# High Suit

Final Requirement

Presented to the Faculty of the College of Engineering Education

2st Year – Bachelor of Science in Computer Engineering

University of Mindanao, Davao City

In Partial Fulfillment of the Requirements

In CPE 211/L (1749) – Data Structures And Algorithms

1nd Semester, S.Y. 2024 – 2025

**Members:**

Lee, John Andrei B. (547456)

Destajo, Christopher John  O. (546094)

Mateo, Reygian R. (54357)

## Table of Contents

## 1. Software Description

**Brief Description of the system.**

cThe game consists of two main stages: a Mini-Game Stage, where players win cards by completing challenges, and a Final Stage, where they face the house in a card battle. Victory comes by building the best deck to outsmart the house in a "best of five" card match.

**Scopes and Limitations**

The scope of the game features a unique blend of mini-games and strategic card play. Players must win mini-games to earn better cards for the final showdown against the boss. It introduces an element of chance by giving random cards upon losses, allowing for replayability and strategy.

The limitations of: the game are focused on single-player experiences with preset mini-games and deck-building mechanics. Customization of mini-games or card rules is limited, and the game's difficulty curve relies heavily on the player's mini-game performance.

**Benefits and Values(Advantages of the system)**

- Strategic Gameplay: Combines the excitement of mini-games with the strategic depth of card battles.
- Replayability: The randomness in card allocation and varying mini-games offers unique challenges in each playthrough.
- Skill and Luck Balance: Success is determined by both skill in mini-games and the luck of card draws, providing a dynamic experience.
- Engaging Mechanics: Players must balance risk and reward, as each win and loss in the mini-games affects their deck strength for the final battle.

**Platform Requirements**

Operating system Windows 10 (64-bit versions)

Memory: RAM 2 GB

## 2. Flowchart

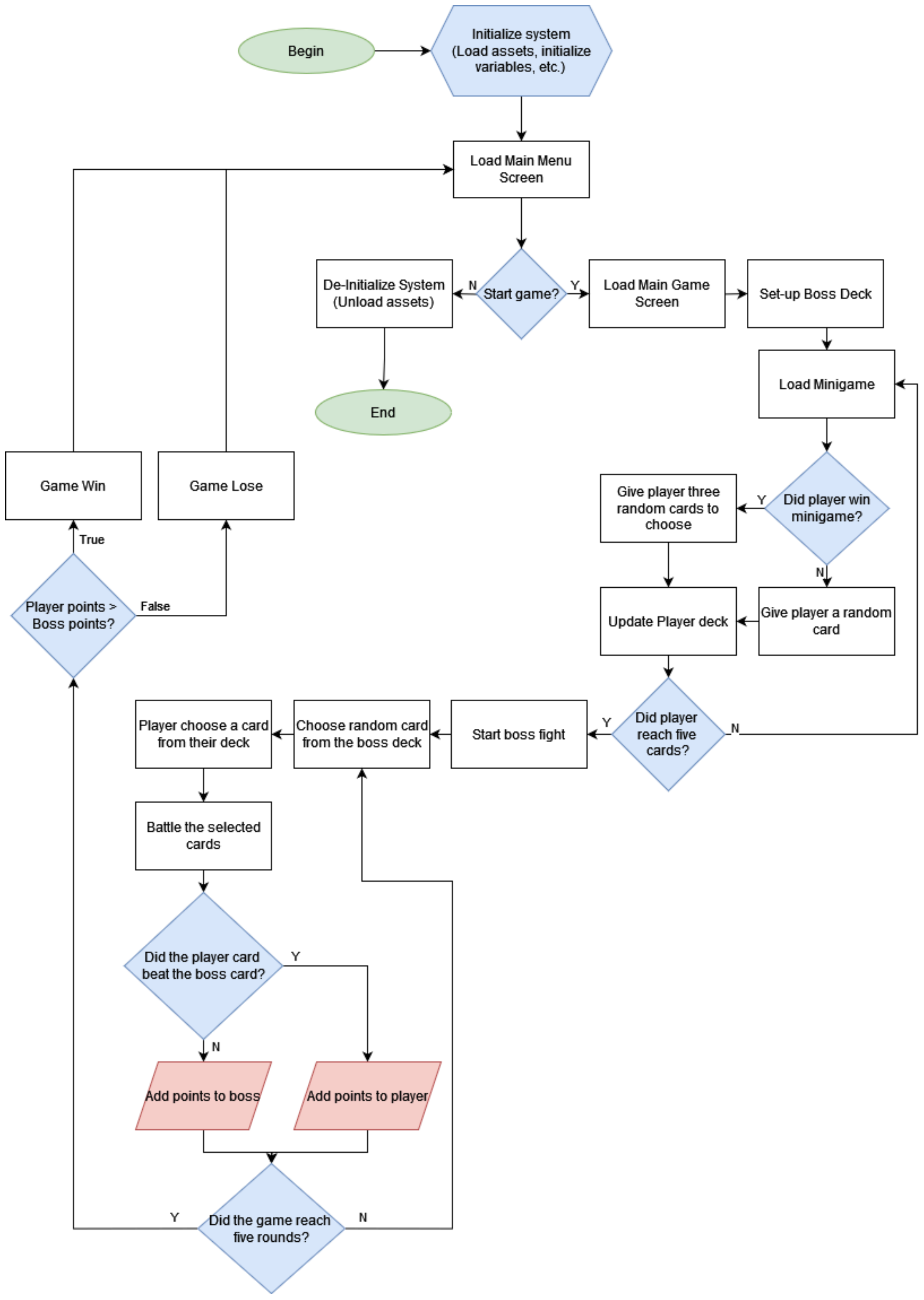
Begin
Initialize system (Load assets, initialize variables, etc.)
Load Main Menu Screen
Start game?
N
Y
De-Initialize System (Unload assets)
End
Load Main Game Screen
Set-up Boss Deck
Load Minigame
Did player win minigame?
Y
N
Give player three random cards to choose
Give player a random card
Update Player deck
Did player reach five cards?
Y
N
Start boss fight
Choose random card from the boss deck
Player choose a card from their deck
Battle the selected cards
Did the player card beat the boss card?
Y
N
Add points to boss
Add points to player
Did the game reach five rounds?
Y
N
Player points > Boss points?
True
False
Game Win
Game Lose

**Figure 2.1. High Suit General Flowchart**

## 3. File Description

CPP Files:

- **main.cpp** - The entry point of the game that initializes and runs the main game loop.
- **button.cpp** - Contains the implementation for button functionalities, including rendering and interaction.
- **deck.cpp** - Manages the deck of cards, including shuffling and dealing mechanics.
- **gameManager.cpp** - Handles the overall game logic and management of game states.
- **gameScreen.cpp** - Manages the display and functionality of the main gameplay screen.
- **mainMenuScreen.cpp** - Manages the layout and functionality of the main menu screen.
- **minigameHandler.cpp** - Coordinates the handling of different minigames and their transitions.
- **minigameScreen.cpp** - Manages the layout and functionality of the screen used for minigames.
- **transition.cpp** - Implements visual transitions between different game screens.
- **minigameArrow.cpp** - Implements the "Arrow" minigame, including rendering and gameplay logic.

Header Files:

- **deck.h** - Declares the deck class, including card handling and deck management functions.
- **game.h** - Declares global objects used throughout the game.
- **gameManager.h** - Declares the game manager class, responsible for overseeing game states and logic.
- **gameScreen.h** - Derived from screen.h. Declares the game screen class, handling the layout and display of the main gameplay area. It also handles the game update and logic.
- **globals.h** - Contains global variables and constants used throughout the game.
- **mainMenuScreen.h** - Derived from screen.h. Declares the class for the main menu, including navigation and UI.
- **minigame.h** - Declares the base minigame class that is extended by specific minigames. A pointer object of this class is used to reference the different objects derived from the class.
- **minigameHandler.h** - Declares the class for handling minigame status.
- **minigameScreen.h** - Derived from screen.h. Declares the screen class for displaying minigames.
- **reasing.h** - Extra helper file from raylib. Responsible for the easing functionality and interpolation.

- **screen.h** - Declares a base class for different game screens to unify their behavior. A pointer object of this class is used to reference the different objects derived from the class.
- **transition.h** - Declares the class for handling screen transitions and animations.
- **minigameArrow.h** - Declares the class for the Arrow minigame, including its specific logic and graphics.

## 4. Screen Output

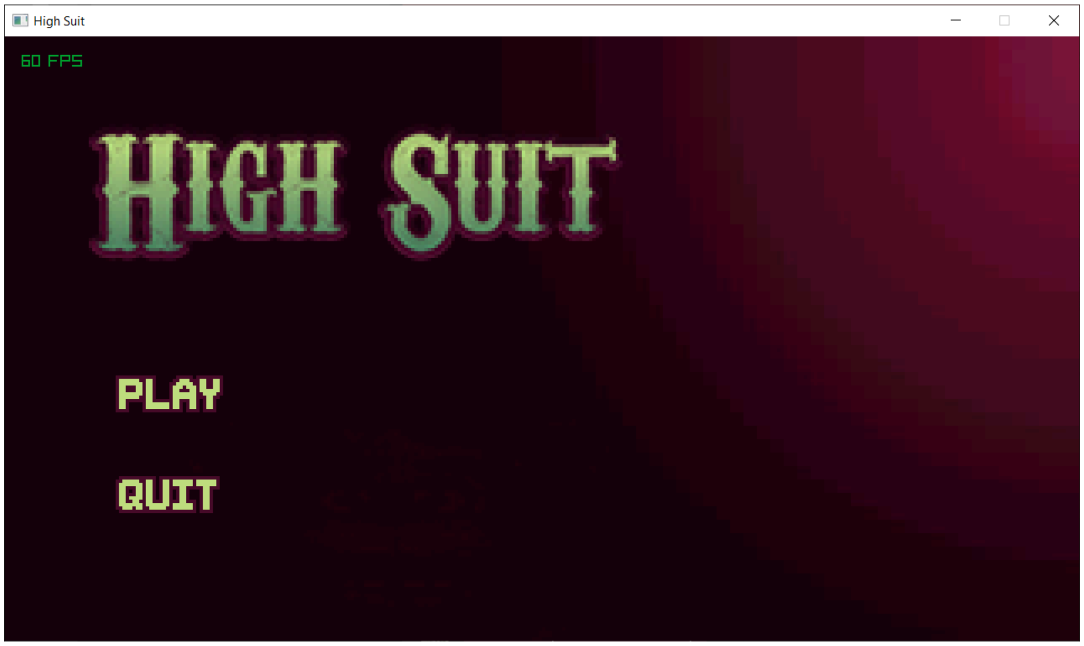

Figure 4.1. Main Menu screen

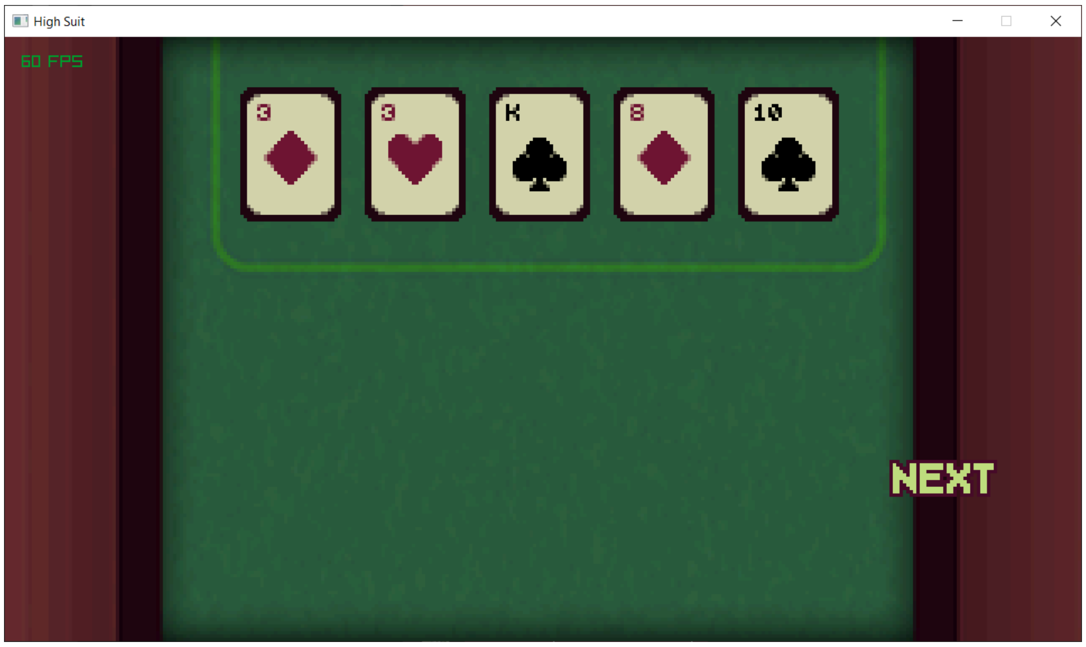

Figure 4.2. Game screen before the first minigame

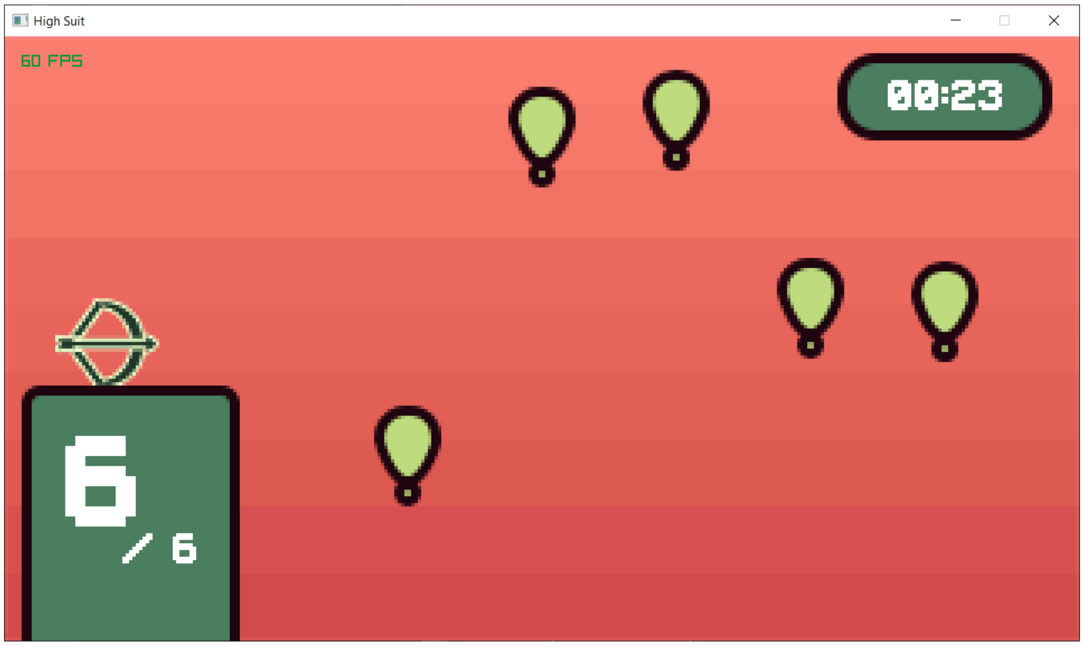

**Figure 4.3. Arrow Minigame screen**

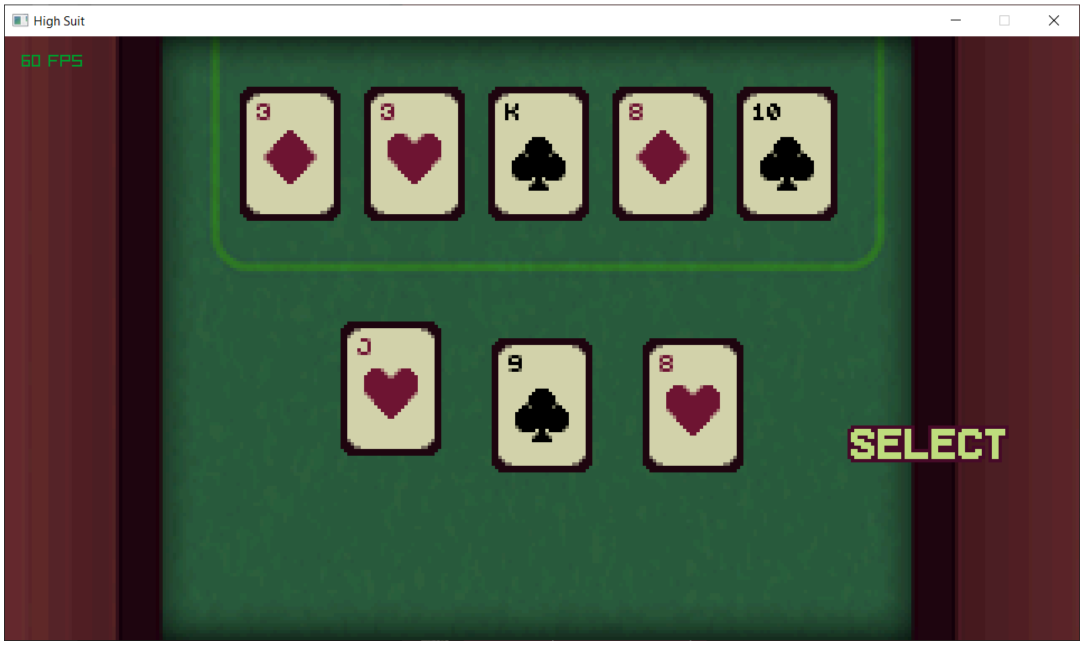

**Figure 4.4. Player is given three cards if the previous minigame is won**

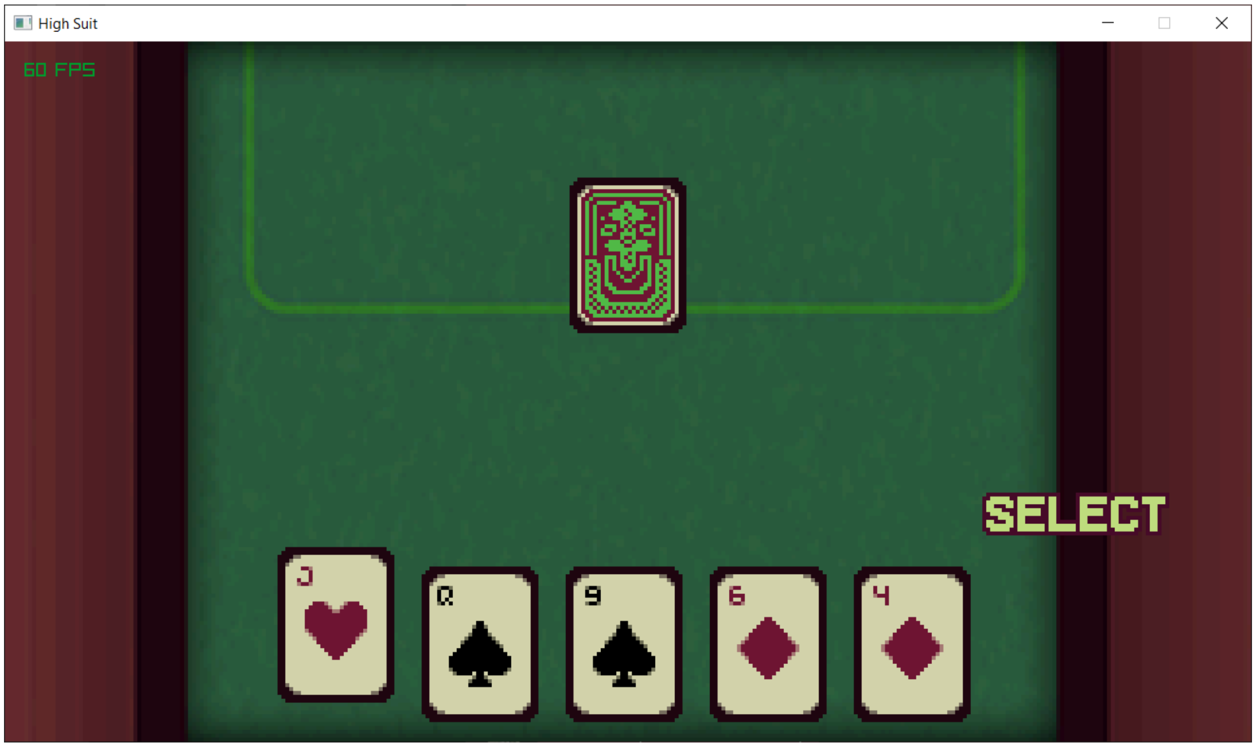

**Figure 4.5. Boss fight. Player chooses a card to battle against the bossCPE**

## References

Santamaria, R. (2013). *Reasings library. https://github.com/raylib-extras/reasings*