QUasino System Design Document

TEAM 23 MONTREAL

Graydon Belsher

Oscar Brown

Abbey Cameron

Victor Ghosh

Matthew Mamelak

Cameron Overvelde

Ethan Silver

Henry Wilson

Roy Li

CMPE 320

Nov. 6th, 2022

Introduction

Online casinos were first popularized in the early 2000s. However, they have seen tremendous growth and mass adoption from new users in recent years. There are several reasons for this rise, including the COVID-19 pandemic, which forced the closure of in-person casinos and led many to switch to online casinos. The convenience of playing classic casino games from the comfort of your home was appealing to many and helped grow many online casinos, such as Roobet, BetMGM, and BetRivers. Furthermore, with the recent passing of provincial laws, online casinos have become legal to operate in Ontario which has allowed for even more growth in this space.

Purpose of System

Despite the growth of online casinos, many potential users are discouraged from playing due to the risk of losing money. The purpose of an online casino with free currency is to provide a fun and entertaining way to gamble without the element of risk. This type of gambling is often used by online casinos to attract new players or to promote new games. Additionally, online gambling with fake money can be used as a low-stakes, convenient, training tool for new or inexperienced gamblers. This app can also act as a more accessible gaming option, as it does not require age verification or identification.

Design Goals

This program will be a virtual casino experience, using the C++ programming language. The objective of the game is to provide an engaging casino experience for players. The game will feature three popular casino games and allows the player to play without betting real money. Players will be given a starting balance — this will be used in place of real currency. The players can spend their balance in the form of casino chips to participate in the games and try to earn more.

The application will feature a home page. This GUI will display the games that are possible to play, as well as the player's current balance. From here, they can select which game they would like to play: Blackjack, Roulette, or Odds Are. Instructions for the games will be available in a drop-down menu once a game is chosen to play.

Table 1. Functional requirements of the system

Stakeholder	Goal	Motivation
User	To be shown how desktop application functions through on-screen guide	I can become more familiar with the app when its first started
User	To pick and play one of the games	I can play the game
User	To view game rules	I can learn to play a new game
User	To view my balance and set limits	I can learn about betting losses
User	To place bets using chips	Betting feels realistic
Blackjack player	To view play tips	I can improve at the game

Roulette player	To view outcome odds	I can learn about the odds
Odds Are Player	To view outcome odds	I can learn about the game odds
User	To have the option to exit each game	I can navigate between the different casino games
User	To save my balance	I can exit the application and reload it with my past balance

Performance

The system should have a relatively quick response time. It should take no more than 2 seconds to process user input, including typed input or mouse clicks, and opening pages. The system will only perform one task at a time to reduce any variable conflicts, etc. The system should also be able to run smoothly on a variety of devices (including Mac and PC). It should have simple controls that can be easily learned and mastered, and it should be able to be played in short bursts. The game should also be able to be played offline with minimal data usage. It must be able to withstand being played for extended periods of time without crashing or losing data.

Dependability

The dependability of a game generally refers to how well the game works, and how stable it is. This can be affected by various factors, such as the quality of the code, the quality of the hardware it is running on, and user input.

The system will not implement any API's or security features, so there should be no latency issues affecting the speed of the system (i.e., no lag time).

The player will interact with the game via the keyboard for entering betting amounts or other game inputs (e.g., guessing a card) and mouse clicks to interact with elements on the screen. The software is a single player game; only one user can interact with the system at a time. Invalid user input will be handled by exception classes, for example, if the player entered a number out of bounds, or if they entered a color that is not used in the game. A series of tests will be conducted to ensure the output received is the output that is specified.

Testing will be performed to ensure all user edge cases are accounted for, and to ensure that the desired behavior correlates with the actual results.

Cost

There is no cost to implementing this software or to updating and maintaining the system, as it will be hosted on the user's own computers.

Maintenance

The system will be hosted locally on each user's device. Each subsystem (game) in this system will function (relatively) separate from the rest of the games. They may share a few classes, but because they are mostly distinct, it will be easy to update and add functionality to the overall system and to individual subsystems. Since each subsystem will have its own set of classes, the code will be easy to follow and understand. Inheritance between classes should help with debugging certain processes and finding the root of an error. A shared repository on BitBucket will be used to push and pull revised code. Using BitBucket will allow access to versions of the program, and see which member is responsible for each segment of code and any updates.

End User Criteria

There are several considerations for the end user criteria of this system. The casino game should work on a variety of operating systems (Windows, MacOS, Linux, etc.). The app should have a user-friendly interface and offer a variety of casino games. The app should provide customer support (instructions) in case any problems are encountered.

The system's graphical user interface (GUI) will have an intuitive layout, with an instruction dropdown menu located in the upper right corner of the screen. The player will also be able to see their credit balance, displayed at the top of the home screen. Once the user chooses a game to play, each game will have an instruction menu accessible at any time. Having a clear set of instructions will make it easy for new players to learn/understand how to play each game, as well as for more experienced players to check if the rules are different from how they normally play.

Finally, whenever the system requests user input, they will be prompted by a popup box to enter information, such as placing their bet. The popup box will be restricted, and the user will receive feedback if their input is invalid.

Definitions, Acronyms and Abbreviations

Below is a list of definitions for some of the language used in this document:

- GUI – Graphical user interface
- Player(s) – Also known as the software application users
- Credits – Players funds are represented via credits
- Chips – Players can place bets from their total balance in the form of chips, which each have different values representing a dollar amount
- API – Application Programming Interface

Software Specifications

Subsystem Decomposition

For clarity purposes, the application has been divided into four subsystems: the menu, Blackjack, Roulette, and Odds Are. The routing controller will control which of these windows is displayed and hidden. Diagrams and use cases are given separately in their respective subsystem sections.

Global Software Control

The UserController class will be implemented to act as the primary global control of the system. This class contains an unsigned private balance variable which will store the user's balance. An accessor method will be used to retrieve the user's balance, so it can be displayed in various places across the application. There will also be a mutator method used to adjust the balance; however, the user will never have direct access to the balance variable to prevent unfair changing of the balance.

The primary global control of page displayed will be handled by the class RouterController. This class will be sent information from the various buttons across the application, then complete the proper actions to display a page. Upon entering a game, the router control object will hide the menu page as opposed to being destroyed to avoid re-rendering the page each time the user navigates to it. This is at the expense of the memory needed to hold the menu page. Conversely, upon exiting a game window, the game window will be destroyed by the route controller when the exit button is hit. This is done to minimize the memory needed by the application and is done under the assumption that an individual game's window will be accessed less frequently than the menu page meaning less re-rendering.

Boundary Conditions

The only boundary condition for the app will be through will be through the menu frame. You can exit through the settings popup window which will begin deconstruction of the CasinoApp object. This is done by calling the exit method of the app frame which is implemented by WxWidgets. Similarly, startup is initiated by the CasinoApp object, but will result in the rendering of the MenuFrame along with buttons to each of the games. The most expected error for this app has to do with the number of credits bet. For example, if the user has 10-dollars in their account and attempts to bet 15-dollars an exception will be thrown. Moreover, if the player has 10-dollars and places two bets which sum is greater than 10 dollars the game should throw an exception. Finally, if a player tries to commence a game without wagering any credit the game will again throw an exception and not begin.

Persistent Data Management

This application will store all data in a local instance of the app and will not utilize a database. This was done to simplify the application and eliminate database transactions to speed up operation.

Access Control and Security

There will only be one user case for this app, a player. A player can interact with all front-end components but will not have access to various private elements of classes. Namely, the userController will be kept private from the user to avoid being able to unfairly change their balance.

Given that the application will not be dealing with real money, there are no real risks or special considerations which must be taken. Additionally, the user will not be required to enter any information such as a username, so there are no real privacy concerns with regards to usernames, passwords, or personal information.

UML Class Diagram

The entire UML class diagram for the QUasino Application is included in the Appendix. While each subsystem subsection will include UML diagrams specific to the functionality of each subsystem.

Subsystem 1: Menu Page and Routing

Use Cases

Enter Settings

Participating actors

- User
- MenuPopup
- RulesPopup

Flow of events

1. User wants to view rules
2. User presses settings button
3. Settings popup appears over the Menu Frame
4. User selects a game for which to view the rules
5. Signal is sent to the PopupControl which chooses the correct rules popup to appear
6. Rules popup appears and displays a description of the game
7. User clicks somewhere not on the popup
8. Rules popup disappears
9. User clicks somewhere not on the settings popup
10. Settings popup disappears and the view is not the original menu frame

Entry condition

- User has selected the settings popup button

Exit condition

- User closes rules and settings popups

Quality requirements

- Popups appear and disappear smoothly and leave the menu frame unaffected

User selects game to play

Participating actors

- User
- MenuFrame
- RouterControl
- Game Window

Flow of events

1. User enters the main menu of the application
2. User clicks a button to enter a game
3. Button sends signal to RouterControl
4. RouterControl determines appropriate Game window to open
5. Game window is rendered
6. Menu Frame is hidden

Entry condition

- User enters the app

Exit condition

- User clicks game button and enters game window

Quality requirements

- Game windows Render in 2 seconds maximum

Object Model

1. Entities: User, GameRules, MenuPopup

2. Control: UserControl, RouteControl, PopUpControl
3. Boundary:
    a. SettingsButton
        i. SettingsPopup
            1. RouletteRulesButton
                a. RulesPopupFrame
            2. BlackJackRulesButton
                a. RulesPopupFrame
            3. PickerRulesButton
                a. RulesPopupFrame
    b. RouletteImage
        i. RouletteButton
    c. BlackjackImage
        i. BlackjackButton
    d. PickerImage
        i. PickerButton

Sequence Diagram

The following sequence diagrams follow the typical flow of operation for a user entering the menu frame, navigating to a game page, and then playing this game. Figure 1 shows the case of the user navigating to the Roulette game, then exiting the game.

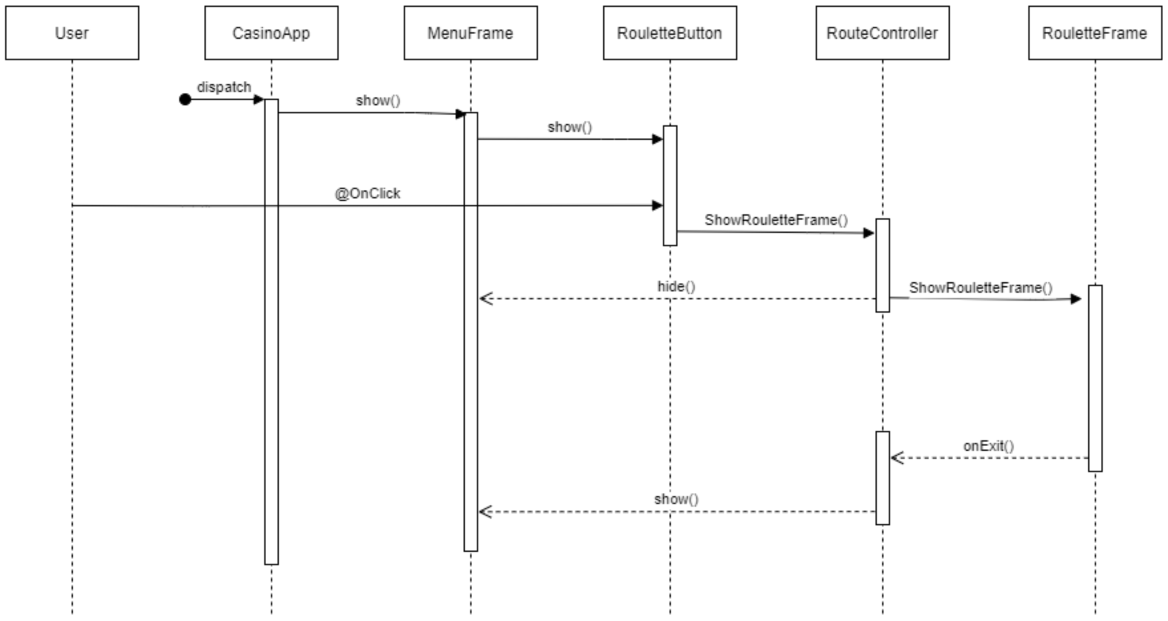


Figure 1: Sequence diagram for main menu.

State Chart Diagram

The state diagram below demonstrates the state behavior of the menu. Implementation and state diagrams for game subsystems are shown in their respective sections. The Settings process is a popup, meaning the menu will remain visible and active during use. The menu will remain active while all other games are being played.

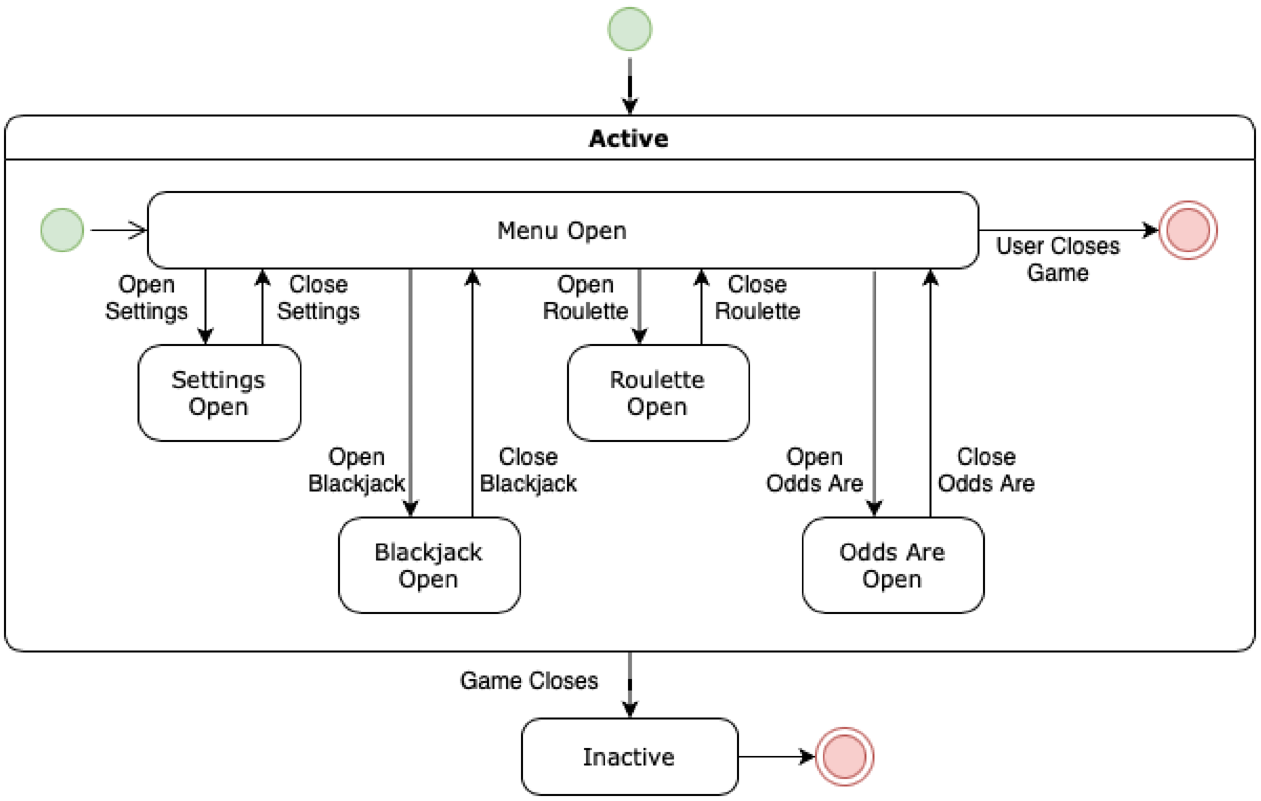


Figure 2: State Chart diagram for main menu.

Use Case Diagram

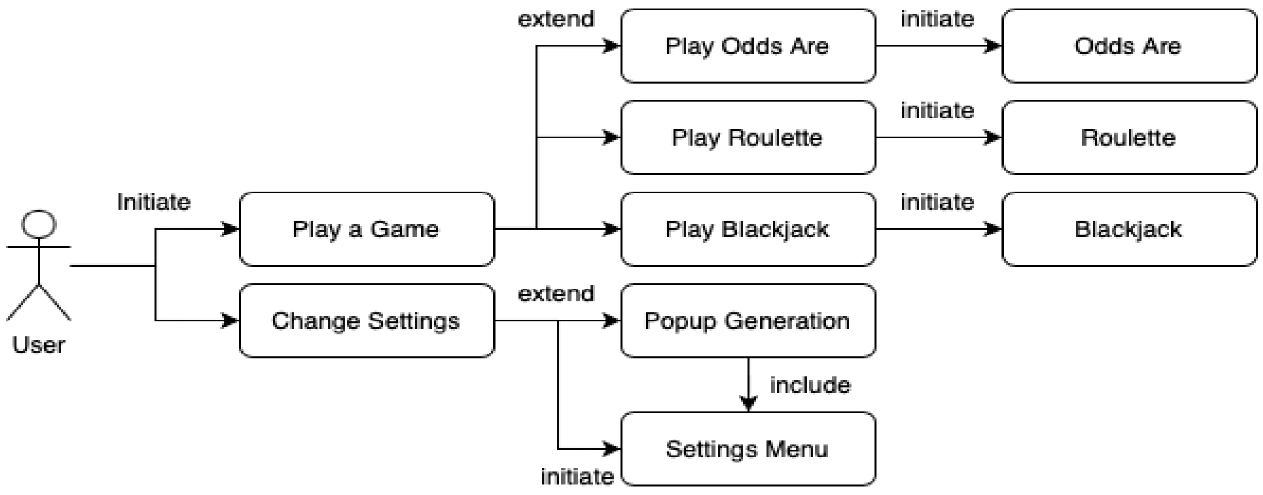


Figure 3: Use Case diagram for main menu.

UML Diagram

The UML diagram below shows the highest level and menu classes for the app. It includes frames for the individual games, the implementation details of which are discussed in their respective subsystem sections.

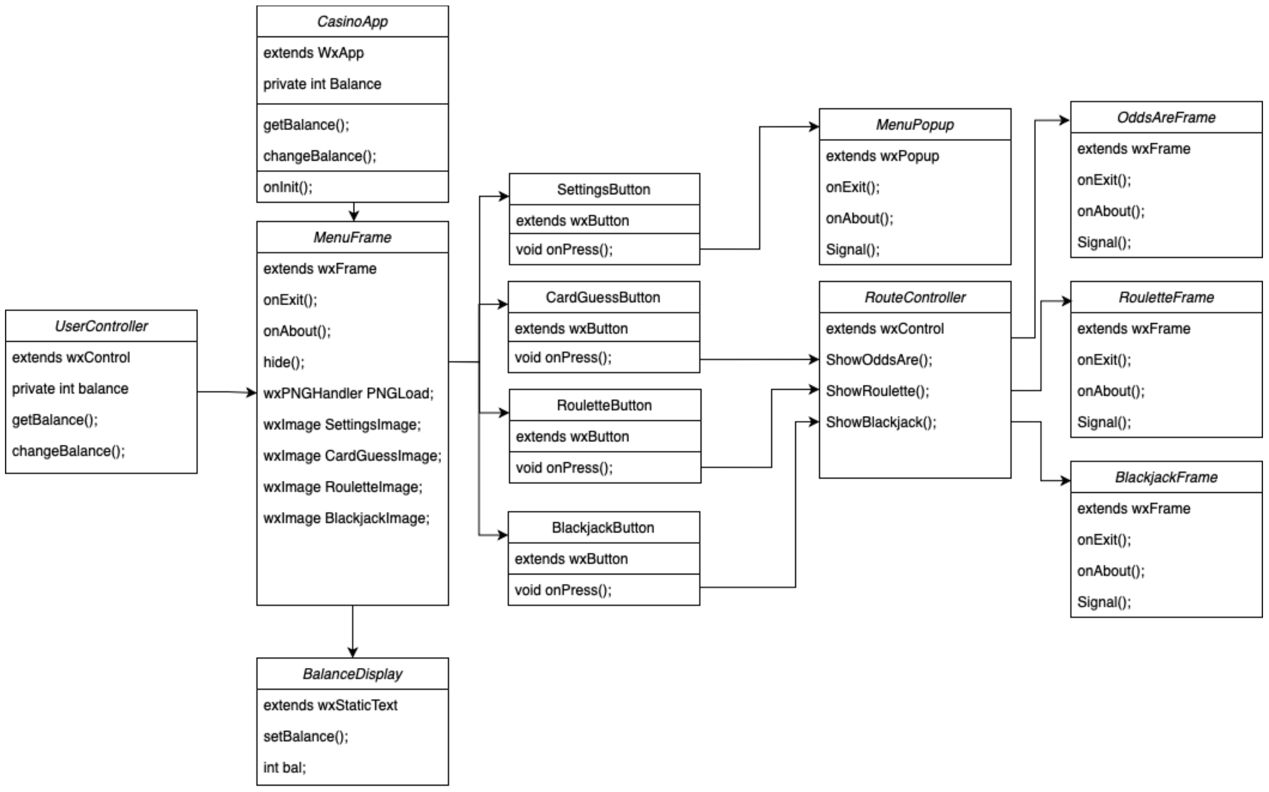


Figure 4: Class diagram for main menu.

Design Goals

While logic associated with the menu page is limited, it is essential that the menu provides a smooth navigation experience for the user. Specifically, this includes the settings popup rendering in under a second and in an aesthetically pleasing manner. While The menu frame is not directly in control of how fast the game frames will rendering, it is required that upon pressing a game button, a signal is sent to the routeController and then passed on to begin rendering the game frame with negligible delay.

While the menuFrame is not expected to be memory intensive, it is important that it uses as little memory as possible. This is due to the decision to keep the menu page hidden instead of destroyed when it is exited to avoid re-rendering. This will also ensure that upon opening the app the app object is able to render the entire menu in less than a second.

Software Hardware Mapping

Mouse clicks on the various buttons of the menu will be mapped to their respective actions. Not special memory considerations will be taken when dealing with the menu as logic and storage needed are limited.

Subsystem 2: Roulette

Use Cases

Play Roulette

Participating Actors

- User
- GameSelection
- Balance

Flow of events

1. User wants to play roulette; selects roulette
2. RouterController responds by displaying roulette UI (wheel, table, chips, balance)
3. Betting round begins
4. User selects which token they would like to bet
5. User selects inside or outside position to place token (process can be repeated multiple times)
6. UI displays location of chips placed in the betting round on the table
7. User submits bets once satisfied, betting round ends
8. Random number is generated for roulette spin, wheel spins until arriving at number
9. Random number is checked against each individual bet
10. Display spins wheel until ball is at random number
11. Money is awarded/ removed to balance following hit/ loss

Entry condition

- User has selected roulette game to play

Exit condition

- User has either won or lost at a round of roulette
- User decides to exit to main menu

Quality requirements

- Bet placing process is smooth and quick
- Wheel spinning animation is accurate
- Checking bets is quick
- Winnings are returned immediately
- Credits lost are deducted from the User after loss

Use Case Diagram

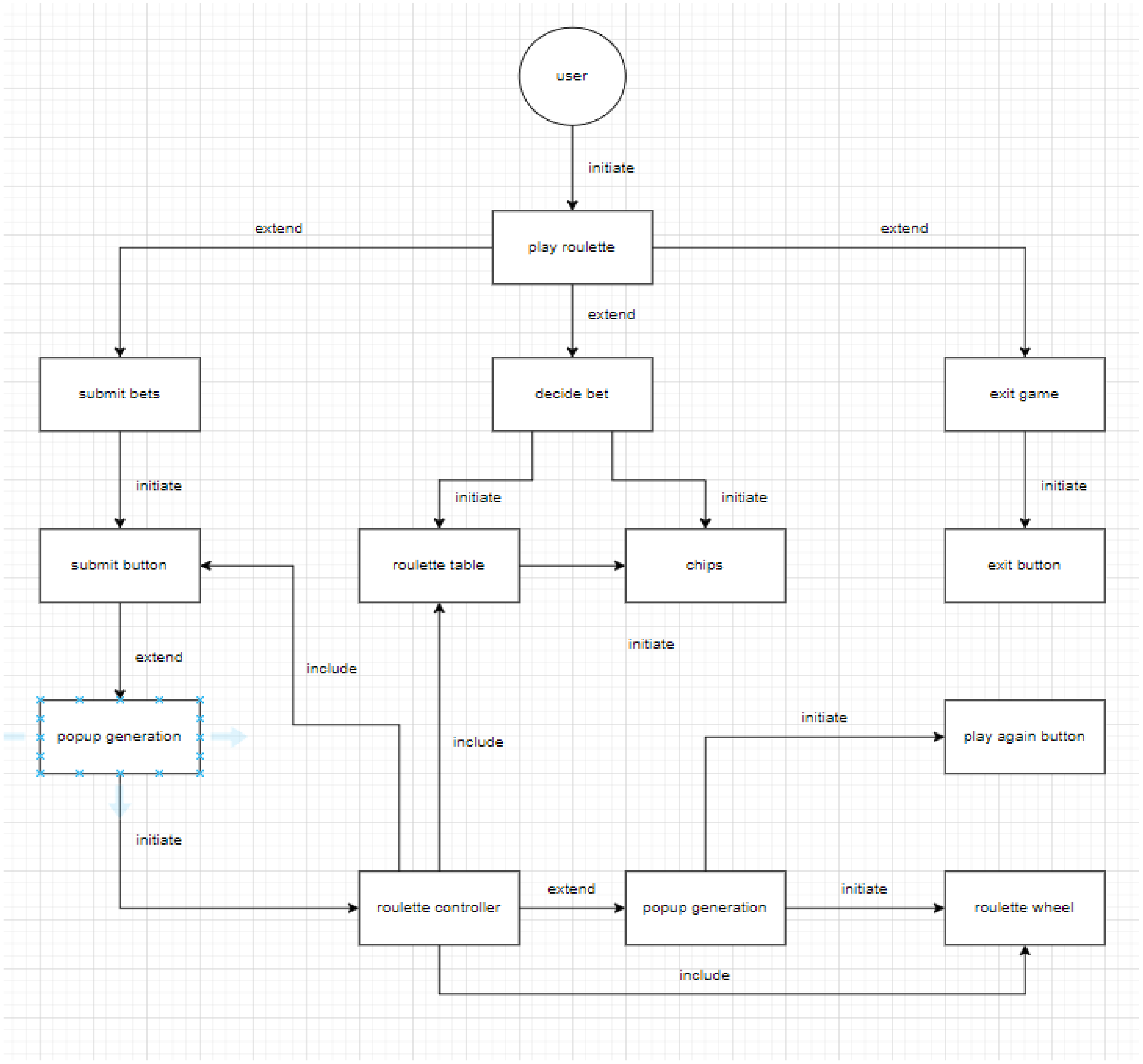


Figure 5: Use case diagram for roulette

Object Models

1. RouletteButton
    a. RoulettePopUpWindow
        i. RouletteTable
        ii. RouletteWheel
        iii. ChipTaskBar
            1. ChipIcon
                a. BetConfirmButton
    b. PopUpWindowCloseButton

Sequence Diagram

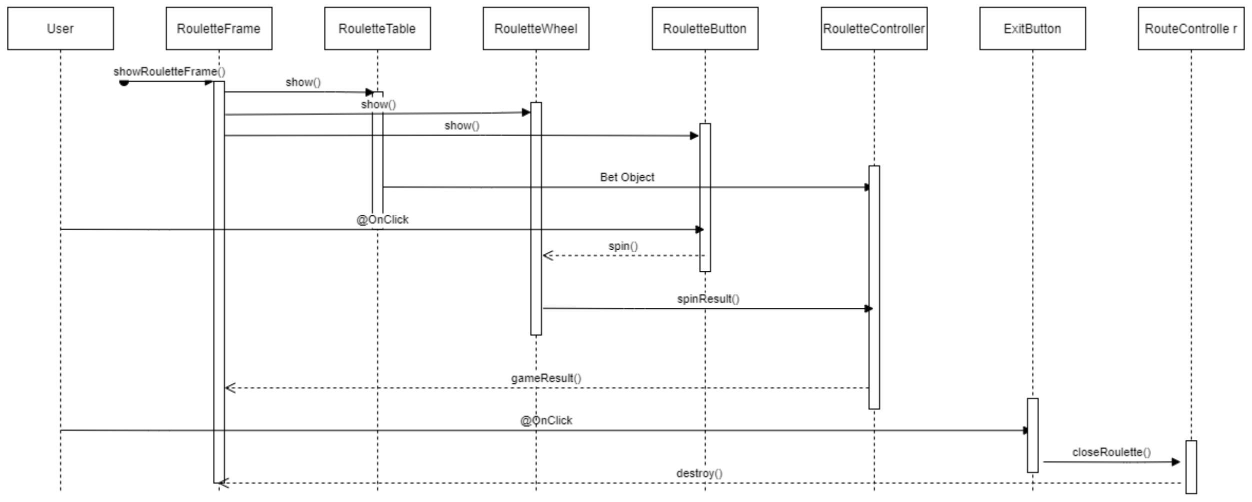
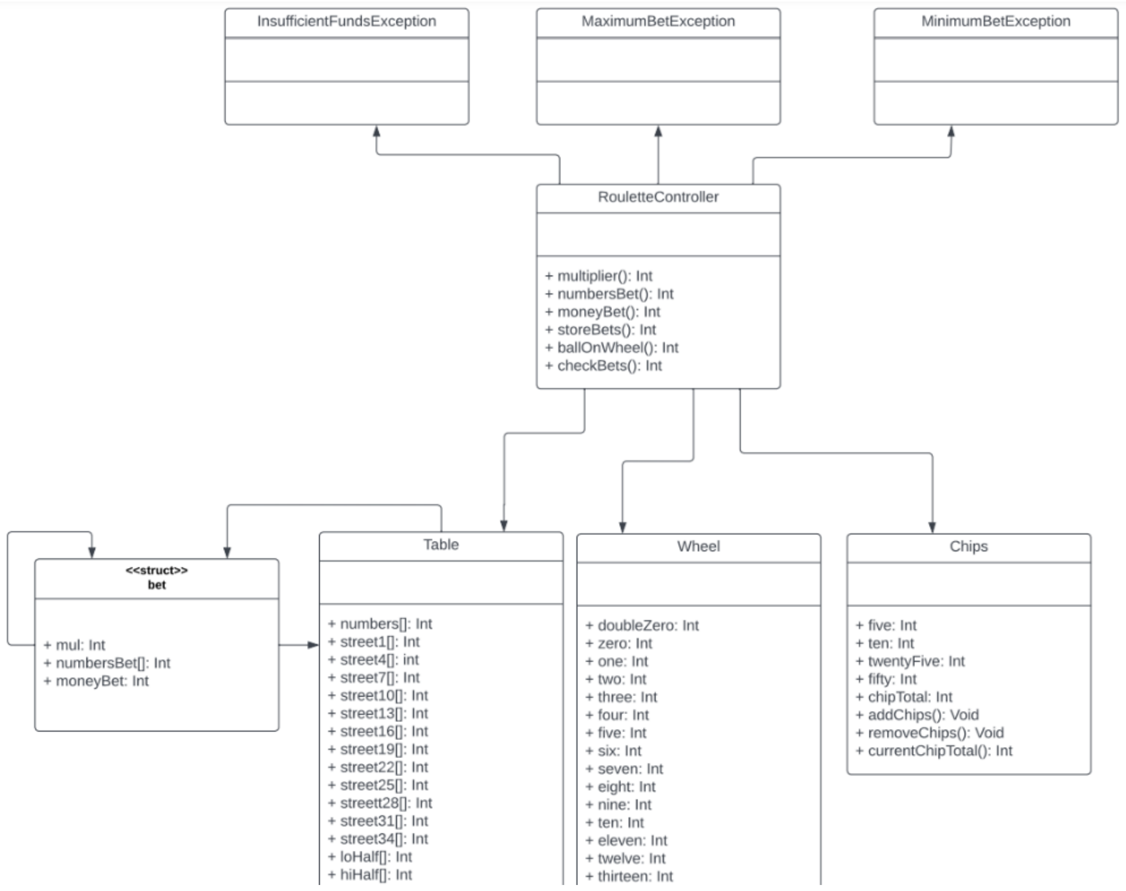


Figure 6. Sequence diagram for Roulette

UML diagram



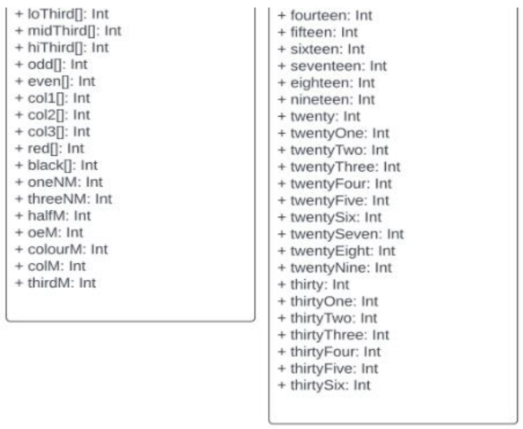


Figure 7: UML diagram for roulette

State Chart Diagram

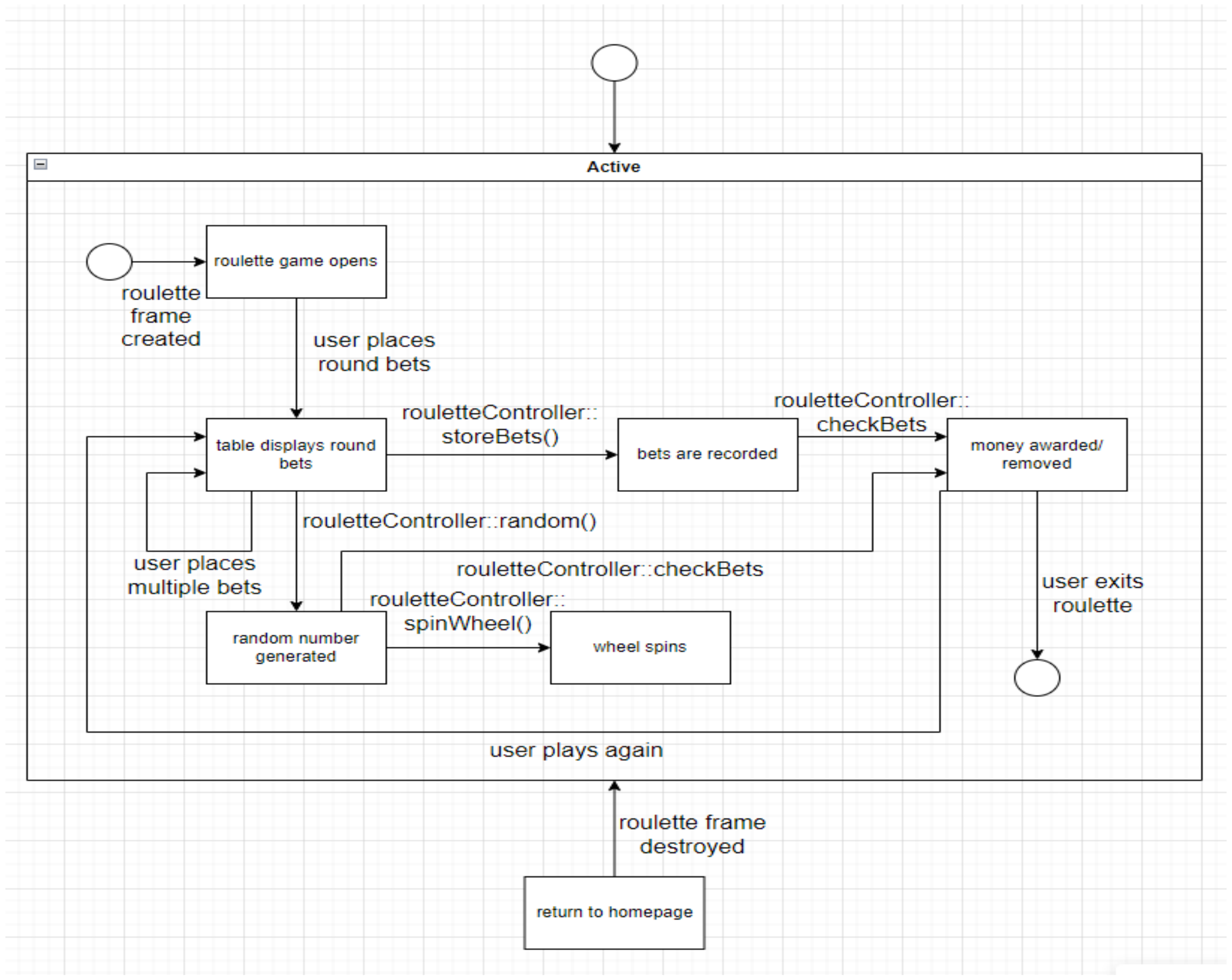


Figure 8: State Chart diagram for roulette

Design Goals

Within roulette, there exist a couple fundamental game functions that require optimization to ensure an ideal user experience. The first of which is the function spinBall() which will be used to generate the random number corresponding to a value on the roulette wheel. The functions degree of randomness will have to be sufficient where our game accurately emulates a real game of roulette. Secondly, the checkBets() function is used to verify whether any of the players active round bets have successfully hit when compared to the roulette wheel value that has been spun. The function will be required to correctly identify how many of the players' bets have landed and consequently award or remove the corresponding amount of money. Both the spinBall() and checkBets() functions will need to efficiently generate a value on the roulette and check the user's current bets to allow the player to continue with to the next round of roulette or exit to the homepage in a timely manner.

Software Hardware Mapping

The User will rely on the keyboard to supply bet input for each individual bet in each roulette round. While the mouse right click will be used to select the roulette game, select inside and outside bets, submit the bets in each roulette round, and select either continue to next round or exit to home page.

Subsystem 3: Blackjack

Use Cases

Play Blackjack

Participating Actors

- BlackJackControl
- Player
- Dealer
- Balance

Flow of Events

1. User wants to play Blackjack
2. User clicks on Blackjack button in main menu to begin game
3. Blackjack game page opens
4. BlackJackControl generates a shuffled deck object
5. User clicks instruction button to learn how to play
6. User chooses number of chips to bet by clicking on chip icons. Screen displays most recently clicked chip and total value of bet
7. Screen shows current balance in bank, each chip added subtracts its value from total balance
8. User clicks “Deal” button to deal hand and begin the round
9. Cards are dealt to dealer and user; graphics show cards being dealt. Players cards are face up; their sum is presented on the screen. Dealer has one card face up, one face down. Face up card’s value is presented on screen
10. User is given options to play the hand: hit or stand. In applicable cases, the player may split or double. The user keeps hitting until they stand, or their hand exceeds a value of 21.

11. Screen reveal dealer's second card. If value is 16 or less dealer will continuously draw cards until the total is 17 or more
12. If player wins hand, they win double what they bet and balance increases. If player loses, they lose what they bet (balance has already been decreased). If the dealer and player tie, player receives back what they bet.
13. User is given the option to play again or exit back to main menu.

Entry Condition

- User has selected Blackjack from main menu

Exit Condition

- User has successfully played a hand of blackjack or has quit before clicking the deal button.

Quality Requirements

- Bet placement is intuitive
- Payout is instant after each hand
- Dealer decisions are accurate

Use Case Diagram

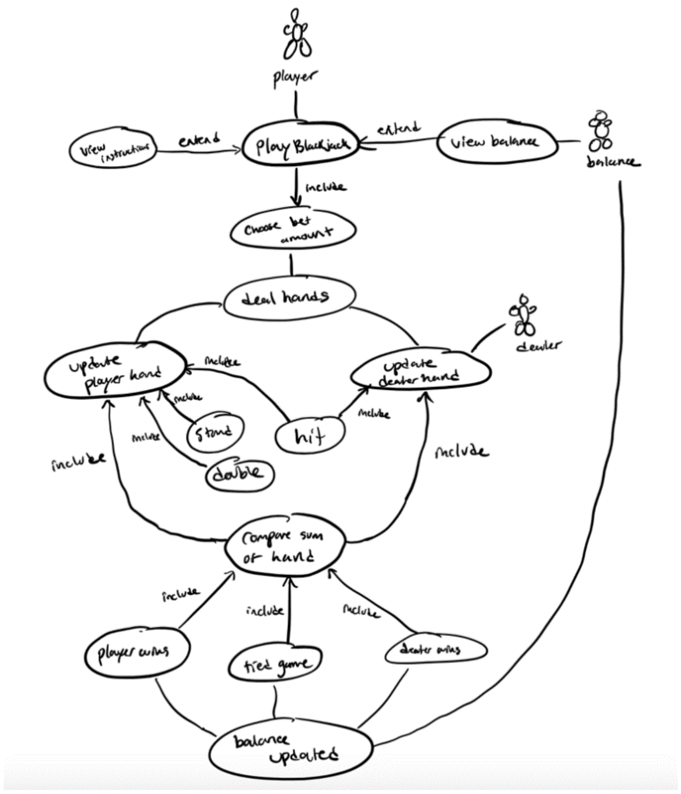


Figure 9: Use case for Blackjack

Sequence Diagram

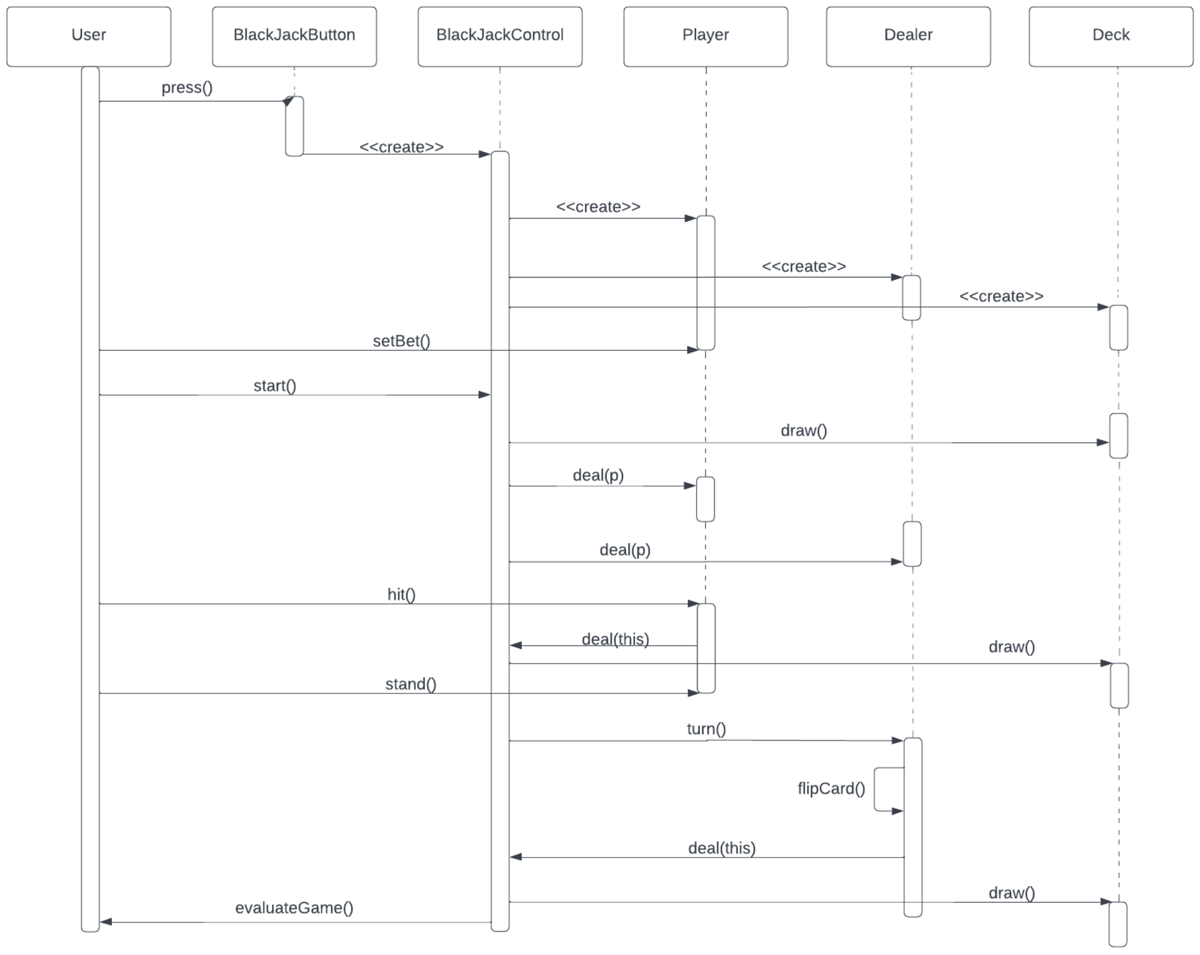


Figure 10: Sequence diagram for Blackjack use case

Object Model

Entities: Dealer, Player, Deck

Control: BlackJackControl

Boundary:

- HelpButton
- BetWindow
    - BetAmount
- DealerWindow
    - HandImage
    - TotalValue
- PlayerWindow
    - HitButton
    - StandButton
    - HandImage

- TotalValue

- QuitButton

Class Diagram

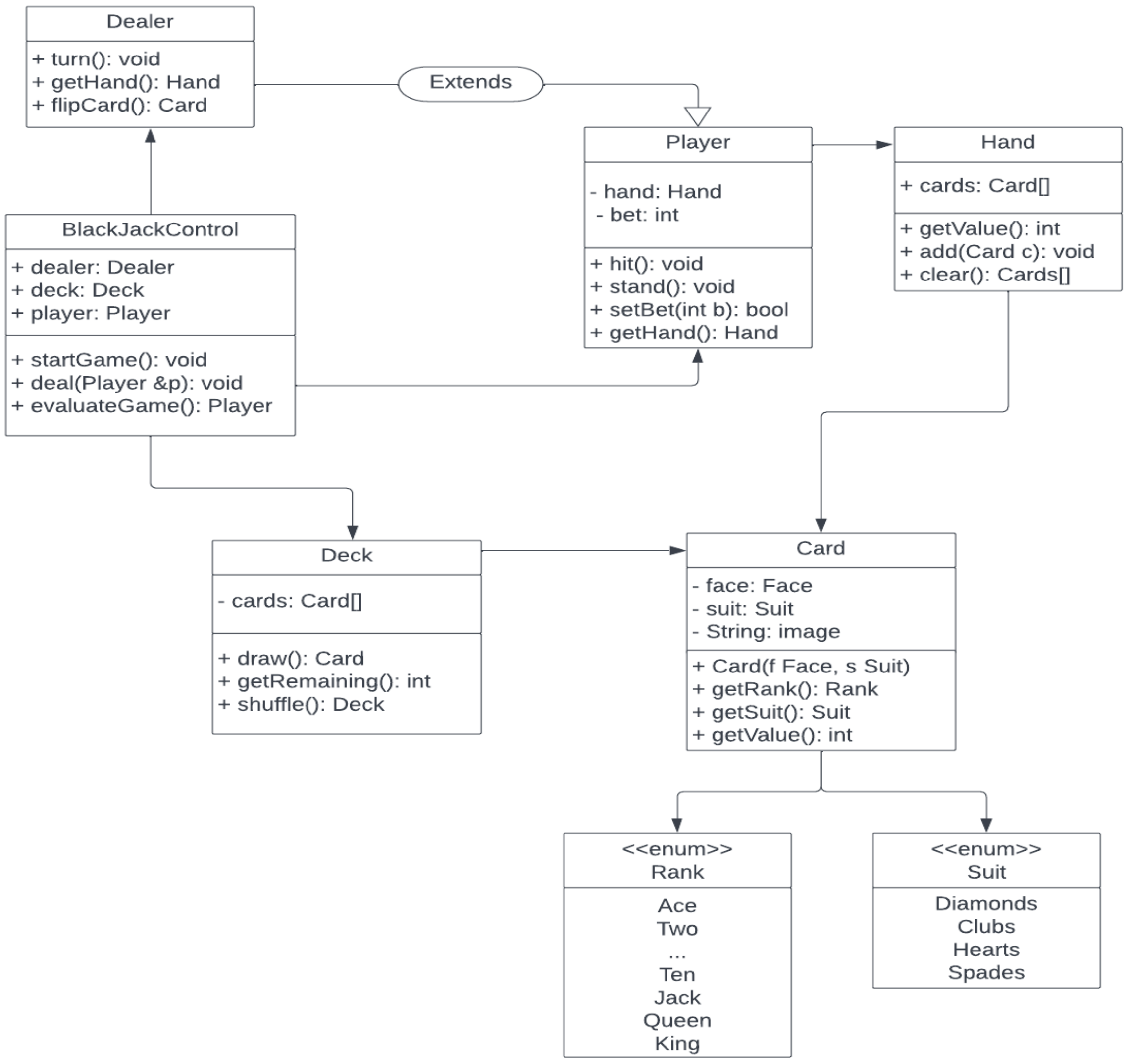


Figure 11: UML diagram for blackjack

Classes:

1. Card: describes attributes of a card
   - Constructor takes a Rank and Suit
   - Private enum Suit
   - Private enum Rank
   - getSuit() returns suit of card
   - getRank() returns rank of card
   - getValue() returns value of card
   - getImage() returns string that contains a filename to a specific image for the GUI

4. Deck: describes a deck of 52 cards
   - Constructor takes no values

- Private cards contains list of cards in deck
- draw() returns Card object and removed it from deck
- shuffle() shuffles cards in deck returning to 52 card deck

5. Hand: describes a hand of cards
    - Private cards contains cards in hand
    - getValue() returns total value of hand
    - add(Card c) adds a card to the hand
    - clear() removes all cards from hand
6. Player: describes a game player
    - Private hand contains the players Hand object
    - Private bet contains the players current bet about
    - hit() implements deal from BlackJackControl
    - stand() ends a players turn
    - setBet()
    - getHand() returns current Hand object
7. Dealer: extends Player
    - turn() has dealer operate as described in flow of events
    - getHand() overrides Player to hide card until flipped
    - flipCard() allows getHand to show all cards in hand

State Chart Diagram

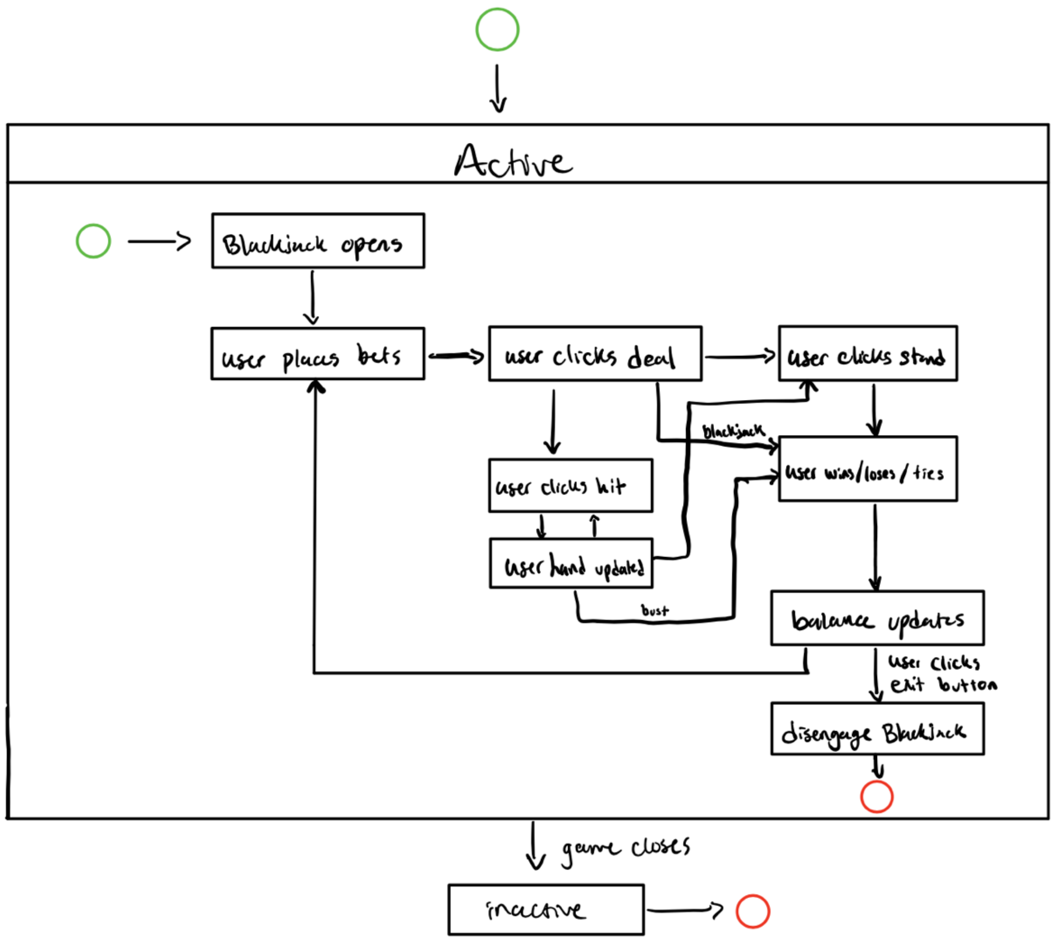


Figure 12: State chart diagram for blackjack

Design Goals

The Blackjack control scheme will be intuitive and easy to learn. After placing their bet players will be presented only with applicable options to the cards they were dealt. Explanations for why these options were presented will be easily accessible under the help menu. Users may also understand the dealer choices by reading the help menu. Finally, a countdown can be found in this menu to inform the user of when the shoe will be replaced. When a shoe or deck is shuffled its must be done randomly but because cards will be pulled from the top this does not need to be perfect.

Subsystem 4: Odds Are

Use Cases

Play Odds Are

Participating Actors

- User
- Balance

Flow of Events:

1. The user wants to play the card guessing game.
2. The user clicks on the card guessing game icon from the home page/main screen.
3. The user is directed to the home page of the card guessing game. They have the option between reading the instructions and starting the game.
4. Once the game has started, prompt the user for two inputs: user bet amount and the guessing multiplier.
5. If the bet amount is invalid, the player will be prompted to enter a new amount
6. The program generates a random card for the player to guess between the bounds that they specified in step 4.
7. Prompts the user to guess a card.
8. Compares the user input with the card generated in step 6. Returns the result.
9. Player gets winnings/loss
10. Players can choose to exit the game or re-play.

Entry condition

- User has selected the card guessing game to play

Exit condition

- User has either won or lost at a round of the game
- User decides to exit to main menu

Quality requirements

- Bet placing process is smooth and quick
- Card matching is accurate
- Checking bets is quick
- Payout/loss is immediate after game completion

Use Case Diagram

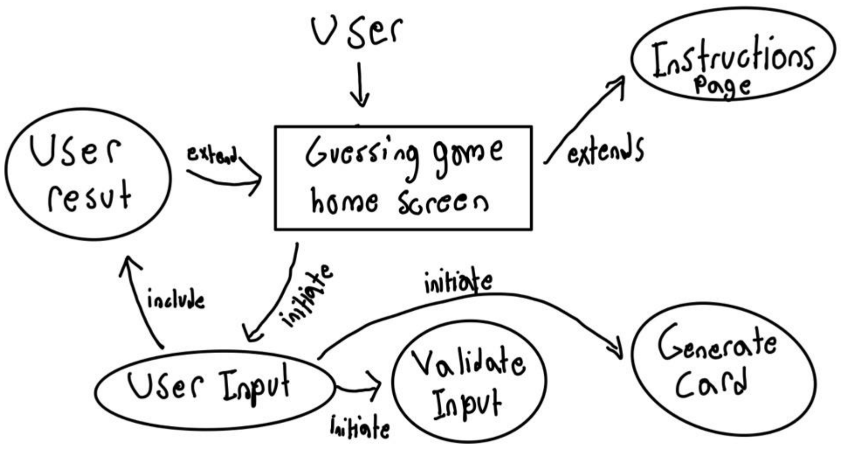


Figure 13: Use case diagram for roulette

Object Models

1. Entities: User, Instructions, GamePopup
2. Control: UserControl, PopupControl
3. Boundary:
    a. InstructionButton
        i. InstructionPopup
        ii. InstructionCloseButton
    b. StartGameButton
        i. GameImage
        ii. GameInputPopup
        iii. ReplayGameButton
        iv. ExitGameButton

Sequence Diagram

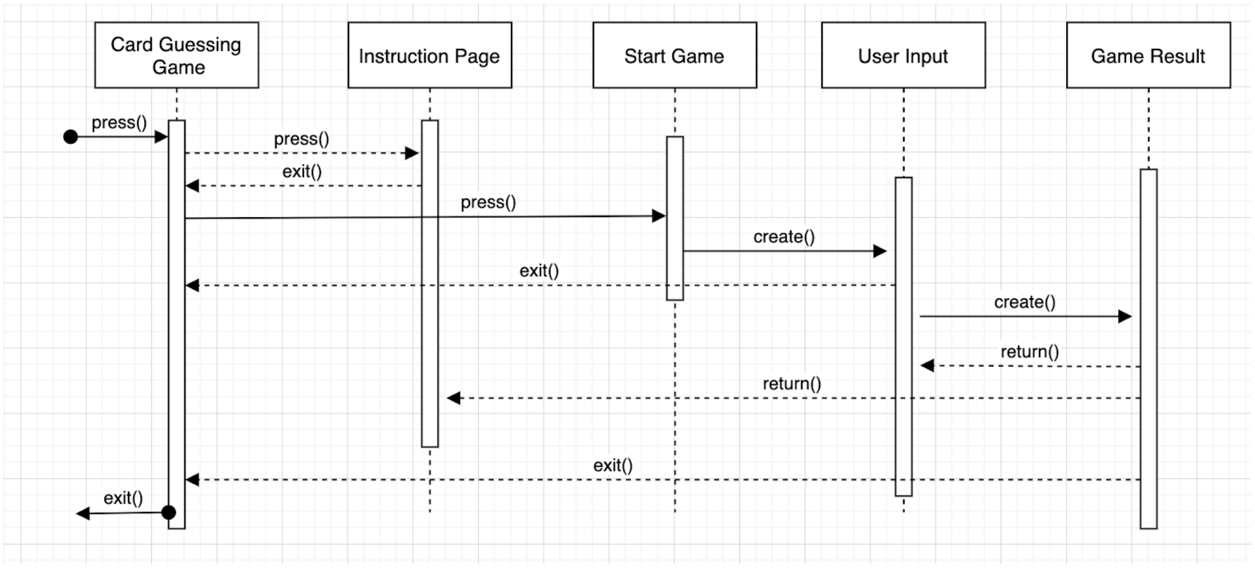


Figure 14: Sequence diagram for oddsAre

Class Diagram

Below is a breakdown of how the class for Odds Are will be structured:

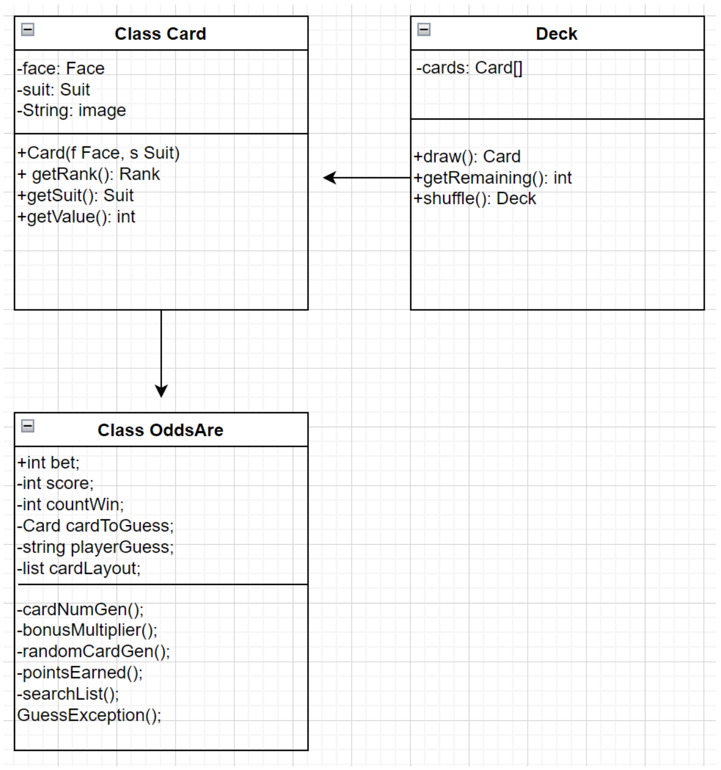


Figure 15: UML diagram for oddsAre

Class OddsAreGame

Functions

CardNumGen();	//gets user input of how many cards they want to guess from
bonusMultiplier();	//calculates if the player gets a win in a row multiplier
randomCardGen();	//generates a random Card object within the boundaries of the range
pointsEarned();	//calculates how much the player earned in a round
cardListGen();	//generate a list up to n cards for player to guess from
searchList();	// searches list to see if playerGuess is in the list
GuessException();	//enter invalid input

Variables

Int bet;	//how much player is betting
Int countWin;	//keeps track of how many wins in a row the player has
Card cardToGuess;	//Card object generated by randomCardGen()
String playerGuess;	//get the players guess
Int score;	//returned by pointsEarned(); final player score
List cardLayout;	//list containing various card objects

Uses Card(), Rank() and Suit() classes used in Blackjack.

State Chart Diagram

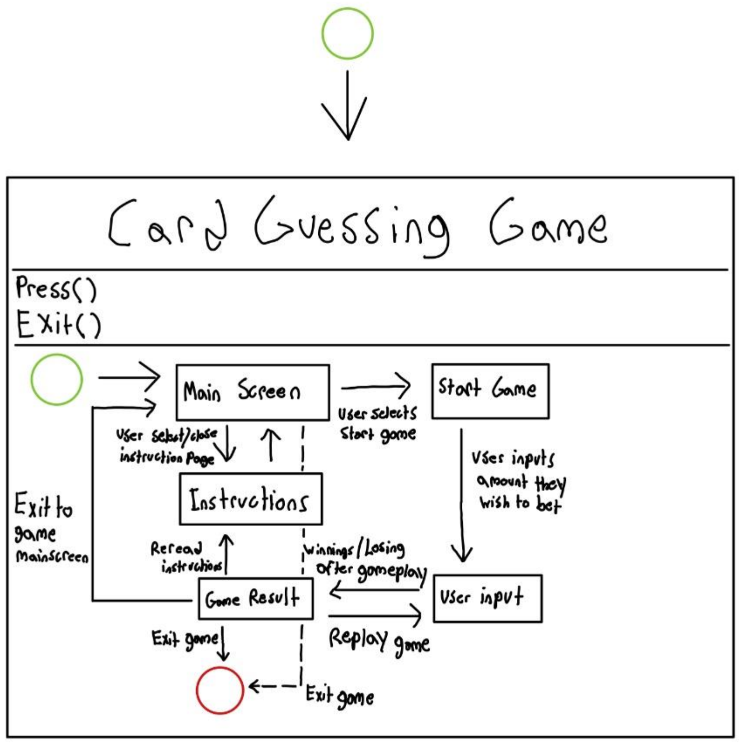
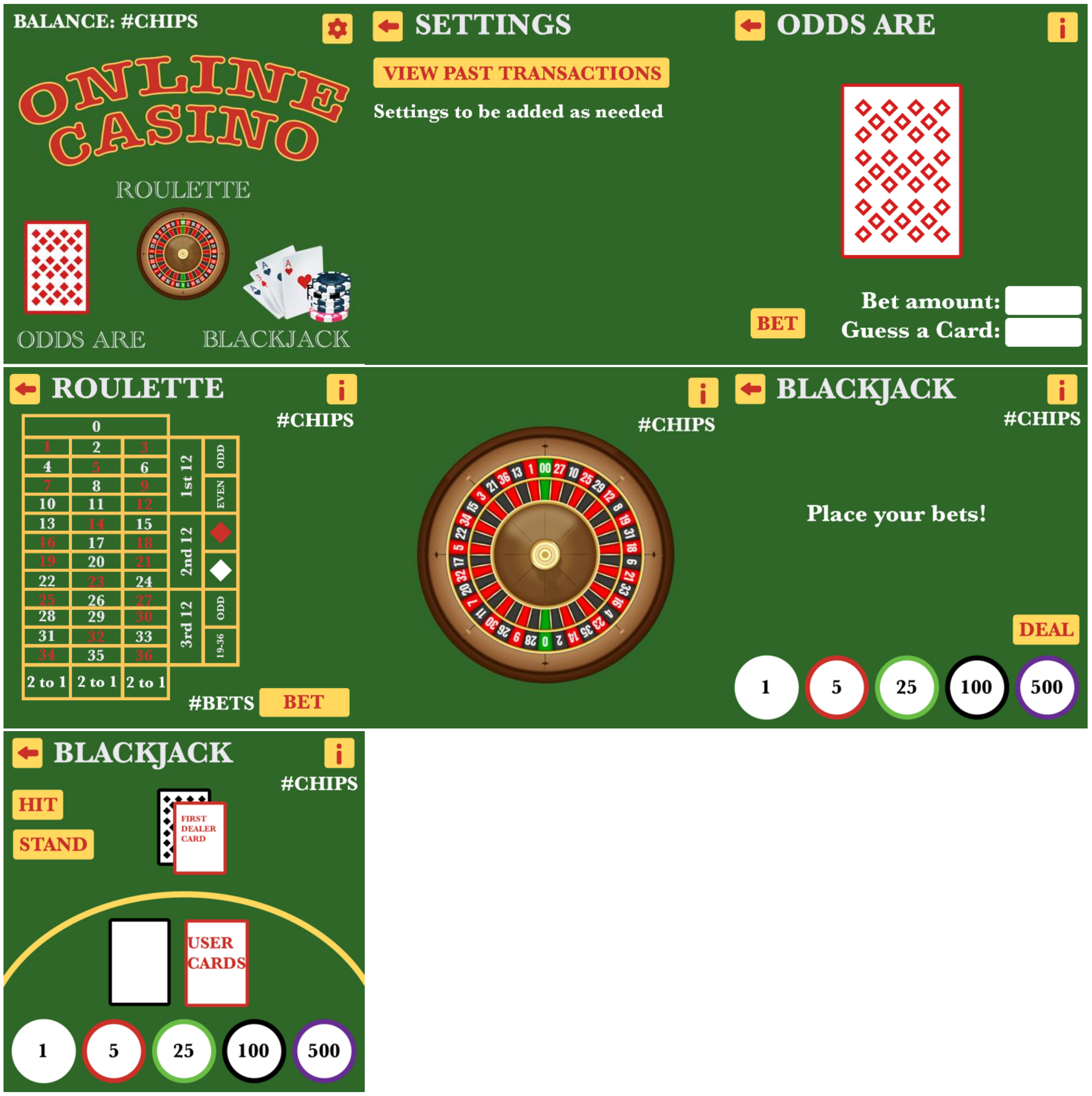


Figure 16: State chart diagram for oddsAre

Design Goals

The game Odds Are is a card guessing game where a player tries to guess what card the dealer has. The game is played with a regular deck of 52 cards; however, the deck is divided based on the level of difficulty chosen by the user. The player can enter how many cards they would like to guess from (i.e., 10 cards, 5 cards, 3 cards etc.). To be clear, each card is unique. For example, if the user chose a range of 3, the cards they would guess from are 2, 3 and 4 (This is specified on the instruction page for the game). Once the user chooses the range, that desired range of cards is generated. The user then has the capability to place an amount they wish to bet and guess a card number. Once the player places their guess, the system searches to see if that guess is correct. If the guess is correct, the player receives their payout based on the certain multiplier and is rewarded with a bonus multiplier if they wish to play again. If a player loses, they lose their bet credit and are not given any bonus multipliers. The score is determined by multiplying the bet by how many cards are in the deck if the player wins (i.e. $5 bet, 5 cards in the deck = 5*5=$25 payout).

GUI Screenshots



Settings to be added as needed

Place your bets!

Figure 17: GUI Screenshots.

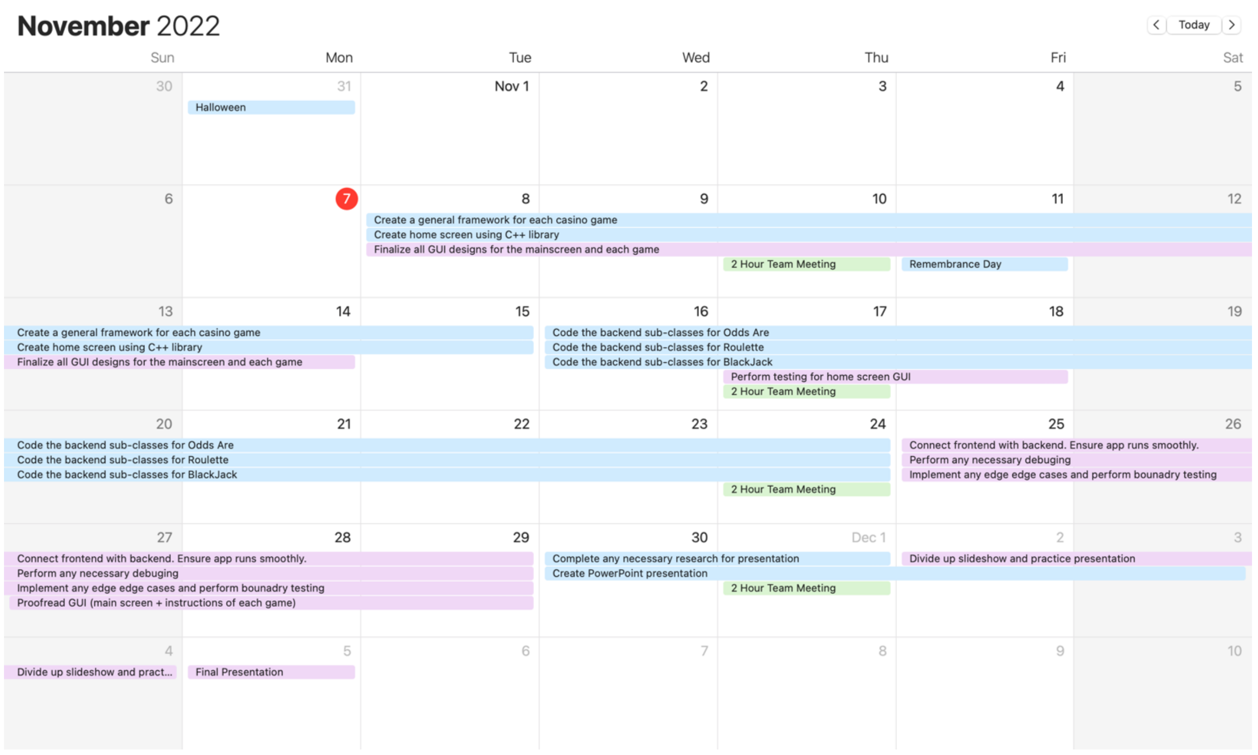
Coding Assignments

The following table shows how the code for this assignment will be divided:

Table 2: Coding Assignments.

Name	**Assignment**
Henry & Ethan	GUI for the entire application, including main screen (All front-end components to this assignment)
Cameron & Victor	All back-end development for the Blackjack casino game
Oscar & Graydon	All back-end development for the Roulette casino game
Abbey & Matthew	All back-end development for the Odds Are casino game

Timeline



November 2022

Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	31 Halloween	Nov 1	2	3	4	5
6	7	8 Create a general framework for each casino game Create home screen using C++ library Finalize all GUI designs for the mainscreen and each game	9	10 2 Hour Team Meeting	11 Remembrance Day	12
13 Create a general framework for each casino game Create home screen using C++ library Finalize all GUI designs for the mainscreen and each game	14	15	16 Code the backend sub-classes for Odds Are Code the backend sub-classes for Roulette Code the backend sub-classes for BlackJack	17 Perform testing for home screen GUI 2 Hour Team Meeting	18	19
20 Code the backend sub-classes for Odds Are Code the backend sub-classes for Roulette Code the backend sub-classes for BlackJack	21	22	23	24 2 Hour Team Meeting	25 Connect frontend with backend. Ensure app runs smoothly. Perform any necessary debuging Implement any edge edge cases and perform bounadry testing	26
27 Connect frontend with backend. Ensure app runs smoothly. Perform any necessary debuging Implement any edge edge cases and perform bounadry testing Proofread GUI (main screen + instructions of each game)	28	29	30 Complete any necessary research for presentation Create PowerPoint presentation	Dec 1 2 Hour Team Meeting	2 Divide up slideshow and practice presentation	3
4 Divide up slideshow and pract…	5 Final Presentation	6	7	8	9	10

Figure 18: Timeline for building & testing the application

Appendix

Entire Class Diagram

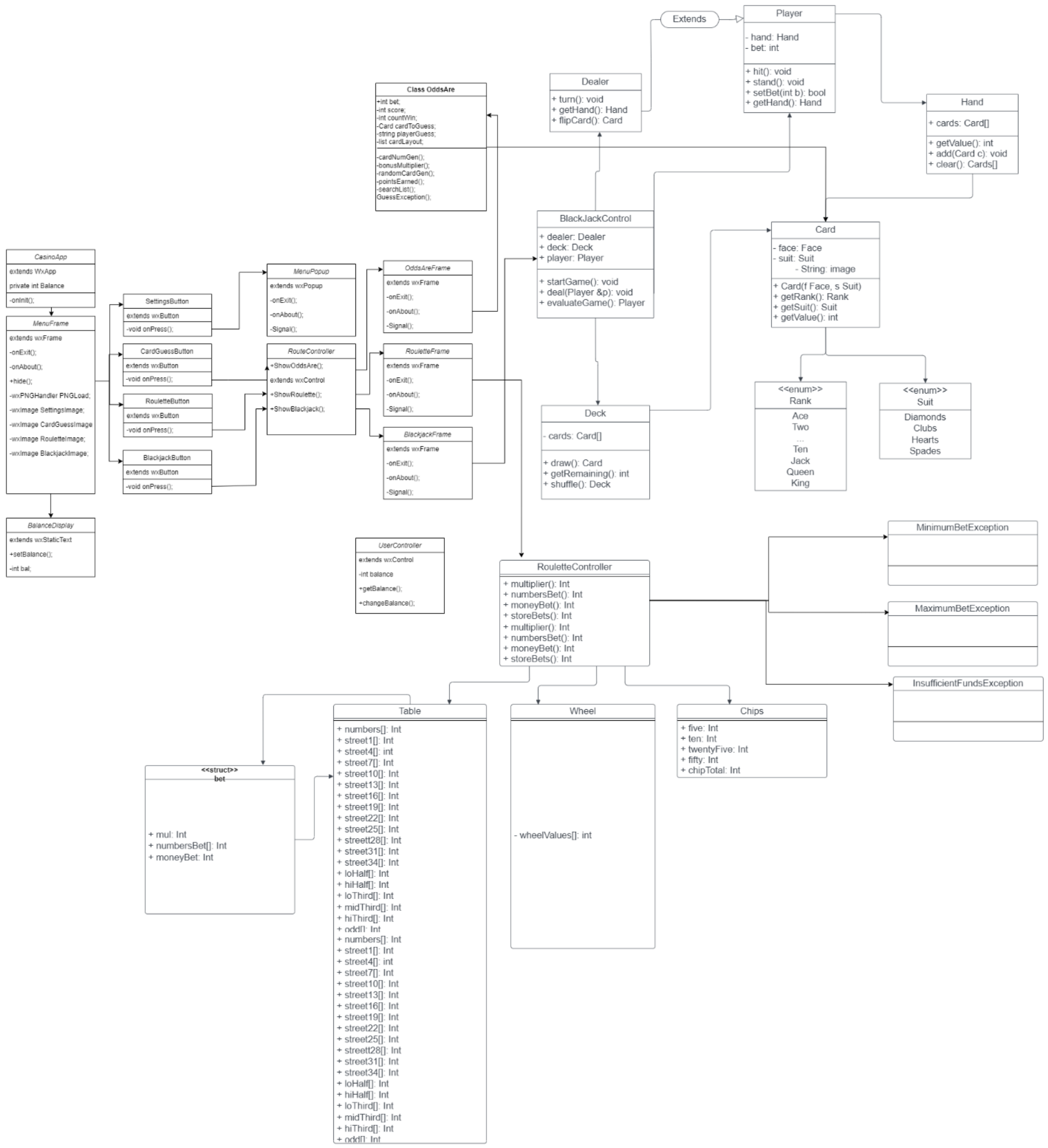


Figure 19: Class Diagram for the entire application.