RMIT UNIVERSITY

COSC 2659 – IOS Development

Assignment 2

Lecturer – Tom Huynh

Jaeheon Jeong – s3821004

06.09.2023

School of Science, Engineering, and Technology

Table of Contents

1. Introduction

Figure1: 'Blackjack Casino' app icon

In this project, a game related to board games or casinos should be selected. Therefore, blackjack is chosen which is the casino game that makes the sum of the cards to nearly 21. The game is famous in every country and a common game with playing each other because it uses a poker card which everyone can get from the mart or keep in the house.

When I first started blackjack was after watching the movie '21'. It is a criminal and gambling movie that uses the blackjack. A normal university boy who is the main character is good at calculating things, so he makes a team with the professor and makes a trick to win the game by calculating the number. When watching the movie for the first time, I did not know the rule so that why the character needs to calculate something. After knowing the rule and re-watching the movie, memorizing all of the cards and calculating the possibility is essential when playing the game. I had played blackjack by a mobile game since watching this movie.

2. How to play

1. How to play ‘Blackjack Casino’

When the app is first initialized, it will show the menu with the app icon on the top. it has 6 buttons that are leading to play view, leader board view, guide view, setting view, stage view, and continue button. When the user wants to start the game and clicks the ‘play’ button, the text field is shown to enter the username before starting the game. After filling in the username and clicking the ‘register’ button, it will move to play view. it has 2 buttons which are ‘hit’ and ‘stand’ on the bottom and each dealer and player has 2 cards automatically. One of the dealer cards is showing the back of the card and if the user wants to get the cards to reach near 21, the user can press the ‘hit’ button or if the user wants to stop getting the cards and see the result, user can press ‘stand’ button. It will show the user win, lose, or draw and automatically start the game until the user loses all the coins. If the user wants to stop during the game, use the pause icon on the right top to quit the game.

2. Rules and tips

The Rule on this project is different from the original blackjack. It is the same to become the winner between the dealer and the player by reaching around 21. However, to make the game simple, the rule is slightly changed. If the user gets an ‘A’ the first time, the user can select 1 or 11 later and if the user gets the same number the first time, the user can split the card and play two times with divided cards. However, in this game, the ‘A’ is always considered as 1 and there is no split although the two cards are the same the first time.

When playing the game, the user should be careful to exceed 21. When the total score of the user’s card number exceeds 21, the user will lose the game. Therefore, users should think about which are the best. The safe range to win the game is 17 to 20. The dealer will normally keep getting the cards until the score reaches 17. After 17, it is dangerous to press ‘hit’ because it will exceed 21 easily if the next card is above 4. The possibility is low to not exceeding 21. Therefore, as a tip, press ‘stand’ when the score is 17 to 20.

3. Code implementation

1. Main features

A. Menu View

Figure2: Menu View

When the app is initialized, it shows the app icon on the top and gives six selections. Each row has two buttons.

```
if isPlaying {

    PlayView(playerName: username)
        .environmentObject(settings)
        .environmentObject(stageSetting)

} else if (isRanking) {

    LeaderBoardView()
        .environmentObject(userDataManager)

} else if (isStage) {

    StageView(playerName: username)
        .environmentObject(stageSetting)
        .environmentObject(userDataManager)
        .environmentObject(settings)
```

```
.sheet(isPresented: $showingInfoView) {
    InfoView()
}
.sheet(isPresented: $showingSettingView) {
    SettingView()
        .environmentObject(settings)
}
```

Figure 3&4: leading other View when clicking the button

By going to another view, each view has a bool type variable and when it becomes true, PlayView, StageView, and LeaderBoardView will compare with the if statement and move to the

following view. InfoView and the SettingView will use a pop-up sheet when the variable becomes true. It can be found in the 'MenuView.gif' from the GIF folder.

B. Play View

When the 'Play' button is pressed, it will show the text field to register the username as appendix 1. It will move to PlayView if the text field is filled. There would be no action if the user presses the button with the blank text field.

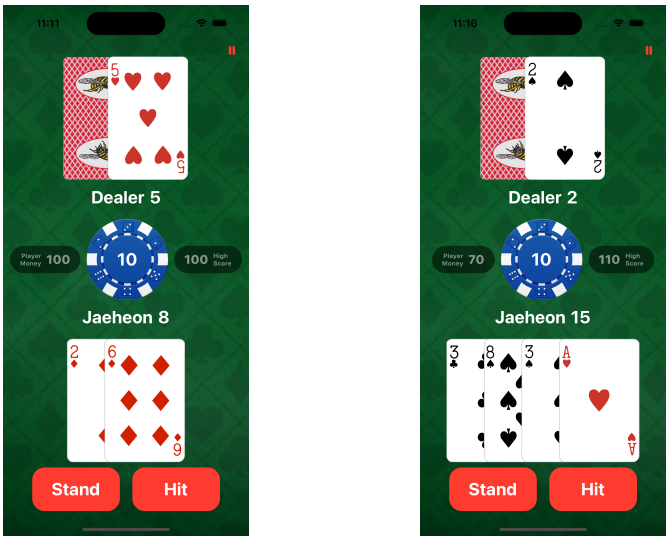


Figure 5&6: initial screen and after press 'hit' button

On the PlayView, below the chip, it will show the username that is registered on the text field. In Figure 5, 'Jaeheon' has been put into the text field. The top two cards are for the dealer, and it will only show the score excluding the covered card number. The below two cards are for the player, and it will change the number when the user gets an additional card by clicking 'hit'. In the middle, both sides of the chip will show the player's current score and the player's high score.

When the player's card number exceeds 21, becomes 21, or clicks the 'stand' button, it will show the result as the appendix 2,3, and 4 with opening the covered card from the dealer. The result will be shown with the sound which is embedded in the file. It sounds different with the win, lose, and draw.

To display a smoother and more realistic game, it has animation to get the card when the player presses the 'hit' button. It uses 'easeInOut' to get the card so that it looks like the dealer gives the card to the player.

For the PlayView, it can see the GIF animation on the GIF file 'PlayView.gif'.

C. Leader Board View

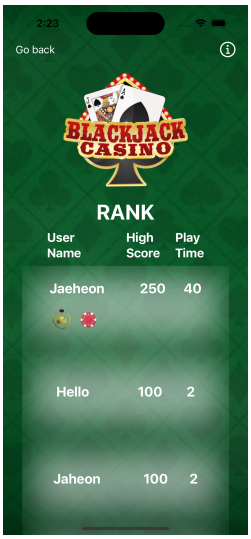


Figure 7: LeadaerBoardView when the 'Rank' button is clicked

The app is used with different users so it has to store all usernames and it should also store the highest score of the user and the playtime with the following user when clicking the 'Rank' from the menu view. It has a 'User' struct and it has an ObservableObject class called 'UserDataManager'.

```
// Define the UserDataManager class
class UserDataManager: ObservableObject {
    @AppStorage("users") var usersData: Data = Data() // Store users data in AppStorage

    @Published var users: [User] = [] {
        didSet {
            // When users are modified, update the AppStorage data
            do {
                usersData = try JSONEncoder().encode(users)
            } catch {
                print("Error encoding users data: \(error)")
            }
        }
    }

    init() {
        // Load users data from AppStorage
        if let loadedData = UserDefaults.standard.data(forKey: "users") {
            do {
                users = try JSONDecoder().decode([User].self, from: loadedData)
            } catch {
                print("Error decoding users data: \(error)")
            }
        }
    }
```

Figure 8: Process of encoding data to JSON file

Array 'users' is published and the data will be encoded as a JSON file. The initializer of the class is responsible for loading data from UserDefaults when an instance of UserDataManager is created. To use the data in the other screen, it uses the 'EnvironmentObject' property wrapper to inject the instance and it will allow it to read the property and observe the change.

There are achievement Badges for the players during the playing. By clicking the info icon on the right top of the LeaderBoardView, it will lead to BadgeInfoView as an appendix 6. the crown badge is given to the player who has the highest score from the users. And the red chip and blue chip badges are for play time. When playtime reaches 10, it will give a red chip badge and when play time reaches 5, it will give a blue chip. There are card badges which are the badge when the player complete the stage. Each card badge has a different score to reach.

The given badge will be shown in the LeaderBoardView below the username with the following achievement by the player. It can see in GIF folder by 'LeaderBoardView.gif'

D. Game Setting View

Figure 9: Setting View for difficulty, language and theme

When the 'setting' button is clicked, it will move to SettingView. For the difficulty, it has 3 options easy, normal, and hard. It will change the betting amount by selecting the different options. 10, 30, and 50 are settled and by choosing the higher betting amount, it will be risky because the starting score is 150 so there are only 3 attempts if the user keeps losing. The chip color will also be changed in the PlayView if the user selects a different level. It can identify in the GIF as 'SettingView.gif'.

```
class Settings: ObservableObject {
    @Published var isEasy = true
    @Published var isNormal = false
    @Published var isHard = false
}
```

Figure 10: 'Settings' class to use in other file simultaneously

To use the different levels with different colors, it makes the 'Settings' class which is the ObservableObject. In the PlayView, it will use the '@EnvironmentObject' wrapper to receive the same data of the changed value from the SettingView.

To close the InfoView, it contains '@Environment(\.dismiss)' to dismiss the sheet when clicking the 'x' icon on the right top.

E. How to play View

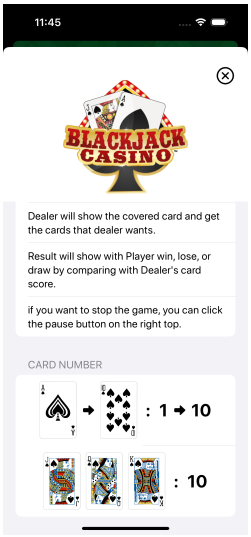


Figure 11: InfoView for how to play the game

When the 'Guide' button is pressed, it will move to InfoView which is the guide on how to play the game on this app. It will use the pop-up sheet to display the view and it is implemented by 'form' and 'section'. 'form' is a container for arranging the list and the section divides the part for different explanations.

The first section shows the steps of how to play the game and the second section gives visual aids about the card number calculation. For ‘J’, ‘Q’, and ‘K’, it will calculate as 10. The Last section will show the application information.

To close the InfoView, it will use ‘@Environment(\.dismiss)’ to dismiss the sheet in the same way to close the SettingView.

It can be checked in the GIF ‘GuideView.gif’.

F. Sound Effect

```
func playSound(sound: String, type: String) {
  if let path = Bundle.main.path(forResource: sound, ofType: type) {
    do {
      audioPlayer = try AVAudioPlayer(contentsOf: URL(fileURLWithPath: path))
      audioPlayer?.play()
    } catch {
      print("ERROR: Could not find and play the sound file!")
    }
  }
}
```

Figure 12: process of reading the sound file

For making the sound effect, the ‘playSound’ function is needed. It will use the AVFoundation framework to work with the audio. It will find the sound file using ‘Bundle’ and the sound will be played by the ‘paly()’ function which is from the AVaudioPlayer.

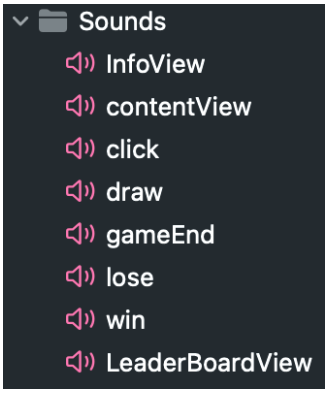


Figure 13: ‘Sounds’ folder to store every sound effect file

All sounds are stored in the ‘Sounds’ folder. Some sounds are for the background music and the others are for the action sound or the result sound. For the menu view, leader board view, and the how-to-play view, have the different background sounds. On the PlayView, when the player clicks

the button, it will play ‘click’ and when the win, lose, or draw is shown on the screen, it will play ‘win’, ‘lose’, or ‘draw’ accordingly.

2. Advanced features

A. Multiple Language Support

Figure 14: Switch mode for language selection between English and Korean

To access the different region players, it has a multiple language support system. The top right of the SettingView contains the switch mode to convert the language to Korean. When it switches to Korean mode, all screens should show Korean. Therefore, for bool type ‘isEnglish, ‘@AppStorage” property wrapper is used with the key ‘isEnglish. It can be used on every Screen by finding the key and it will read the user default automatically when the app is launched.

To display Korean, it will check true or false for the text and if the ‘isEnglish’ is true, it will show Korean as appendix 6, and 7.

It can be checked in the ‘LanguageToggle.gif’

B. Toggle Theme Setting

Figure 15: Switch mode for light theme and dark theme[1]

To play in the different themes, there is light mode and dark mode on the left top of the SettingView. To change the theme in the whole screen, the ‘@AppStorage” property wrapper is used with the key ‘darkmode’. The process is similar to the language switch mode. In this case, when the darkmode is true, it will change the image and color. The background and button color will be changed from Appendix 8 to Appendix 9.

For the light mode, the background color is green which is normally used in the casino table and the button uses red color. For the dark mode, the background color changes to dark red which

is also used in the casino table. For the button, dark green is used. About the text, both light and dark use white text because when using black text on a green background, the text is not clearly visible. The back of the card design is different from the theme. Red is used in the light mode and blue is used for the dark mode. It can be identified by 'ToggleTheme.gif'

C. Game progression and Levels

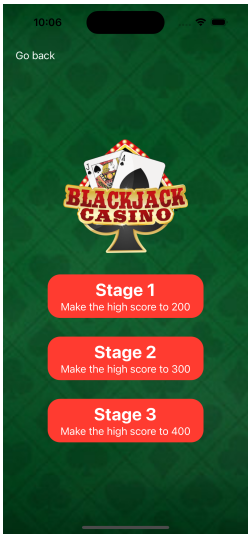


Figure 16: Stage View for game progression

It has 3 stages and before entering the StageView, a username is needed to access it. Each stage has to reach the required score. Stage 1 is 200, Stage 2 is 300, and Stage 3 is 400. It will use the 'StageSetting' struct as an ObservableObject so that it can read data on the other file when the value is changed. After completing the stage, it will show the alarm in the PlayView as an Appendix 19 shows.

```
class StageSetting: ObservableObject {
    @Published var stageOne = false
    @Published var stageTwo = false
    @Published var stageThree = false
}
```

Figure 17: 'StageSetting' class with ObservableObject

After clicking the stage button, it will move to PlayView. If the Stage 1 button is selected, then the ‘stageOne’ will switch to true. On the PlayView, it will detect the change of the ‘stageOne’ to true and if the ‘highScore’ reaches 200, it will show the alarm that the Stage is completed. The ‘reachStageOne’ variable in the User class will turn to true and it can see the badge in the leader board view.

It can be found out by ‘StageView.gif’ in the GIF folder.

D. Save and Resume

When the user exits the app and removes the memory, it has a resume system that the user can continue to play the game with the previous current score and highest score. The ‘continue’ button will occur only when the player removes the app during the game. if the player loses all money or quits the game by using the pause button, the ‘continue’ button will not do anything.

```
@AppStorage("continueUsername") var continueUsername: String = ""
@AppStorage("continuePlayerScore") var continuePlayerScore: Int = 0
@AppStorage("continueHighScore") var continueHighScore: Int = 0
```

Figure 18: Variables for storing in @AppStorage property wrapper

To save the data although the app is removed from the memory, it uses ‘@AppStorage’ to store 3 variables. ‘continueUsername’ stores the username that the player went out during the game. and the rest is storing the data of the current score and the highest score.

It can be checked in the ‘ContinueView.gif’.

4. Conclusion

In conclusion, this project helps to use the sound effect, ObserableObject by making the game. during the development of the game, it gives the progress of how the data is settled and how to make the logic for the simple game.

Furthermore, to make the better game, it could add more logic to the blackjack rule by making 2 cases when the player gets the ‘A’ at the first and splitting the card when the first 2 cards are the same number. Although it will need more functions and be hard to make the game, it will be entertaining by calculating the possibilities while playing the game.

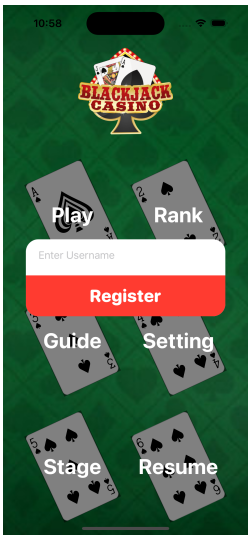
By creating the blackjack game, some features are learned how to use the app and some improvements to make it a better app.

The video can be found in this link: https://youtu.be/JJNrhzcK45g

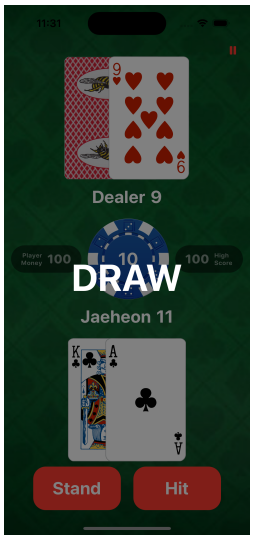
References

[1] “DarkmodeToggle,” GitHub, https://github.com/Inncoder/SwiftUI-Animations/tree/master/Animations/DarkmodeToggle (accessed Sep. 6, 2023).

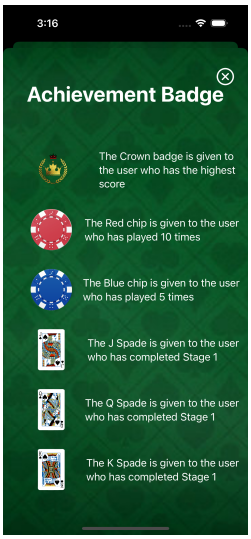
Appendices



Appendix 1: Text field appear with the register button



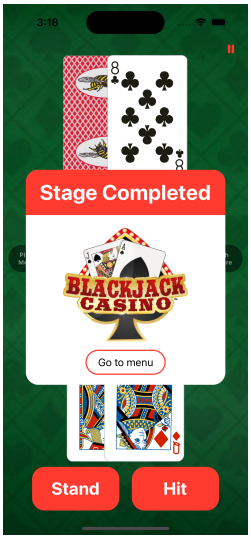
Appendix 2,3,4: Result during playing the game



Appendix 5: Achievement Badge Info View

Appendix 6, 7: Screen after changing the language switch mode to Korean

Appendix 8, 9: Screen after changing the light theme to dark theme



Appendix 10: Result after complete the stage