CSCI 205: Final Project
Ashley Albert
Eva Frankovic
Taylor LaMantia

Design Manual for Black Jack

Introduction

For our game of Black Jack, the creation of cards is needed as the basis to this game, thus we achieve that in a *Card* class. The *Card* class is responsible for having a visibility boolean, a rank, and a suit. The rank and suit are represented as enumerations, which helps compose the *Card* class. The *Card* class then composes the *Hand* and *Deck* class. The *Hand* class's card list is composed of cards from the *Card* class. The *Hand* class also represents, besides a hand of cards, the size of the hand and the number of aces in the hand as variables. This class can initialize a hand, make cards visible, find the best total in the hand, add a card to the hand, and clear the current hand with its methods. The *Deck* class has a list of cards in the deck and a list of cards taken from the deck as variables, thus it is composed of the *Card* class. This class' methods can initialize a deck, draw a card out of the deck, check for cards in it, shuffle the cards, and reset the cards. Both the *Hand* and *Deck* classes are associated with the *Card* class.

After the initialization of cards, hands, and decks, a user and a dealer is needed to actually play the game. Both the *User* and *Dealer* classes are composed of the *Hand* class. The *User* class initializes a user based upon a string of the user's name, places bets, handles money, clears the hand of cards, and deals the cards with its methods. This class has variables that represent money, the current bet, the current hand, and the name of the user. If the user attempts to place a bet above the amount of money they have, the "InsuffiecientFundsException" will be thrown. There is a dependency of the *User* class on the enumeration "WinState". The *Dealer* class only initializes one variable that is for the dealer hand. The methods of this class initialize the dealer, make or deal the dealer's cards as visible, and clear the hand. The *Game* class is composed of these things: the *User* class, the *Dealer* class, the WinState enumeration, as well as the *Deck* class. The *Game* class has variables to represent the player, the dealer, the deck, the size of a full deck, the amount of initial funds, and the initial WinState. The methods of the *Game* class include initializing the game, placing bets, evaluating the hands, resetting the game, allowing for the user to stand or hit, dealing out a hand, and handling the winner.

All of these classes, enumerations, and exceptions mentioned above are part of the model, but now we move onto the controller. The controller of our Black Jack game consists of five controllers: *BlackjackWelcomeController*, *BlackJackLoginController*, *BlackJackSettingsController*, *BlackJackPlayController*, and *BlackjackGameOverController*. First, the *BlackjackWelcomeController* has a label for the title, a button to press play, an image view to show the welcoming image, and an anchorpane to initialize the background. This class has a constructor and an initializing method as well as a method that determines what scene happens next when the button experiences an action, such as being pressed. This class also depends on the FXMLScenes enumeration and the blackjackMain class, both of which will be

discussed later. The next class, *BlackJackLoginController,* contains variables for the user's name, a go button, an initial balance label, a text field, and a balance. The methods within this class are the constructor, an initializer, the method that determines what scene happens next when the button experiences an action, and an event handler. This class also depends on the FXMLScenes enumeration. Third, the *BlackJackSettingsController* has four variables, each for a different button. It contains a constructor, an initializer, and a method that determines what scene happens next when the button experiences an action. Similar to the welcome controller, this class depends on the FXMLScenes enumeration and the blackjackMain class. Fourth, the *BlackJackPlayController* is responsible for helping the whole entire game run and appear on the screen to play. It contains variables for the player and dealer hands, the new and current balances, the amount in the pot, the reset pot value, the WinState, and if the game is started or not. Methods in this class include the constructor, the initializer, actions for if each chip is pressed, actions for each buttons that are pressed, a reset, an update total, a method that determines what scene happens next when the button experiences an action, and update methods for the dealer and user flowpanes. This class is composed of the WinState enumeration and the *Game* class while also depending on the *Hand* class and FXMLScenes enumeration. Lastly for the controller, there is the *BlackjackGameOverController* class. This class has variables for a button and an end balance label. Its methods include a constructor, an initializer, a method that determines what scene happens next when the button experiences an action, and a set balance. This class depends on the *BlackJackPlayController* and blackjackMain classes and the FXMLScenes enumeration.

Finally, the view of our Black Jack game consists of an FXMLScenes enumeration and a blackjackMain class, as mentioned earlier. Many classes are dependent upon these two things. The class "blackjackMain" is associated with the FXMLScenes enumeration and consists of a clip for the music and a boolean "isPlaying" to check if the music is playing or not. The methods in this class include the constructor, the method to load the next scene on the stage, the start, a music player and stopper, and a main. The enumeration "FXMLScenes" simply lists the possible scenes that there are to load, such as the welcome scene or the quit scene. Visually, our game shows the welcome scene first and prompts the user to select play or setting. Selecting play will take the player to a login screen while selecting settings will allow the user to turn the sound of soothing jazz music on or off then return back to the welcome screen. Once at the login screen, the user will be prompted to enter their name then select "Go". The play screen will then appear and have the user place bets below their given amount by clicking on the chips before selecting "Deal". Once selecting "Deal", cards will appear on the screen with a total score of the cards. From there, the user chooses to hit or stand. If the user hits, they receive another card, but if they stand they do not receive another card and keep what they have. The user cycles through these buttons until they find a value they want to stand with or if they lose. Other outcomes can be the user winning or tying with the dealer. The user can then decide to select "Play Again" or "Quit". If the user selects to play again, the play screen will reset, otherwise the selection of quit will lead to the game over screen and end the game.

User Stories

Our game of Black Jack is based on five types of players: a risky player, a strategic player, a card shark, a new player, and a power player. First, the risky player is described as someone who likes to win when there are high stakes. This player will bet nearly all of their chips to win and likes to play their Black Jack games at a fast-pace. This player also happens to enjoy being given opportunities to win back their losses. Our program was created to allow for risky players in that the game allows you to bet as much as you would like as long as it is below or equal to the amount of funds you have, meaning they could bet it all. This program was also designed to be able to be played quickly, if the user would like, as well as play multiple games in a row if they would like to win back their losses and build up their funds. The fact that this program has no ads and does not require real monetary funds benefits the risky player in their way of playing.

The second player type is the strategic player. The strategic player likes to think carefully about their odds in the game and makes their moves with a lot of thought in each play. Even the number of decks being used and the cards that have already been played are kept in mind as the strategic player goes through the game. Our game of Black Jack does not have any sort of time limit, thus letting the strategic player take their time thinking about what strategic move they would like to make.

The card shark is the third type of player who is looking to play this Black Jack game. This type of player is well versed in all card games, not just Black Jack, and impresses their friends as well as people at the casino quite often with their skills and knowledge of card games. A card shark type of player can use our program at home or on-the-go to enjoy their favorite pass time without having to travel to the casino, use their real money, give their information, and avoid ads. A serious player like this will enjoy being able to practice the game with no risks. The fact that our program has no ads also allows for the card shark to remain focused and play numerous times in a row without something pesky popping up.

The fourth player type is the new player. The new player of our Black Jack game, and Black Jack in general, has no experience in card games at all. The new player prefers a low-stakes game since they are just learning the ins and outs of how playing card games and placing bets works. Our program is able to help the new player learn how to play our game of Black Jack since it has instructions on the screens of what to do and is rather self-explanatory. There is also no risk for a new player in our game since it does not involve any real monetary funds.

The last player type is the power player. The power player is willing to go all-in no matter what and is a super competitive player. This power player has plenty of experience and is willing to do whatever it takes to win the game, even if that means betting all of their chips. Our program is great for a power player like this since they are able to bet however much they like, as long as it is less than or equal to the amount they have on the program, without any real life repercussions. The power player will also be able to select “Play Again” to keep playing the game over and over to continue winning and competing as they please.

Object-Oriented Design

CRC Cards

These eight CRC cards, as seen below, are some of the key classes to our game. These classes are discussed in depth within the Introduction section of this manual.

Class: Card	
Responsibilty	Collaboration
Initializes a single card with: • a suite • a rank • a value • visibility	Dealer Deck Game Hand User blackjackMain

Class: Game	
Responsibilty	Collaboration
Initializes a user, dealer, and deck in a game Can reset the game Places bets Deals a hand of cards to the user and dealer Player can stand or hit Hands are evaluated	Card Dealer Deck Hand User blackjackMain

Class: Chip	
Responsibilty	Collaboration
Initializes a value to a chip	

Class: Hand	
Responsibilty	Collaboration
Initializes a linked list of cards as a hand Has a size and number of aces Adds card to list and hand Can clear the list and hand Computes all the totals possible in the hand Finds the best score in the hand Determines the cards visibility	Card Dealer Deck Game User blackjackMain

Class: Dealer	
Responsibilty	Collaboration
Initializes a dealer's hand Sets the visibilty of the dealer's cards Gets the best score of the dealer's hand Has the ability to get or clear the dealer's hand	Card Game Hand blackjackMain

Class: User	
Responsibilty	Collaboration
Initializes a user's hand and amount of money Places bets Deals a hand of cards to the user with a visibility Gets the best score of the user's hand Has the ability to get or clear the user's hand Handles money	Card Game Hand blackjackMain

Class: Deck	
Responsibilty	Collaboration
Creates a linked list of all the cards, aka a deck Creates a linked list of all the cards taken from the deck Can shuffle or reset cards Pulls a card from the deck Can check if a deck has cards	Card Dealer Game Hand User blackjackMain

Class: blackjackMain	
Responsibilty	Collaboration
Loads the scenes for our GUI Plays or stops music	All of the FXML Files

UML Diagrams

Our initial UML class diagram was developed after extensive research of other object-oriented black jack programs. The first iteration of the UML class diagram developed in LucidChart is a high-level overview of the key classes we wanted to include and that we believed were necessary, such as the player, dealer, game, deck, card, hand, and the view/controller classes. The preliminary UML class diagram in Figure 1 below shows the initial plan for connecting the classes.

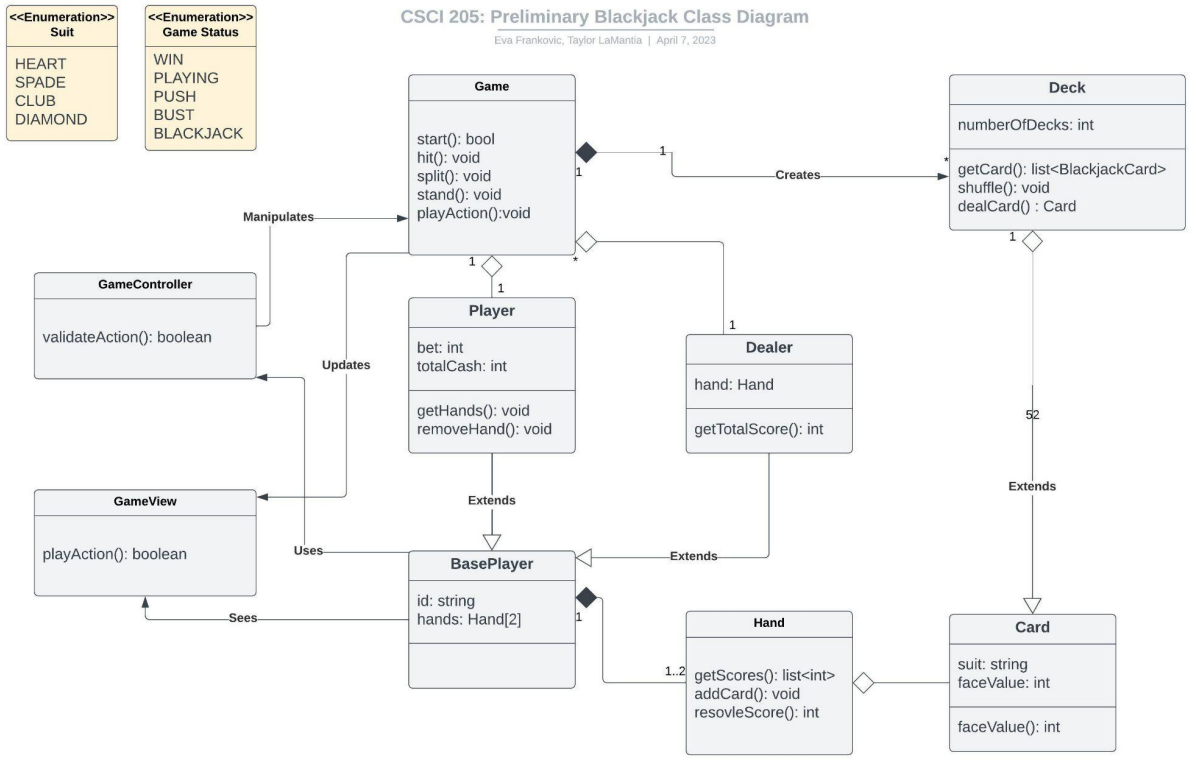

Figure 1: Preliminary Blackjack class diagram

Over the next month and 4 sprints, the final UML class diagram developed to include additional classes that were found to be necessary as can be seen in Figure 2 below. Some of these classes that we have included are additional controller classes and more enumerations, like the FXMLScenes enumeration.

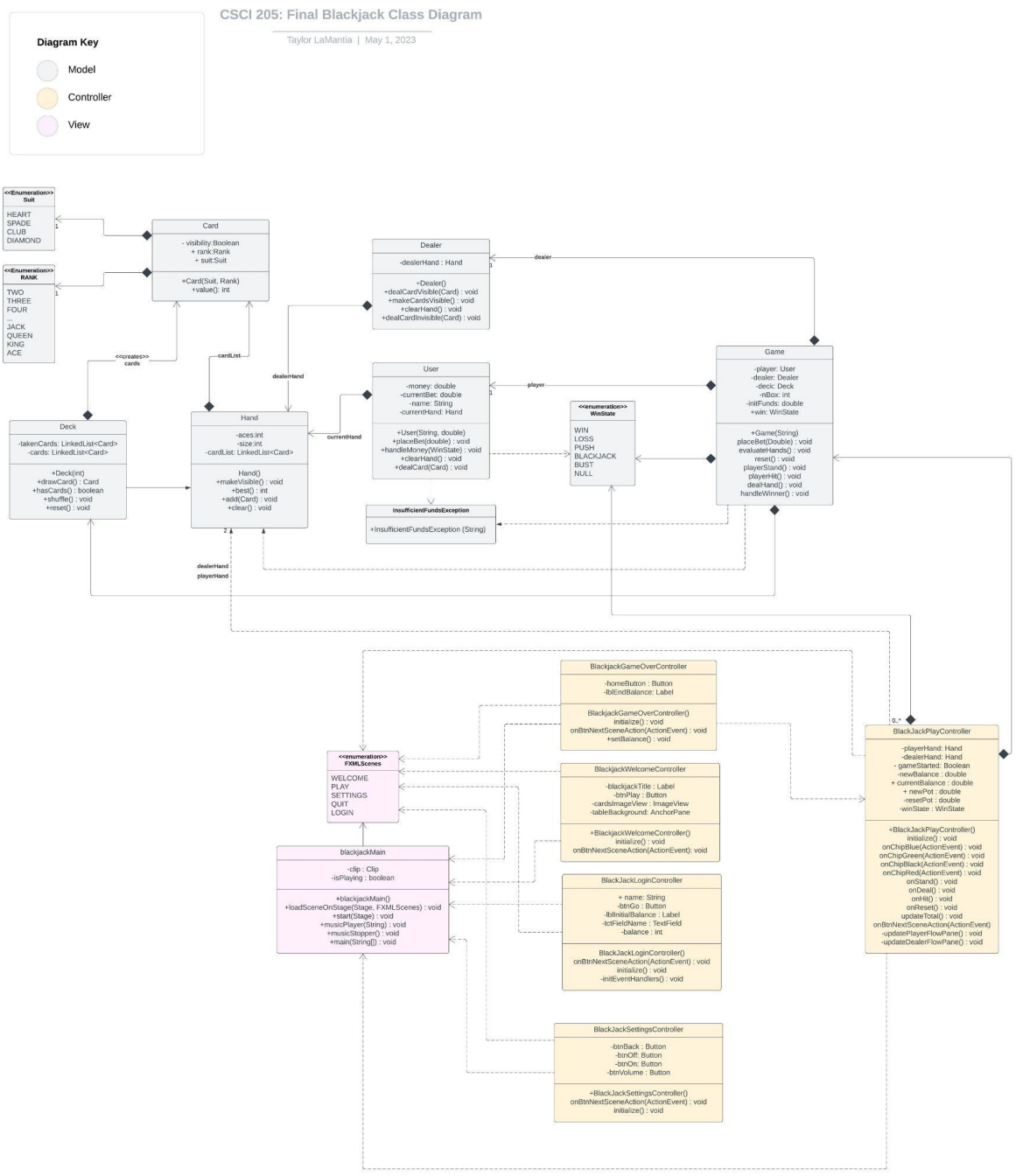

Figure 2: Final UML Diagram

Figures 3 through 7 are zoomed in screenshots of our IntelliJ generated UML diagram. Once again, these classes are discussed thoroughly in the Introduction section of this document.

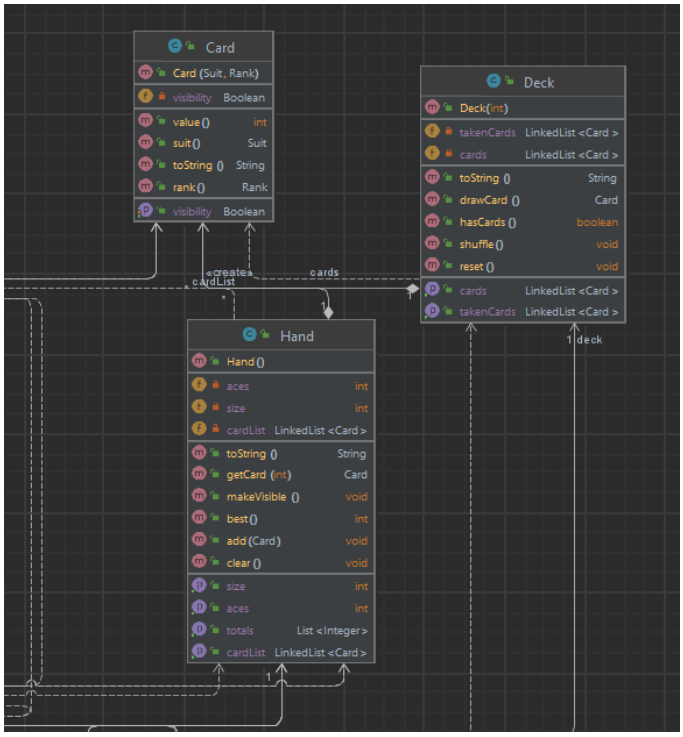

Figure 3: Card, Deck, and Hand Classes

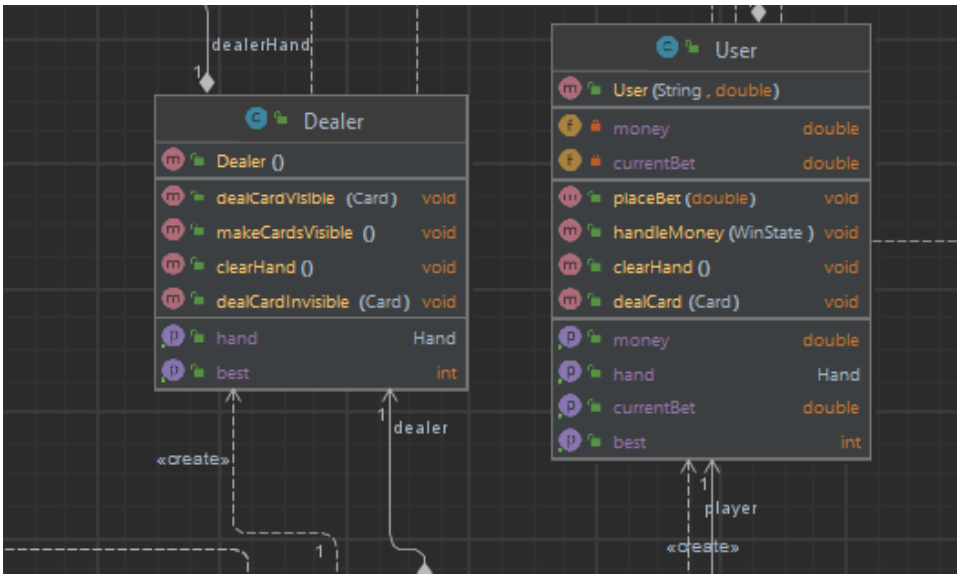

Figure 4: User and Dealer Classes

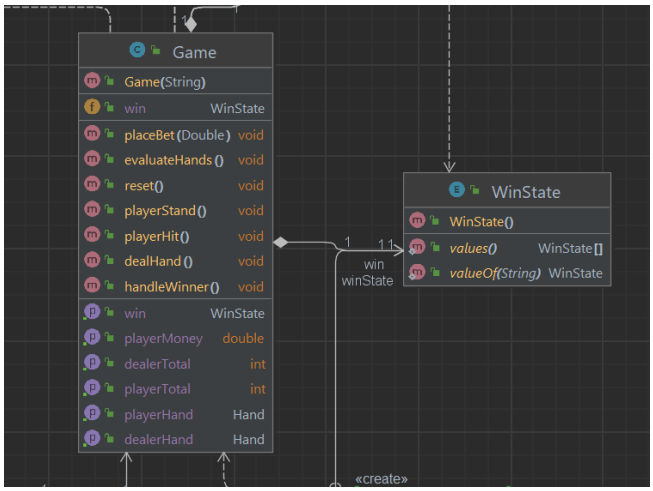

Figure 5: Game Class and WinState

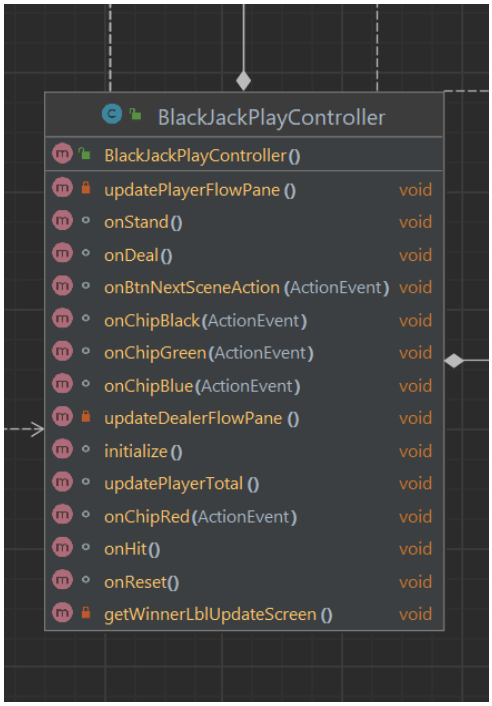

Figure 6: Play Controller Class

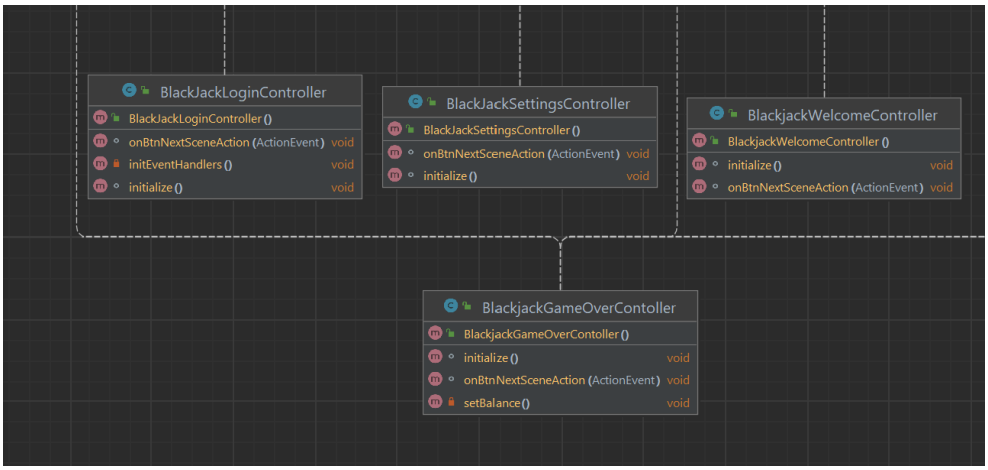

Figure 7: Login, Settings, Welcome, and Game Over Controller Classes