b. Initialize Score:

- Set the initial value of the score variable to zero when the game starts.

c. Increment Score:

- Define conditions under which the player's score will increase. In the Chrome Dino game, the score typically increases as the player progresses further in the game, such as by surviving longer or covering more distance.

d. Update Score Display:

- Continuously update the display to show the player's current score. This could be done by rendering the score on the game screen and updating it whenever the score changes.

e. End of Game Score:

- Once the game is over (e.g. when the player collides with an obstacle), the final score should be displayed to the player.

f. High Score Tracking:

- Optionally, track and display the player's high score. If the current score exceeds the previous high score, update the high score accordingly and display it to the player.

Figure 4- 1 High Score Condition

4.4.9 Game Over

a. Define Game Over Conditions:

- Identify the conditions that should trigger the end of the game. In the Chrome Dino game, the primary game over condition is typically when the player character (dinosaur) collides with an obstacle.

b. Implement Collision Detection:

- Use a collision detection algorithm to check if the player character collides with any obstacles during the game. This can be done by comparing the bounding boxes of the player character and each obstacle.

c. Check for Collisions:

- Continuously check for collisions between the player character and obstacles as the game progresses. This is typically done within the game loop.

d. Handle Game Over:

- When a collision is detected between the player character and an obstacle, set the game over flag to true.
- Optionally, display a game over message or animation to inform the player that the game has ended.

e. Stop Game Loop:

- Once the game over flag is set to true, stop the game loop to prevent further gameplay updates.

f. Display Final Score:

- Optionally, display the player's final score on the screen to show their performance in the game.

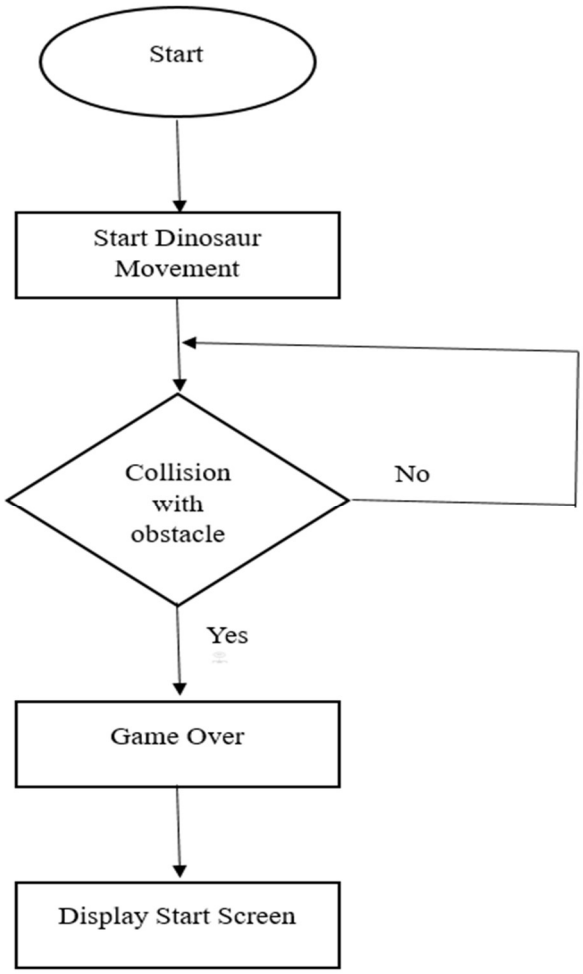

Figure 4- 2 Game Over Condition

4.4.10 Restart

a. Define Restart Function:

- Create a function that resets the game state to its initial state. This function will be called when the player chooses to restart the game.

b. Reset Game State:

- Reset all game variables, such as player position, score, and any other relevant state variables, to their initial values.

c. Clear Obstacles:

- Remove all obstacles from the game world to start with a clean slate.

d. Display Start Screen:

- Optionally, display a start screen or any other introductory message to prompt the player to start the game again.

e. Wait for Player Input:

- Wait for the player to input the command to restart the game. This could be done by listening for a specific key press or by providing an on-screen prompt.

f. Trigger Restart:

- When the player input is received (e.g., the player presses the "R" key), call the restart function to reset the game state.

g. Restart Game Loop:

- Start the game loop again to allow the player to continue playing the game from the beginning.

h. Handle Quitting:

- Provide an option for the player to quit the game instead of restarting. If the player chooses to quit, exit the game loop and end the game.

5. SCOPE AND APPLICATIONS

Computer games are today an important part of most children's leisure lives and increasingly an important part of our culture as a whole. 2D games are created to entertain players and offer them engaging challenges and enjoyment. As games have become more complex in terms of graphics, complexity, interaction, and narrative, this attempt to make a 2D game with eye-catching graphics is sure to appeal to game lovers. [6]

- This simplistic approach to creating PC games by using an open graphics library rather than a well-popular game engine provides the user with a good alternative for game development using C.
- It further opens the way for learning game development at a higher level using C for both developers and users.
- Even though 2D graphics may seem limited compared to 3D, there are many opportunities for innovation in 2D games.
- Unique art styles such as pixel art, vector graphics, or hand-drawn animations can help games to be distinctive and memorable.

6. TIME ESTIMATION

Table 6- 1 Time Estimation

Tasks	**Week 1**	**Week 2**	**Week 3**	**Week 4**	**Week 5**
Project discussion					
Documentation					
Project research					
Pseudo Coding					
Coding					
Debugging					

7. FEASIBILITY ANALYSIS

7.1 Economic Feasibility

Since this project is made using C and SDL2, there's no financial cost required for building this project. Along with that, all the software being used in the development of the project is easy to access and learn so, there is not any financial support needed.

7.2 Technical Feasibility

The game has basic 2D graphics and animations that are not demanding on system resources and can run on most devices. The game uses basic physics for jumping and gravity effects, and simple collision detection between the dinosaur and obstacles. These methods are computationally efficient. Game development frameworks and libraries streamline the game development process for games like the Dino Game by providing features such as graphics rendering, input handling, physics simulations, and audio management. We can add power-ups, environments, and better graphics to enhance the game. Also, performance can be optimized with techniques such as sprite sheets, object pooling, and efficient rendering.

7.3 Operational Feasibility

It runs on any device (Windows/Linux/mac OS) provided that the device meets the system requirement.

7.3.1 System Requirement

- 32bit / 64bit Operating system
- 1 GB or more RAM
- 256 MB or more VRAM
- 900 x 640 display
- WINDOWS OS (8 or higher) or Linux or Mac OS

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 8 March 2024]

[2] G. Bateson, "History of the Dinosaur Game" [Online]. Available: The Origins of Chrome Dino Game – From Past to Present (coolmathgames.com) [Accessed 14 March 2024]

[3] Tvtropes, "Video Games/Canabalt" [Onlime]. Available: https://tvtropes.org/pmwiki/pmwiki.php/VideoGame/Canabalt [Accessed 8 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 6 March 2024]

[5] Geeksforgeeks, "Dino Game in C" [Online]. Available: https://www.geeksforgeeks.org/dino-game-in-c [Accessed 8 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 15 March 2024]