GAME-HUB

By –

CODE CRAFTERS

CODE CRAFTERS TEAM:

1. **Harsimran Singh B2 2401030148**
2. **Mahi Gupta B2 2401030135**
3. **Shreysi Gupta B2 2401030157**
4. **Aryan Sharma B2 2401030159**

Abstract:

The Game Hub is **a console-based project** developed in **C**, featuring classic games like **Snake and Ladder, Tic-Tac-Toe**, and additional planned games such as **Math Asylum** and Dice Roll Simulator. This project emphasizes **fundamental programming** concepts including arrays, pointers, and file handling, making it an educational resource for understanding essential coding techniques. Game Hub includes options for AI opponents and **multiplayer gameplay**, providing a versatile gaming experience.

A user-friendly menu allows players to navigate between games easily, and a basic **leaderboard** tracks player scores for friendly competition. By incorporating color into the text, the project enhances the user experience without relying on complex graphical libraries, focusing on accessibility and simplicity. Avoiding the use of graphics libraries like SDL2 or even conio.h, Game Hub remains compatible across systems, serving as an excellent foundation for learning game development and data management. The design aims to give users a straightforward yet engaging experience, demonstrating how games can be created with core C language elements while leaving room for future expansions and improvements.

This documentation provides insight into the code structure, guiding developers through the logic behind each game and offering a practical platform for mastering game mechanics in C.

Concepts of SDF-1 USED:

1. **Pointers**
    - For efficient memory management and data manipulation in game mechanics.
2. **Arrays**
    - Used to store game data, such as boards (Tic-Tac-Toe) and player positions (Snake and Ladder).
3. **Structures (structs)**
    - To group related data elements, such as player details and game state.
4. **File Handling**
    - For saving and loading game data, such as leaderboard scores and player progress.
5. **Dynamic Memory Allocation**
    - To handle variable data sizes and allow for flexibility in game states.

Control Flow and Logic

6. **Conditional Statements (if-else, switch)**
    - For controlling game flow and handling user input and game rules.
7. **Loops (for, while)**
    - To iterate over game elements and run game loops effectively.
8. **Functions**
    - For modularizing code, making it reusable and organized across different games.

Game-Specific Features

9. **Random Number Generation**
    - For adding randomness in games like Dice Roll Simulator and other chance-based mechanics. Using rand() , srand().

10. **Multiplayer Logic**

- To manage data and gameplay for multiple players simultaneously.
- Use Global variables

11. **Basic AI Implementation (multiple and nested if else)**

- For providing computer opponents in games like Tic-Tac-Toe.

12. **Leaderboard System**

- To track and display player scores, enhancing replayability and competition.

User Experience Enhancements

13. **Text Coloring**

- For adding color to console text to improve user engagement and experience.

Error Handling and Validation

14. **Error Handling**

- To validate user input and handle unexpected inputs gracefully

Header files:

1. **#include <stdio.h>**

- Provides functions for input and output operations, such as printf, scanf, and file operations.

2. **#include <stdlib.h>**
    - Includes functions for memory allocation (malloc, free), random number generation (rand), and other utility functions.
3. **#include <string.h>**
    - Provides functions for manipulating C-style strings, such as strcpy, strlen, strcmp, and strcat.
4. **#include <time.h>**
    - Used for handling date and time information, particularly for generating random numbers with srand(time(NULL)) to ensure different outputs each time the program runs.
5. **#include <unistd.h>**
    - Used in UNIX-based systems for functions like sleep to pause execution for a certain duration.
6. **#include <windows.h>**
    - A Windows-specific library that includes various system functions. In **Game Hub**, it could be used for color handling in the console (SetConsoleTextAttribute) or for managing delays (Sleep).

Implementation of the Project:

MAIN BODY:

In the main menu the program begins by loading or initializing player data, such as name, balance, and stats. After initialization, the main menu is displayed, offering several game options like Snake and Ladder, Tic-Tac-Toe, Casino Hub, Rock-Paper-Scissors, Math Asylum, and an option to exit. Based on the player's choice, they are navigated to the corresponding game or feature, where they can interact and make decisions. Player progress is saved during gameplay, ensuring continuity. Once the player exits or finishes a game, their data is saved, and the program ends.

SNAKE-AND-LADDER:

The Snake and Ladder game starts by initializing the player and the game board. Players take turns rolling a dice, moving forward by the rolled number, and then checking if they land on a snake or ladder. If they land on a snake, they move back; if they land on a ladder, they advance. A variable tracks the player's position, updating it after each dice roll and adjusting for any snakes or ladders. The game checks if the player has reached the winning position after each move. Players are prompted to continue or quit after each turn. The game loop alternates turns between players, updating positions and checking for interactions with snakes, ladders, or a win condition. It continues until a player reaches the final position or chooses to quit. The game ends when a player wins or chooses to exit, with the option to play again. The board is represented by a list with special indices for snakes and ladders, and the dice roll is simulated randomly.

TIC-TAC-TOE:

The Tic-Tac-Toe game starts by initializing the game board and assigning players to two symbols (typically X and O). Players take turns selecting empty cells on the 3x3 grid to place their symbols. After each move, the game checks for a winning condition (three matching symbols in a row, column, or diagonal) or if the board is full, which would

result in a draw. If no winner is determined, the board is updated, and the game continues with the next player's turn. The game loop alternates between players until there is a winner, the board is full, or the players choose to quit. At the end of each game, players are given the option to start a new game or exit. The flow includes checks after every move to ensure the rules are followed, such as ensuring the cell is not already occupied. The game ends when a player wins, the board is filled with no winner (a draw), or the user decides to exit.

ROCK-PAPER-SCISSORS:

The Rock-Paper-Scissors program starts by displaying a welcome message and game instructions. It checks if the player wants to begin and then initializes the scores for both the player and the computer. The player selects their move (rock, paper, or scissors), while the computer randomly generates its choice. The two choices are compared to determine the winner of the round, with the appropriate score updated.

The game continues for a predefined number of rounds (10), with the results of each round displayed after comparison. Once the rounds are completed, the program announces the final winner based on the scores or handles a tie scenario if necessary. Players are then prompted to play another game or exit. If they choose to replay, the game resets; otherwise, it ends with a final thank-you message. This structure ensures smooth gameplay and proper handling of user interaction at every step.

CASINO-HUB:

The casino hub is based on credit-score system. It checks for a player's credit score and depending upon their score the leaderboard is altered. The stipulated credits decided before playing are lost if the player loses any of the game in this hub or they gain certain credits upon winning the games of the hub. It's a pure luck-based section of the gaming hub to enhance the player's gaming experience and provide thrill.

The **Casino Hub** function provides players with a variety of game options in a continuous loop, including Guessing Game, Horse Betting, High Stakes Showdown, Dice Rolling Simulator, and a Leaderboard. Players are greeted and prompted to choose an option, with secure input handling. Depending on their selection, corresponding game

functions are called, the leaderboard is displayed, or progress is saved before exiting to the main menu. Invalid inputs are handled with a retry prompt, ensuring smooth gameplay and replayability until the player chooses to exit.

NUMBER-GUESSING GAME:

The Number Guessing Game is a simple casino-style game that lets players wager credits on their ability to guess a random number between 1 and 10. Players have three attempts to guess the number, receiving hints if their guess is too high or too low. If successful, the player doubles their bet, while failure results in losing half of it. The game ensures input validity, requiring players to enter a number within the specified range and not allowing bets that exceed their available credits. The random number is generated using the current system time to ensure unpredictability. Additionally, the player's updated credit balance is saved at the end of the game, maintaining their progress. With features like colored output for better user experience and robust input validation, this implementation creates an engaging and fair gameplay experience.

HORSE BETTING GAME:

The horse Betting Game is a dynamic and visually engaging simulation of a horse race where players can bet on their chosen horse and potentially win credits. Players place a bet, select a horse, and watch as the race unfolds in real-time, with each horse progressing along a track based on randomized speeds. The visual display includes distinct colors and emojis, enhancing the experience. The chosen horse's speed is influenced by the bet amount, adding a layer of strategy and unpredictability to the game. The race concludes when a horse reaches the finish line, determining the winner. If the player’s horse wins, they earn ten times their bet; otherwise, they lose the wagered amount. The game ensures robust input validation, adjusts player credits accordingly, and saves progress at the end. With features like real-time updates, customizable speed adjustments, and visually rich feedback, this implementation delivers an exciting and interactive betting experience.

DICE SIMULATOR GAME:

The Dice Game is an interactive and engaging dice-rolling game where the player competes against the AI to achieve the highest locked score over five turns each. During

each turn, both the player and the AI roll two dice to generate a score. The player can choose to either "lock" their current score as final or roll again to aim for a higher score. The AI makes its decision based on logic, locking its score if it surpasses a threshold or exceeds the player's locked score.

The game dynamically displays the dice rolls using ASCII art and provides real-time updates about the game's status. At the end of five turns, the locked scores are compared to declare the winner. Winning rewards the player with credits, while losing deducts credits, but safeguards are in place to ensure the player's credits never drop to zero.

This game emphasizes strategy and luck, combining simple gameplay mechanics with visually appealing dice displays, and integrates seamlessly with the broader casino system by updating the player's credits and progress.

MATHS ASYLUM:

Maths Asylum is an interactive and educational math-based game designed to enhance problem-solving skills through engaging gameplay. The implementation involves generating random math challenges across various difficulty levels, covering topics like arithmetic, algebra, geometry, and puzzles. Players solve these challenges within a time limit to escape virtual "rooms" in the asylum. The game is structured with a dynamic scoring system that rewards speed and accuracy, while penalties apply for incorrect answers. Using vivid text-based or graphical interfaces, *Maths Asylum* tracks progress and adapts difficulty based on performance, ensuring a balanced mix of challenge and learning.

LEADERBOARD:

The leaderboard in *Game Hub* uses an efficient sorting algorithm to process and rank player data stored in a file. Here's how it works:

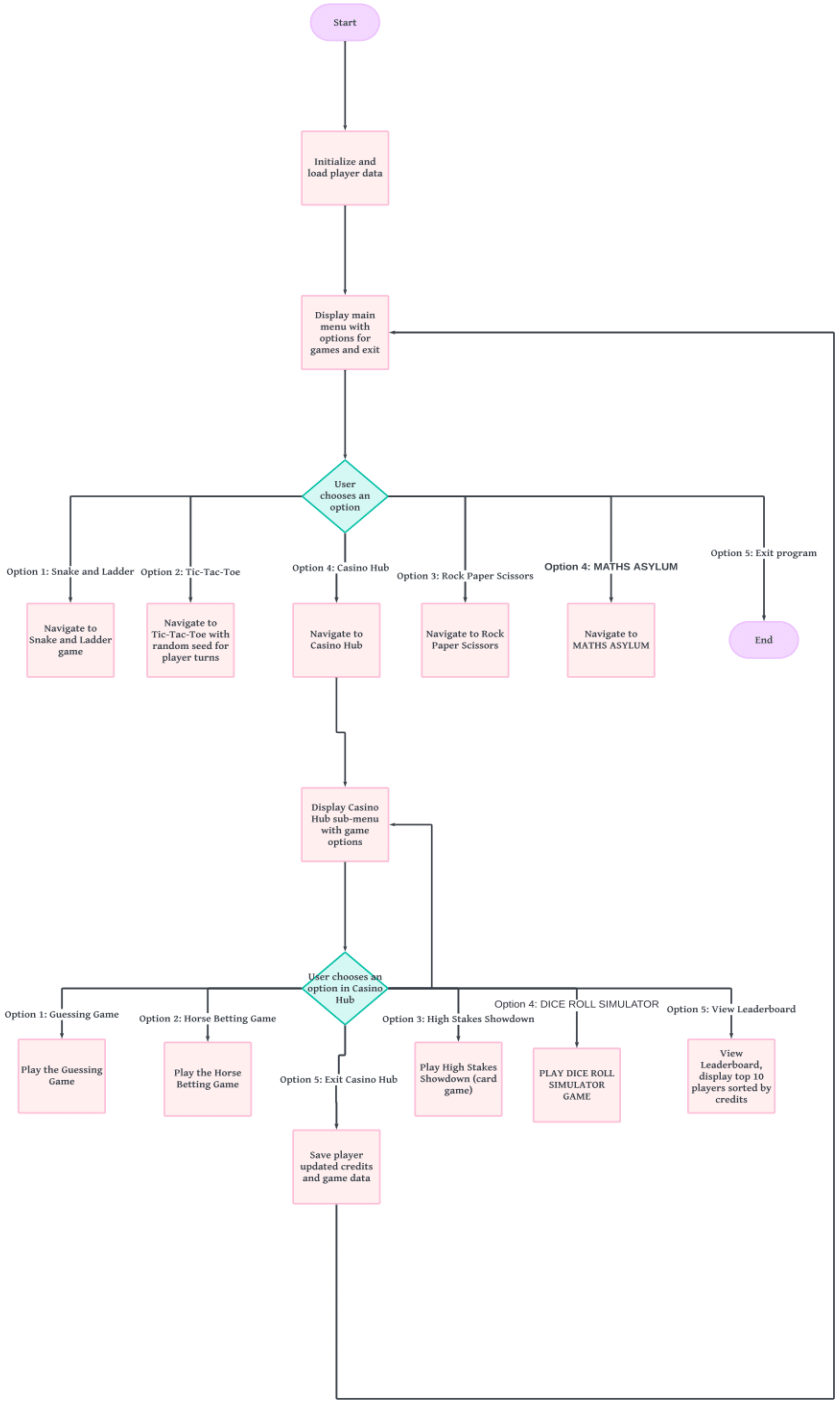
1. Data Storage: Player performance data, such as scores, game IDs, and timestamps, is stored in a structured file format like JSON, CSV, or a database table. Each entry includes unique identifiers for players and games.

2. Loading Data: When the leaderboard is accessed, the system loads the relevant file or dataset into memory. For large files, it uses optimized techniques like partial loading or indexing.
3. Sorting Algorithm: The system applies a sorting algorithm (e.g., QuickSort or MergeSort) to organize player data based on specified criteria like highest score, most achievements, or fastest times. For large datasets, it may use an external sorting technique to handle data efficiently without overloading memory.
4. Ranking Logic: After sorting, rankings are assigned by iterating through the sorted list. Ties are resolved based on secondary criteria like timestamps (e.g., earlier scores take precedence).
5. Dynamic Filtering: Filters such as game-specific, regional, or friends-only leaderboards are applied by extracting subsets of the data before or after sorting. This ensures quick and focused results.
6. Updating Files: After processing, the leaderboard updates are saved back to the file or database, maintaining consistency for future queries.

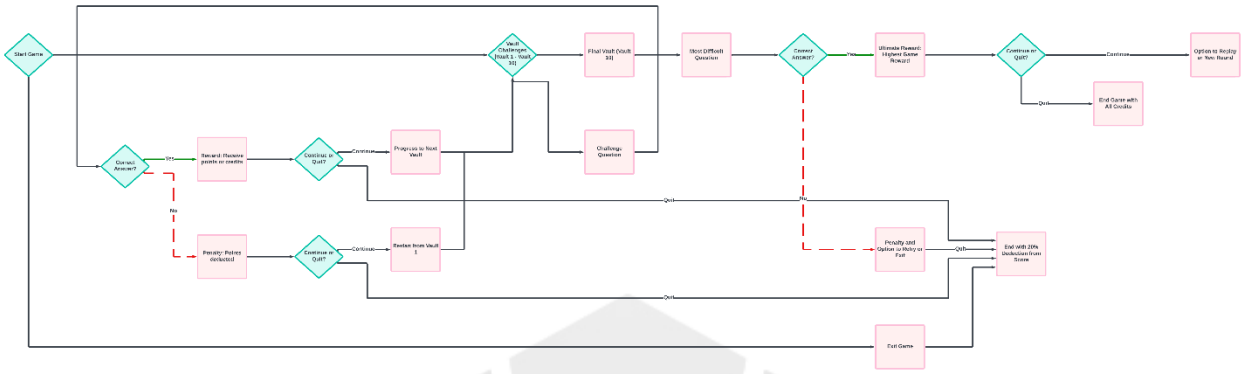
This systematic approach ensures fast, accurate, and scalable leaderboard operations even with high volumes of player data.

Design of the program:

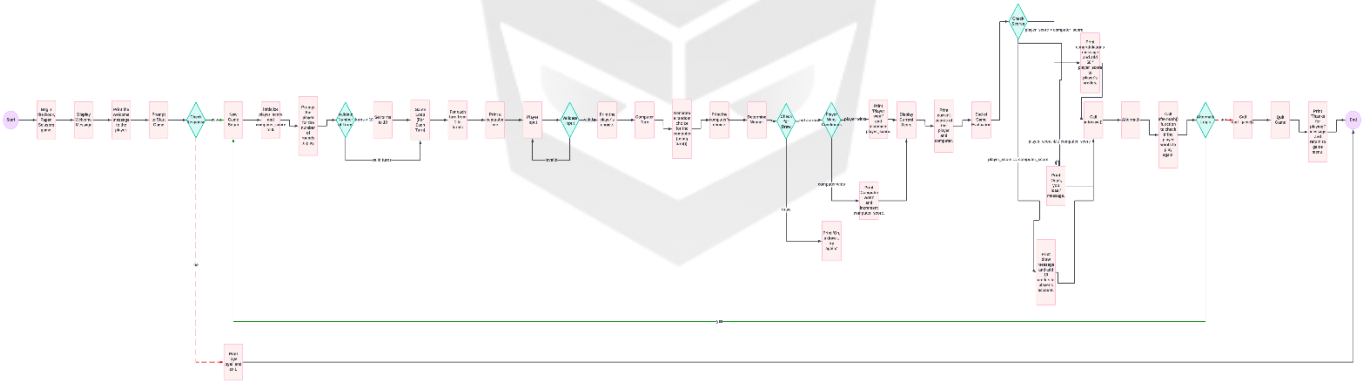
FLOWCHART FOR MAIN LOOP

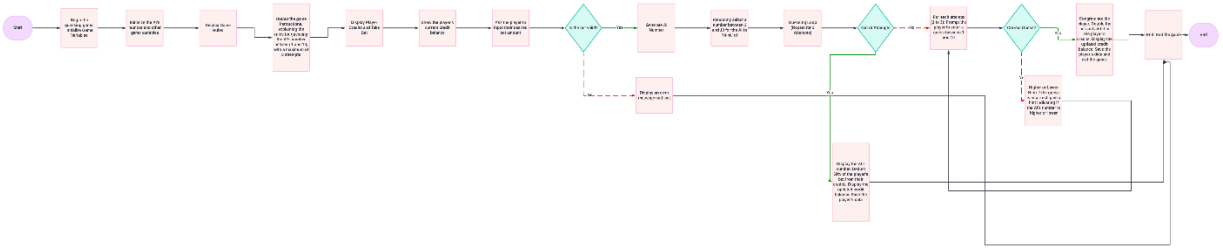


FLOWCHART FOR MATHS ASYLUM

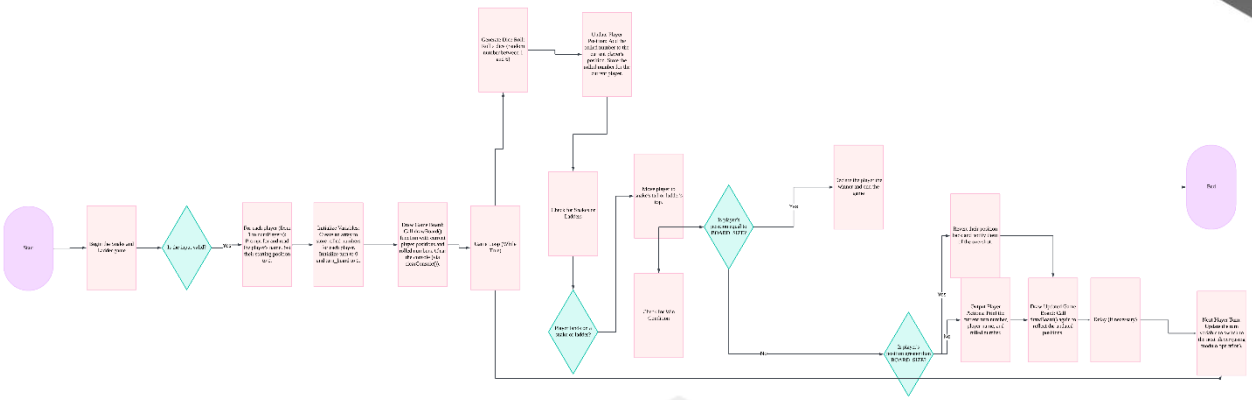


FLOWCHART FOR ROCK PAPER SCISSOR AND NUMBER GUESSING

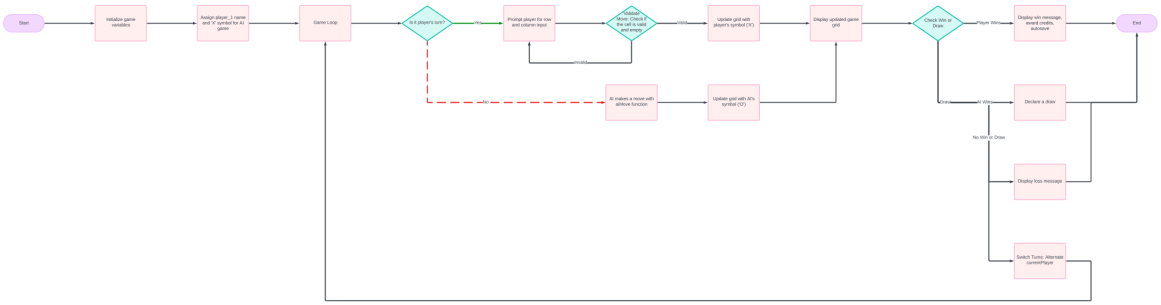




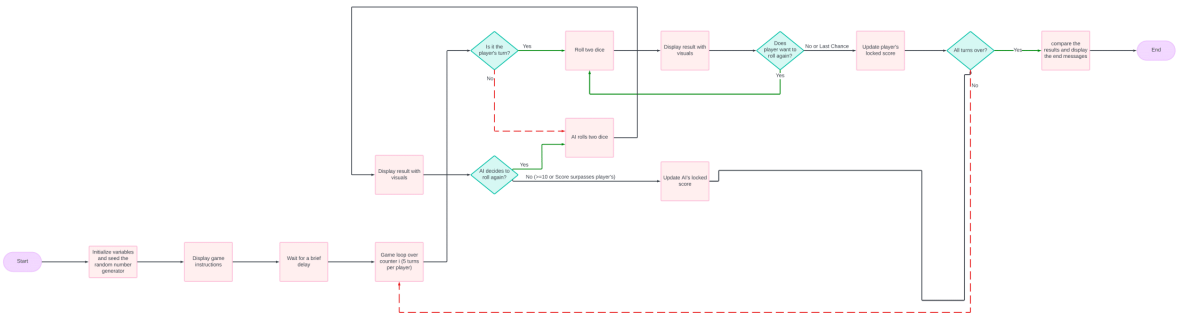
FLOWCHART FOR SNAKE AND LADDER



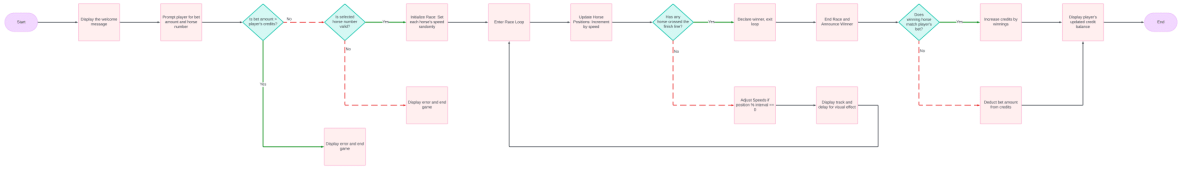
FLOWCHART FOR TICK TAK TOE



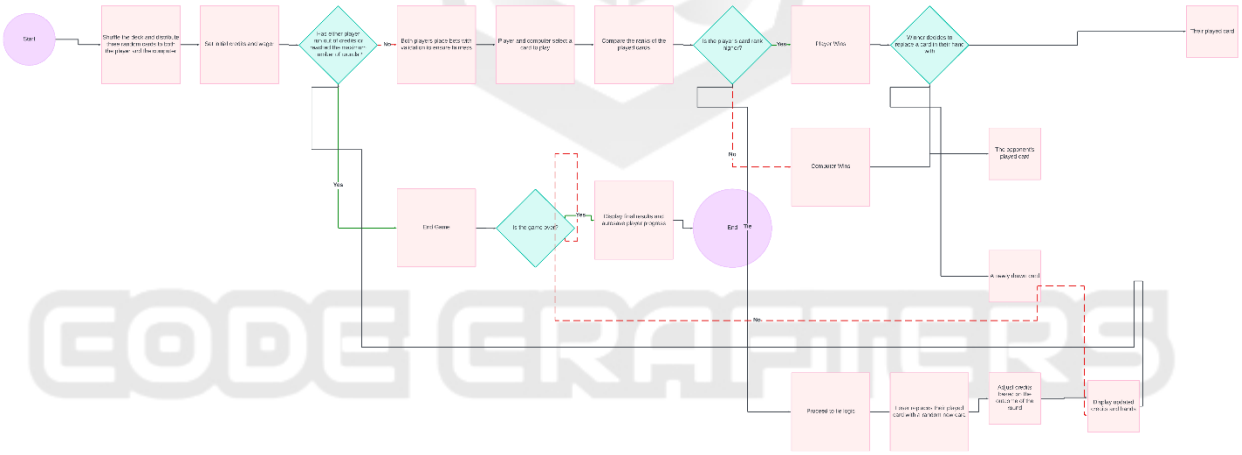
FLOWCHART FOR DICE SIMULATOR



FLOWCHART FOR HORSE BETTING GAME



FLOWCHART FOR HIGH STAKES SHOWDOWN



REFRENCES:

The implementation of this project was guided by concepts from class notes and supplemented with information from reputable online resources. The class notes provided a strong foundation in core programming techniques and algorithm design, ensuring an efficient and well-structured implementation. To further enhance the functionality, additional insights were drawn from popular online resources, including:

1. GeeksforGeeks – For understanding sorting algorithms and optimizing the leaderboard's ranking system.
2. W3Schools – As a quick reference for programming syntax and file-handling techniques during implementation.
3. Stack Overflow – For troubleshooting specific coding challenges encountered during development.
4. Python Official Documentation – For ensuring the correct usage of libraries and built-in functions.
5. Medium Articles – To explore best practices in creating dynamic and interactive gaming features.

THANK YOU