RMIT University Vietnam

COSC2659: iOS Development

Semester: 2023B

Assignment 2: Report

Ngo Chi Binh

s3938145

A. Introduction

- The Game I decided to create for this assignment is ‘Card War’, a relatively simple game with gameplay that relies more on luck than skill. The reason this game was chosen was the relative simplicity of the mechanics. This means that the rule is easy to understand, the game can be played using one hand, and the player does not have to devote too much attention to it.

- I chose this game because it reminded me of the simpler days of mobile gaming when they were momentary time wasters that required little attention and commitment. This is opposite to the modern-day mobile gaming landscape, where even the simplest game requires a certain amount of focus, and is more concerned with generating revenue rather than entertaining its users.

- Finally, the reason this game was chosen was due to my own limited skill. I was working on another game entirely when I realised that I didn’t have enough time to fully implement said game, and thus had to downsize to a more manageable project. My goal of creating a low-commitment mobile game remains the same, but the scale of the project has significantly decreased. Despite that, I believe that it still serves as a showcase of my current capabilities as an iOS developer, and incorporates lessons learned from the course, as well as from further research.

B. Project description

- The project consists of 5 main parent Views, each with child views that display all the features required by this assignment. The main views are:

    - The ‘Menu View’: This consists of the app’s title, 4 buttons that lead to the ‘Game View’, ‘Leaderboard View’, ‘Setting View’, and the ‘How to View’
        - At the top of the Menu View is a button that toggles the App between ‘Light Mode’ and ‘Dark Mode

    - The ‘Leaderboard View’ displays a list of all ‘Users’ registered on the App, as well as their ‘Highscore’.
        - As this is a leaderboard, the list is sorted by the user’s ‘Highscore’ in descending order.
    - Selecting one of the players the user to the ‘History View’, which displays the gameplay history of that particular player.
        - Below the gameplay history is an expandable tab that when tapped, expands into a list of ‘Achievements’ that the player has achieved.

    - ‘Achievements’ that the player has achieved appear in green, while ‘Achievements’ that the player hasn’t achieved appear in grey

- The ‘Settings View’ has 3 buttons, each corresponding to a difficulty level.
    - They are: ‘Easy’, ‘Medium’, and ‘Hard’
    - Tapping on one changes the game’s difficulty

- The ‘How to View’ consists of a brief description of the game, how to play the game, as well as an explanation of what changes each difficulty level applies to the game.

- Entering the ‘Game View’ first takes the user to a list of all ‘Users’ registered on the local version of this App. The list is arranged in alphabetical order
- At the top right-hand corner of the view is the ‘Add User’ button. Users can click this button to open up a ‘Register User View’, which allows them to register a new ‘User’
    - The new ‘User’ only requires a ‘Username’. There are some requirements: the name must not be empty, and the name must not have already been taken by another ‘Registered User’.
    - If no ‘Username’ was provided before selecting the ‘Register User’ button, or the provided ‘Username’ has been taken, an alert would appear, informing the user of the error.
    - Users can try again until the new ‘Username’ satisfies both conditions
- Users can then select a name on the list in order to start the game as that ‘User’.
- The actual ‘Game View’ consists of two cards, with the left card belonging to the ‘User’, while the right card belongs to the Cpu.
    - Below the cards are in descending order:
        - The ‘Deal’ button, which deals new cards to the ‘User’ and the ‘Cpu’
        - The ‘User’ name and their current score
        - The Cpu and its current score

- The gameplay is as follow:
    - The ‘User’ taps the ‘Deal’ button.
    - The game randomly deals the ‘User’ and the Cpu one card each.
    - The cards are playing cards with values ranging from 2 to 14 for the ‘User’, and varies for the Cpu.
        - On ‘Easy’, the Cpu’s highest card value at 10
        - On ‘Medium’, the Cpu’s highest card value at 12
        - On ‘Hard’, the Cpu’s highest card value at 14
    - If the ‘User’s card if of higher value, they score a point, and vice versa
        - Nothing happens if there’s a draw.
    - Whoever reaches the point limit first is the winner

- On ‘Easy’, the point limit is 10
- On ‘Medium’, the point limit is 25
- On ‘Hard’, the point limit is 40

C. Implementation Details

a. Software Pattern: MVVM

- This application employs the Model - View - ViewModel (MVVM) software development pattern. This is a refined version of the Model - View - Controller pattern. In MVVM, the presentation logic now all resides in the View, while the ViewController sits in between the View and Controller, transforming data in human-readable format, while also handling the update of said data

b. CRUD: UserDefault

- In order for data to persist on the app, it must be stored somewhere. UserDefault was chosen as the data storage method for this App, due to the relatively small size of the ‘User’s data.
- The data was initially loaded from JSON files and then saved to ‘UserDefault’. All subsequent initializations load data directly from ‘UserDefault’.
- This allows for new ‘Users’ to be created and saved onto the App, ‘User’s gameplay history can be stored and revisited, and new ‘Highscore’ can be recorded.

c. Sound Effects and Background Music

- All actions include a sound effect that indicates that it was selected
- Each main View has distinctive background music
- The ‘Game View’ ‘Deal’ button has a unique card shuffling sound effect
- A different sound effect will play depending on the if the ‘User’ wins or lose
- Each ‘NavigationLink’/’Button’ has a ‘.simultaneousGesture’ modifier, which executes the code that plays a sound effect on tap.
- The code to play the View’s background music is inside the ‘.onAppear’ and ‘ '.onDisappear' modifier, which plays the View’s background music whenever that View appears and stops when it disappears.

d. Features:

i. Menu View

- The ‘Menu View’ was implemented via a ‘NavigationView’ and various ‘NavigationLinks’, with each ‘NavigationLink’ taking Users to a corresponding view

- Each ‘NavigationLink’ has a custom ‘buttonStyle’ that makes them appear like buttons

ii. Leaderboard View

- The ‘Leaderboard View’ employs a List in order to display all ‘Users’ that are registered on the App.
- The ‘User’ data comes from the ‘UserViewModel’, which handles the loading of ‘User’s data, updating data, and saving any changes made to the data.

1. Highscore

- The ‘User’s data is sorted by their ‘Highscore’ by creating a sorted copy of the ‘User’s data and using the ‘sorted(by:)’ function.
- The sorted list is then put into a ‘List View’.
- Each ‘List’ item is a ‘NavigationLink’ that takes Users to the gameplay history and achievements of the selected ‘User’

2. Gameplay History

- The ‘User’s gameplay history was passed from the parent ‘Highscore View’ to this view.
- The ‘User’s gameplay history data is then displayed inside View

3. Achievements

- The data for the ‘Achievements’ comes from the ‘AchievementViewModel’, which handles the data of ‘Achievements’ stored in ‘UserDefault’
- Each ‘Achievements’ Object has an ‘achieved’ variable, which is set to ‘false’ by default
- The ‘AchievementViewMode’ has a function which checks if a ‘User’ fulfill the requirement(s) for an ‘Achievement’, and set the ‘achieved’ variable to ‘true.
- The View assigns a different background color and opacity to indicate which ‘Achievements’ the selected ‘User’ has ‘achieved’
- The ‘.onDisappear’ modifier calls a function in the ‘AchievementViewModel’, which resets the ‘achieved’ variable of all ‘Achievements’ to false whenever this ‘History View’ of a specific ‘User’ is exited.
- This process repeats for all ‘Users’ on the ‘Leaderboard View’

iii. Game View

- The first child View encountered after the ‘Play’ button is selected is the ‘Select User View’. This View also employs a List View and data from the ‘UserViewModel’ in order to populate a List of all ‘Users’ registered on this local version of the App.

1. User List

- The ‘User List View’ employs ‘List’, ‘NavigationLink’ and data from ‘UserViewModel’ in order to populate the ‘List’
- The View also has another View Model in the form of ‘GameViewModel’, which handles all data related to the game, including the current ‘Difficulty’ level.
- If there are no ‘Users’ currently registered on the device, this View will instead display the ‘NoItem View’, which displays the empty List, but with a message asking Users to create a ‘User’ before continuing.
- The ‘Add User’ button displays a View that allows users to open up the ‘Register User View’.
- When Users select a ‘User’ from the ‘List’, that specific ‘User’s data and the current ‘Difficulty’ will be passed on to the ‘GameView’, and start the game.

2. Registration

- The ‘Register User View’ is only rendered if the ‘show’ variable inside the ‘SelectUser View’ is ‘true’.
    - The ‘Register User View’ consists of a ‘textfield’ and a button.
    - When the button is clicked, ‘UserViewModel’ checks if the provided name already exists in the database.
    - If the provided name already exists, an ‘Alert’ is called, telling Users to input another name.
    - The same happens if the ‘textfield’ is empty.
    - If the name is accepted, the View calls a function from the ‘UserViewModel’ to create a new ‘User’, and ‘append’ it to the database
- The ‘Add User’ button toggles the ‘show’ variable between ‘false’ and ‘true’

3. Primary View

- All of ‘Game View’s data is handled by the ‘GameViewModel’.
- Every time the ‘Deal’ button is clicked, it calls a function from the ‘GameViewModel’ which assigns a random number between 2 and 14 to the ‘User and the Cpu.
    - Because the ‘GameViewModel’ is an ‘ObservableObject’, and all of its variables, such as:

        - ‘User’s current card,
        - Cpu’s current card,
        - ‘User’s current score,
        - Cpu’s current score,
        - …etc
    - … are all ‘@Published’ variables, any changes to those variable is reflected in the ‘Game View’
    - The functions inside ‘GameViewModel’ simply mutate its variables, and the changes are reflected in Views that reference those variables
- Functions inside the ‘GameViewModel’ determine the maximum score and the range of the Cpu’s card depending on the ‘Difficulty’ variable.

4. Game Over

- Once either the ‘User’ or the Cpu reaches the maximum score determined by the ‘Difficulty’ variable, a ‘GameOver View’ is rendered.
- The ‘Game View’ has variables like:
    - The Boolean ‘playerWins’ variable, which becomes ‘true’ if the ‘User’ reaches the maximum score first. This helps the ‘GameOver View’ determine who won.
    - The Boolean ‘gameOver’ variable which references the ‘gameOver’ variable from ‘GameViewMode’, which determines if anyone has reached the maximum score
    - The ‘finalScore’ variable, which records the maximum score achieved when the game is over. This is set to the ‘User’s score if they won, or zero if they lost
    - When the variable ‘gameOver’ is set to ‘true’, the ‘GameOver View’ is rendered.
    - Inside the View is a message that changes depending on whether the ‘User’ won or lost their final score, and a button that takes the ‘User’ back to the ‘Menu View’

    a. Win

    - If the ‘User’ won, the View displays a congratulation, the final score and depending on the ‘Difficulty’
    - Execute a function from the ‘UserViewMode’ which update current ‘User’s data
        - Increment the number of games the ‘User’ played
        - Increment the number of wins depending on the difficulty
        - Increment the number of total wins

- Update the ‘User’s ‘Highscore’ if the current score is higher than the stored ‘Highscore’
- Calculate the ‘User’s current win ratio

b. Lost

- If the ‘User’ lost, the View displays ‘Game Over’, and updates the ‘User’s data.
- The update function from the ‘UserViewModel’ updates the following:
  - Increment the number of games the ‘User’ played
  - Recalculate the ‘User’s win ratio

iv. Game Settings View

- The ‘Difficulty’ Model is an ‘enum that maps each level of ‘Difficulty’ to a number. Each number represents the maximum score for the game.
- The ‘Settings’ View Model handles the ‘Difficulty’ data. The ‘environmentObject’ property wrapper allows any View to access the ‘Difficulty’ variable..

1. Difficulty

- ForEach(Difficulty.allCases, id: \.self) { level in
- Button(action: {
- userSettings.difficulty = level
- }, label: {
- … create a column of 3 buttons, one for each level of ‘Difficulty’. Selecting a button sets the ‘Published’ ‘difficulty’ variable inside the ‘Setting’ View Model to the corresponding ‘Difficulty’.

v. How to Play View

- The ‘How to View’ uses ‘List’ and ‘Section’ in order to create a guide for how to use the App, how to play the game, and how each ‘Difficulty’ level changes the game.

e. Advanced Feature(s)

i. Completed: ‘Dark Mode’

- The App uses the ‘colorScheme’ environment value in order to affects the App’s color scheme
- The App sets the default color scheme to ‘.dark’ using a ‘State’ variable, and function that changes the color scheme to ‘.light’ if it detected the current color scheme as ‘.dark’ and vice versa

ii. Incomplete:

1. Multiple Levels
2. Save and Resume Game

3. Localization

D. Conclusion

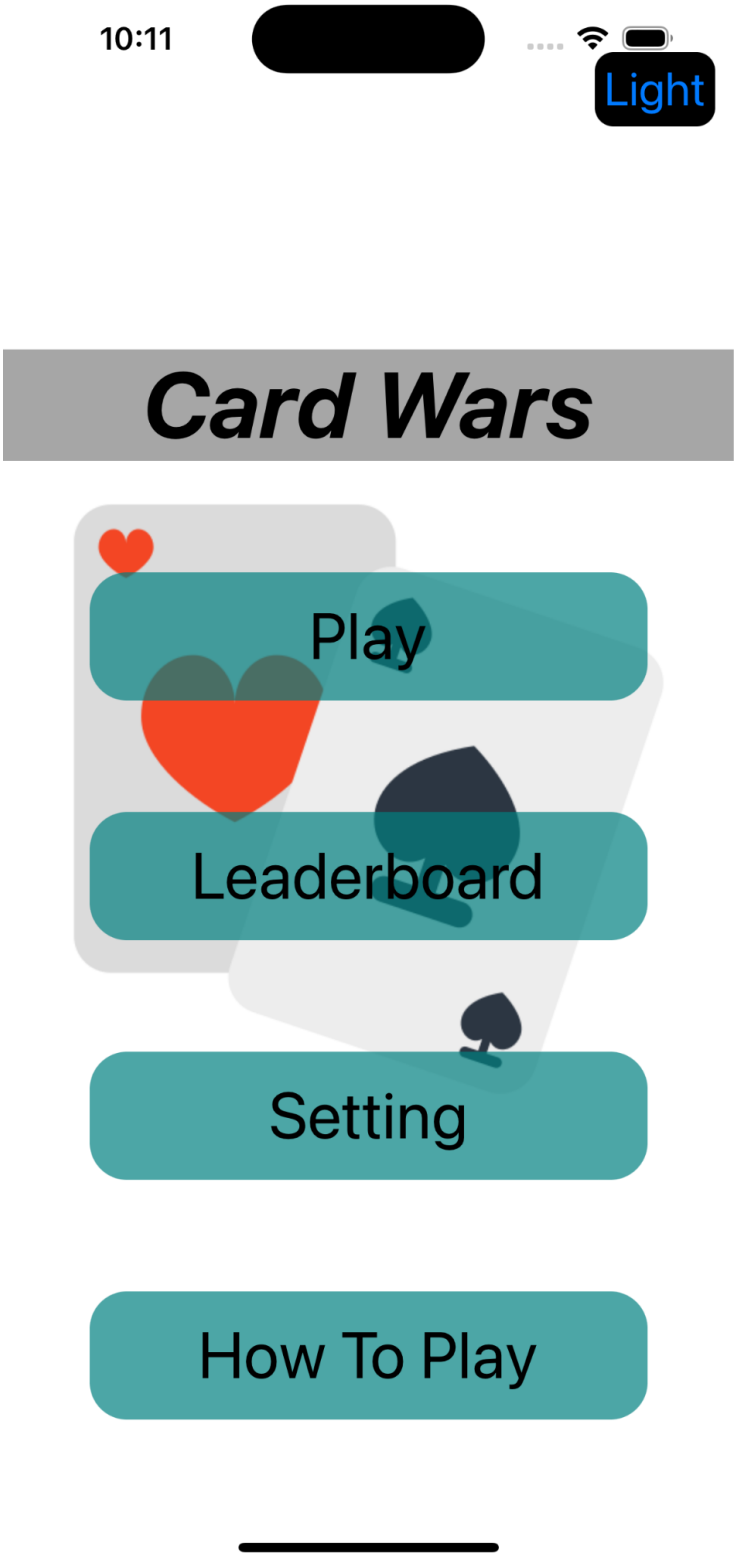
- All basic features of the App have been implemented and tested, with a few minor bugs that do not affect the overall operation of the App. However, the overall UI and UX design elements of the App required improvements. The game that was created for the App is overly simplistic, with the bare minimum level of interactivity from the user. Of the basic features, only the 'Dark Mode' features have been completed.

- Overall, while I believe that I have used every aspect of SwiftUI that has been covered by the Course, I also did the bare minimum, especially when it came to the design of the game itself, and the implementation of advanced features. This is due to my less-than-stellar time management and prioritisation skills, as I believed I spent far too much time working on the App's core functionality before moving on to the Game. This will be a valuable learning experience for me when it comes to managing software development projects as a whole, and iOS development projects in particular.

E. Appendix

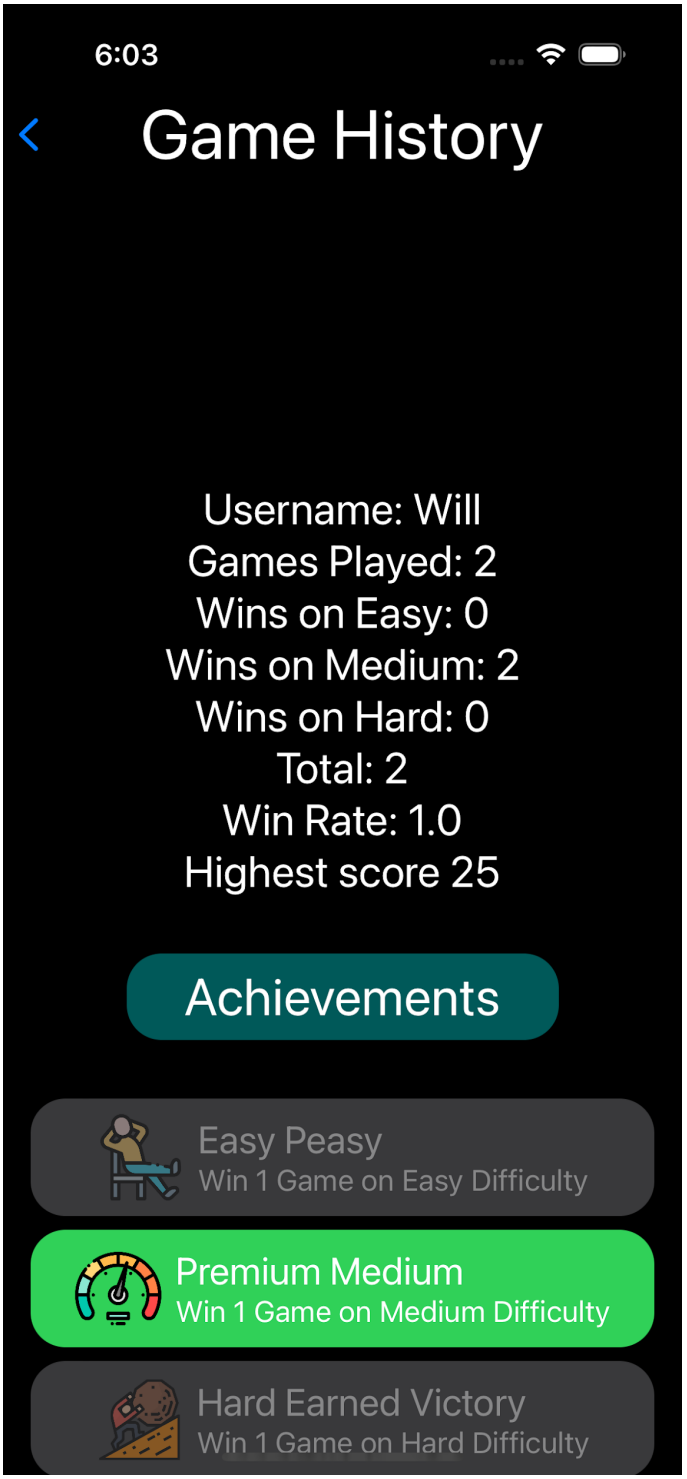
- Video Link: https://youtu.be/bq3dcWVFWGk

Main Menu - Dark Mode

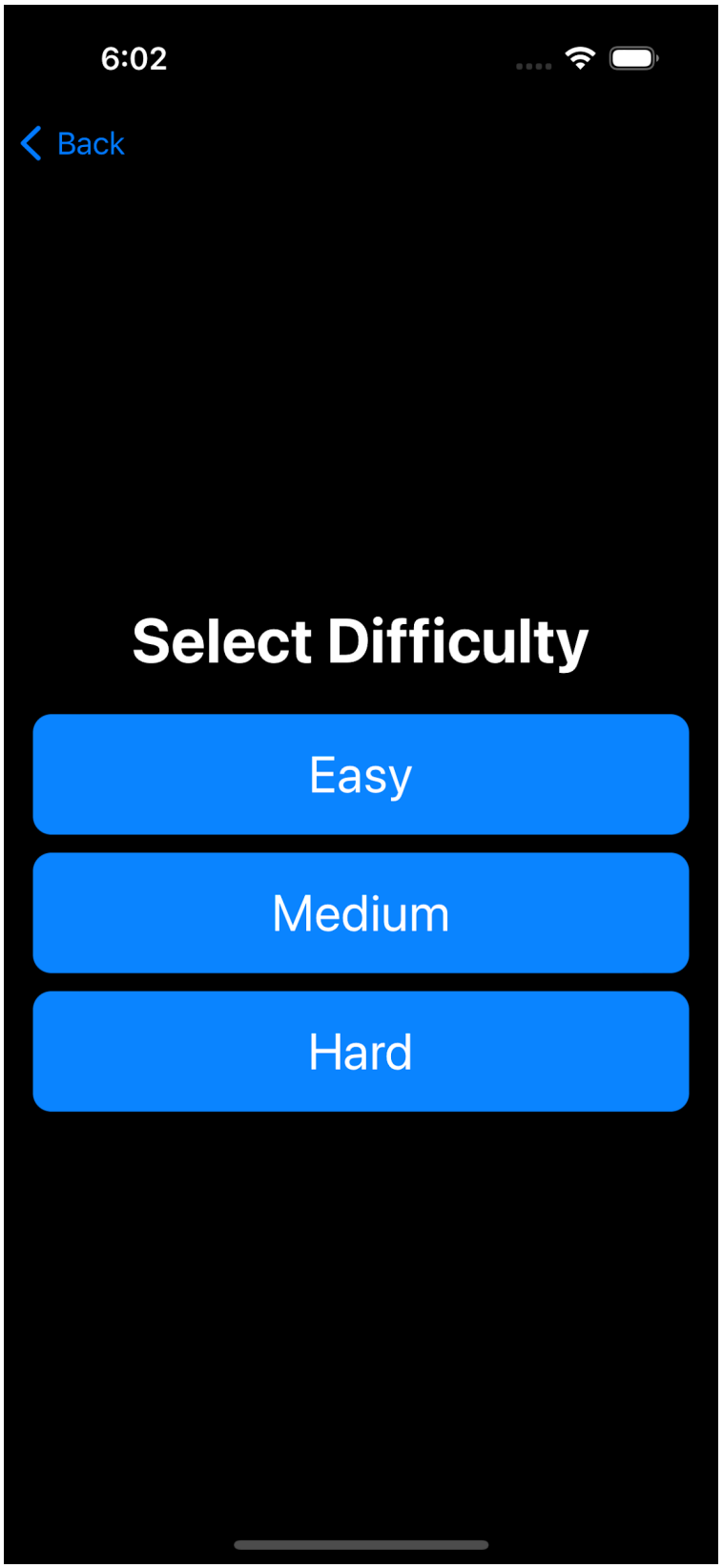


Main Menu - Light Mode

Leaderboard

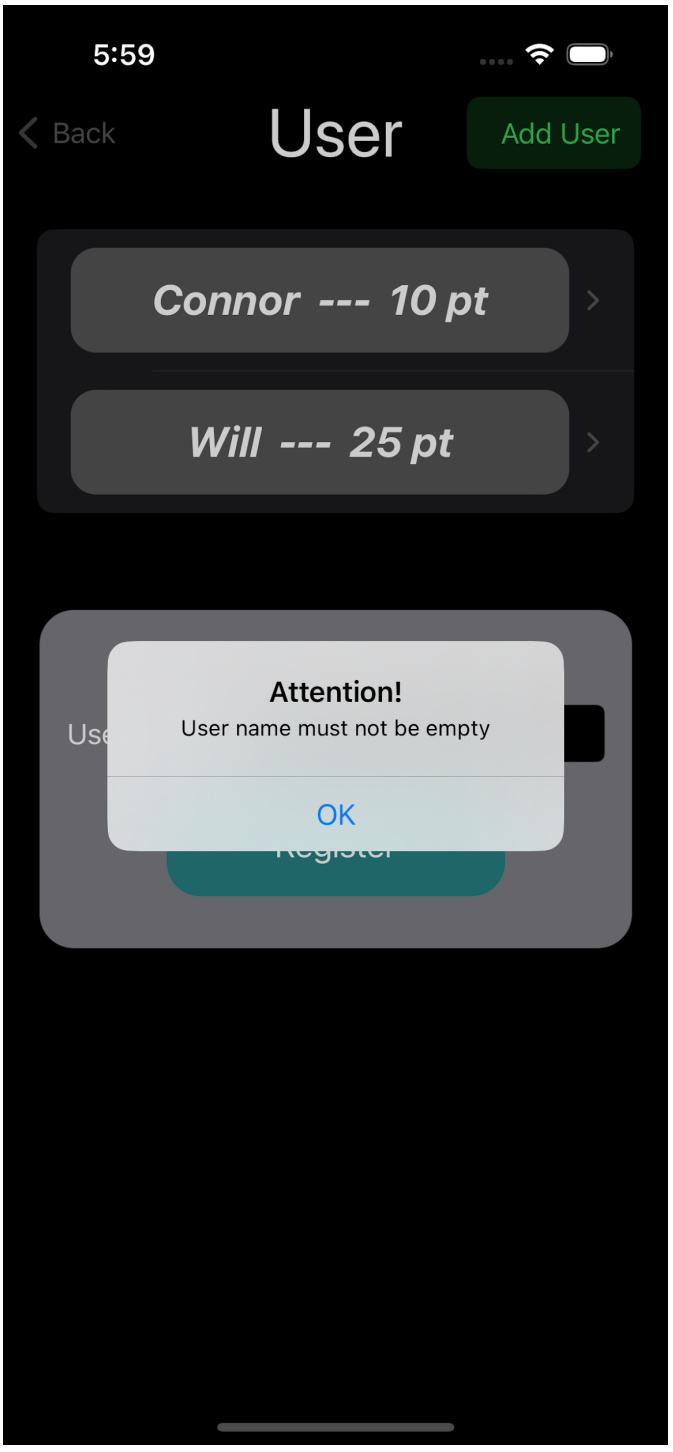


Leaderboard - History - Achievement

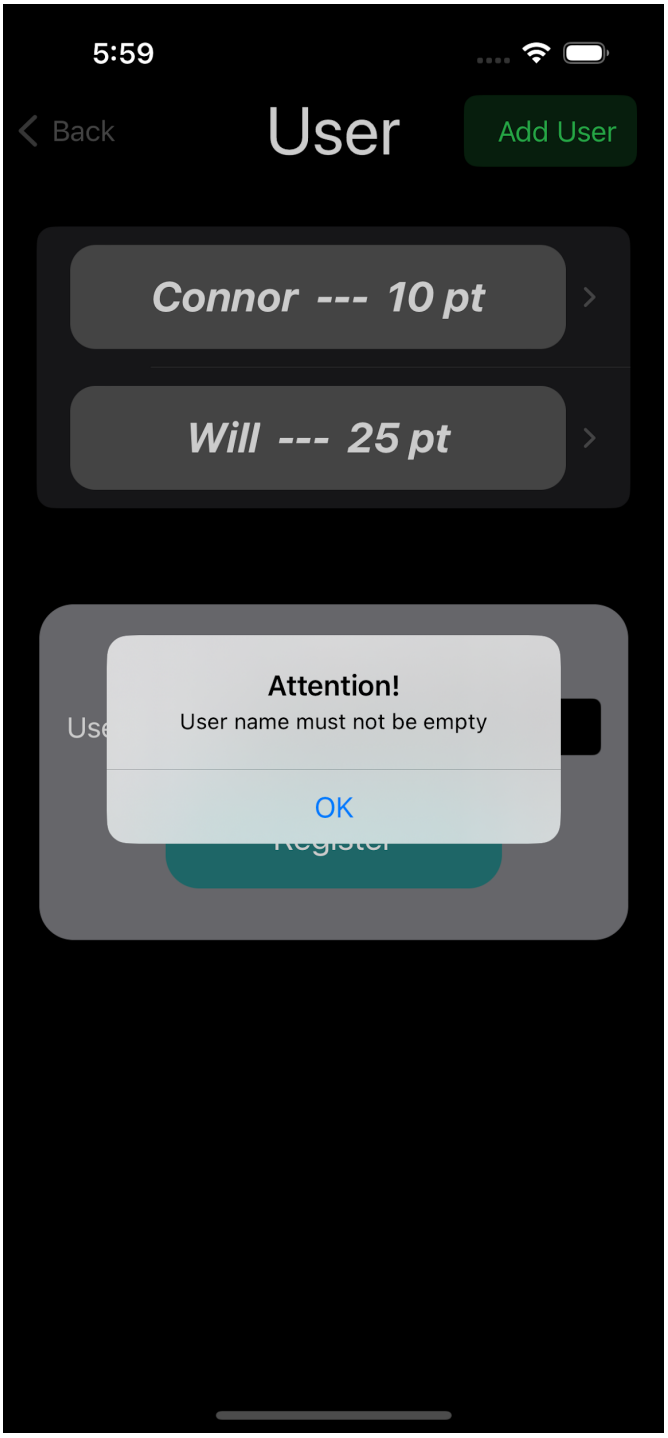


Select Difficulty

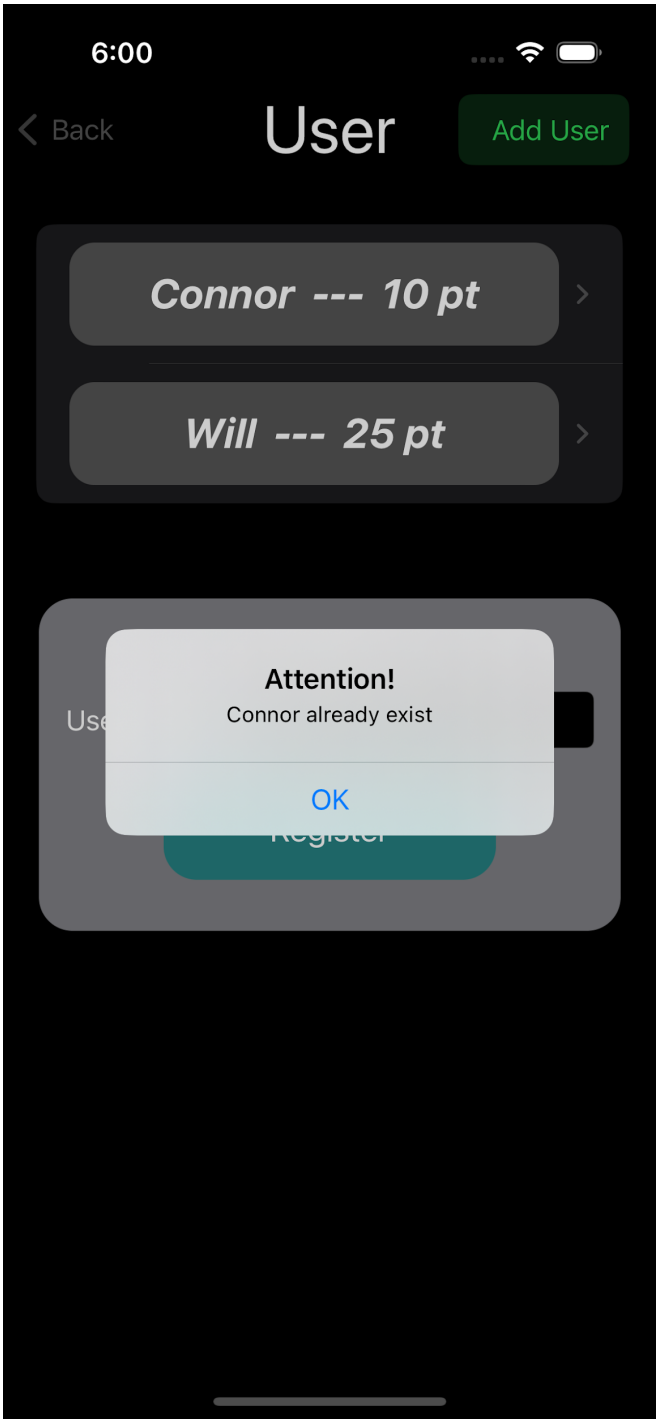
Selecting Users - Users



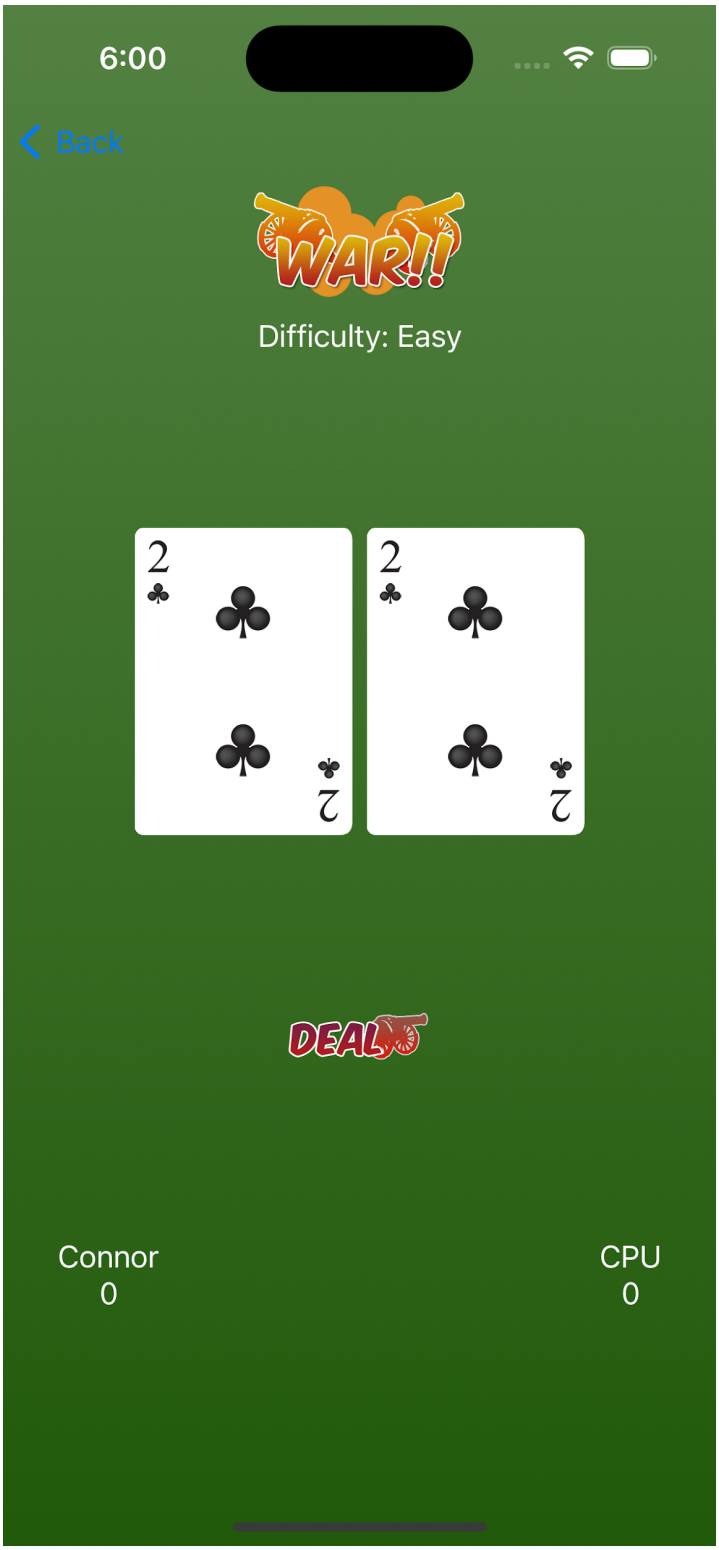
Selecting Users - No Users



Adding User - Name Empty

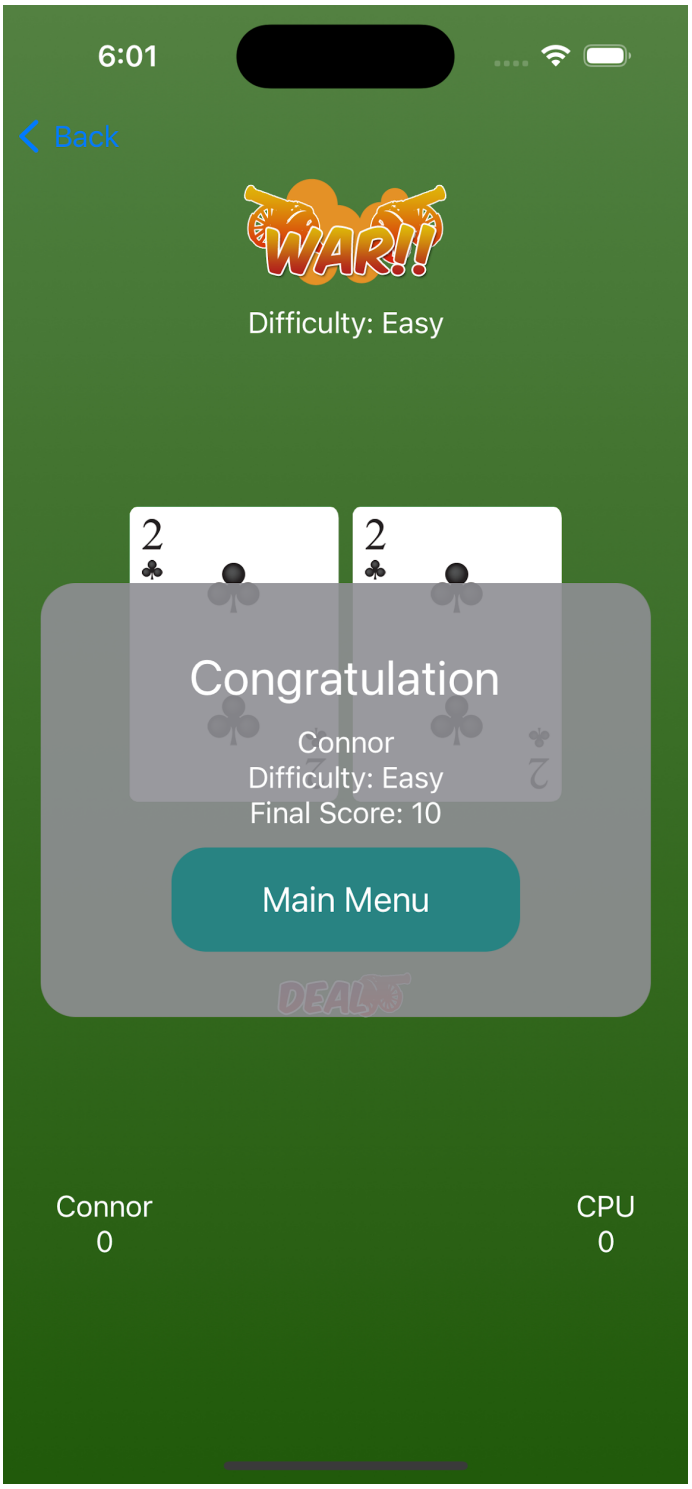


Adding User - Name Existed

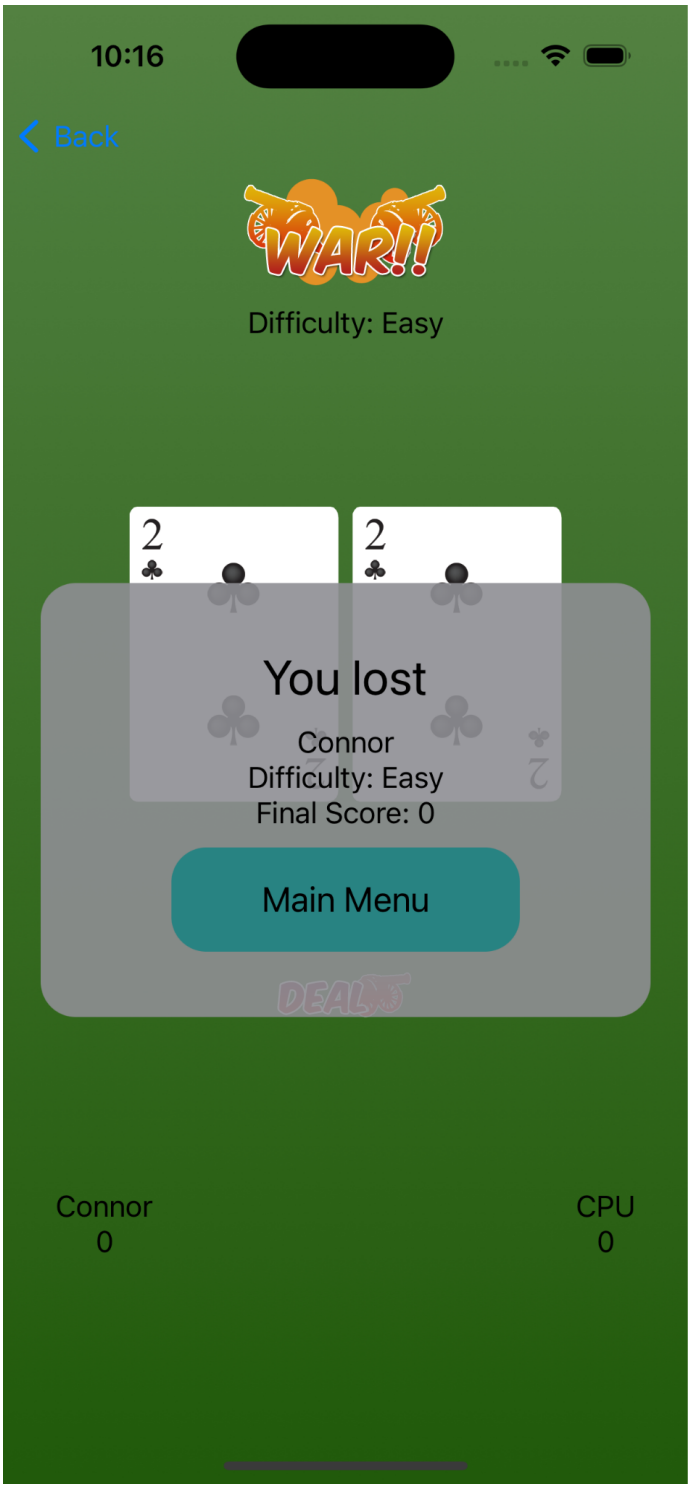


Game View

Game View - In Progress



Game View - Won



Game View - Lost