Github Repository

https://github.com/GitUser807/Lylecisc191project

Project Presentation: Blackjack With a Twist Game

Week 1: Project Proposal

---

1. Planned Working Time

- **Working Days:** Friday - Sunday
- **Working Hours:** 10 AM - 2pm

---

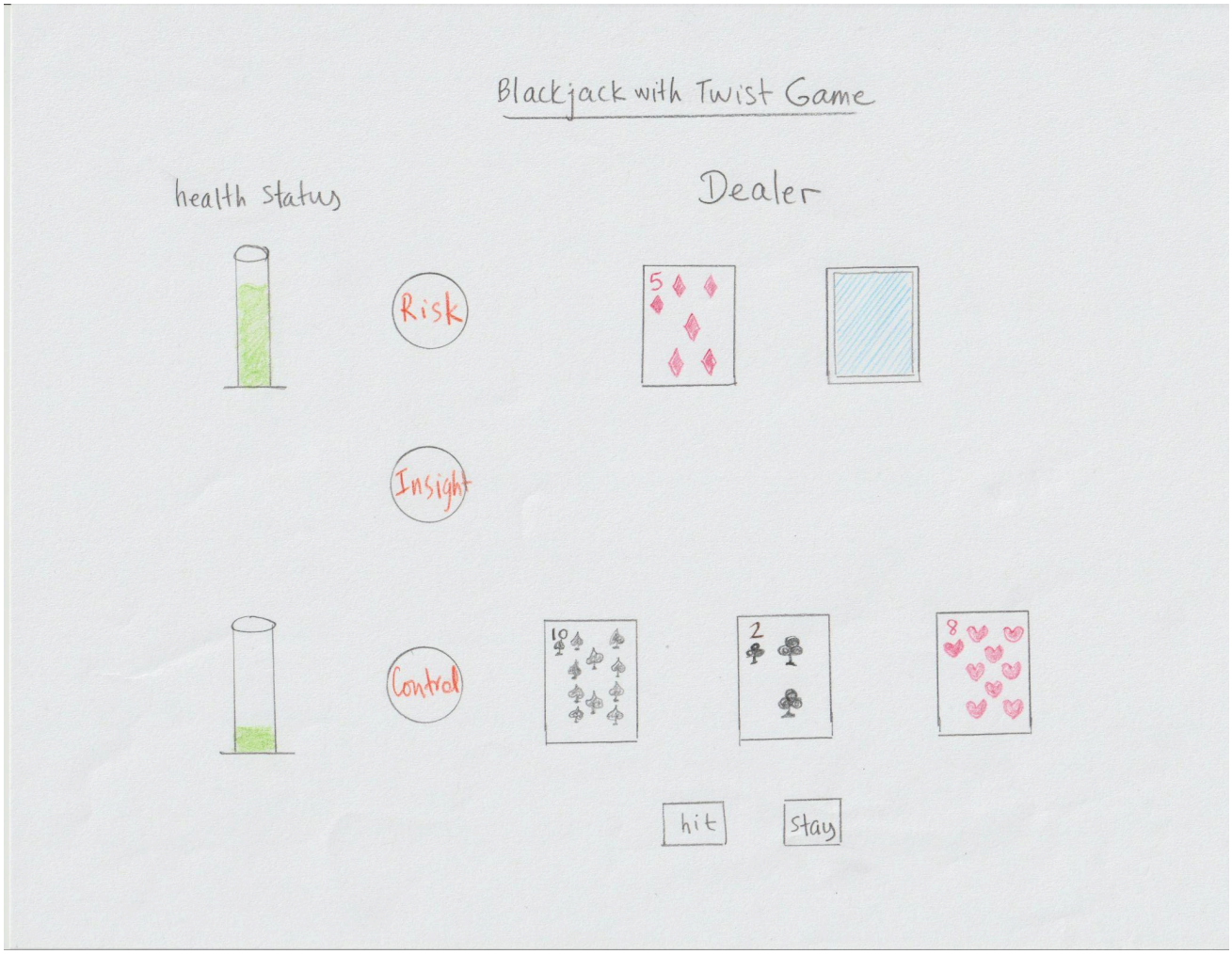
2. Project Pitch

Overview

Blackjack is one of the most popular card games, and the goal is to have a hand value closer to 21 than the dealer without exceeding it. In this project, I will create a playable Blackjack game with a graphical user interface (GUI), where a user can play against the dealer. I decided to base my game off of it but it has a twist. Both the player and the dealer have a health bar and whoever reaches 0 health first loses the game. The dealer will alternate turns as though you are against a non-dealer player. Both the dealer and the player will have multiple turns with each of their turns ending when either choose to hit or stay respectively. The round will end only when both the player and the dealer consecutively choose to stay on back-to-back turns. Additionally, whoever wins the round gets a power card. This power card can have different effects based on the type you want.

The game will include the following main features:

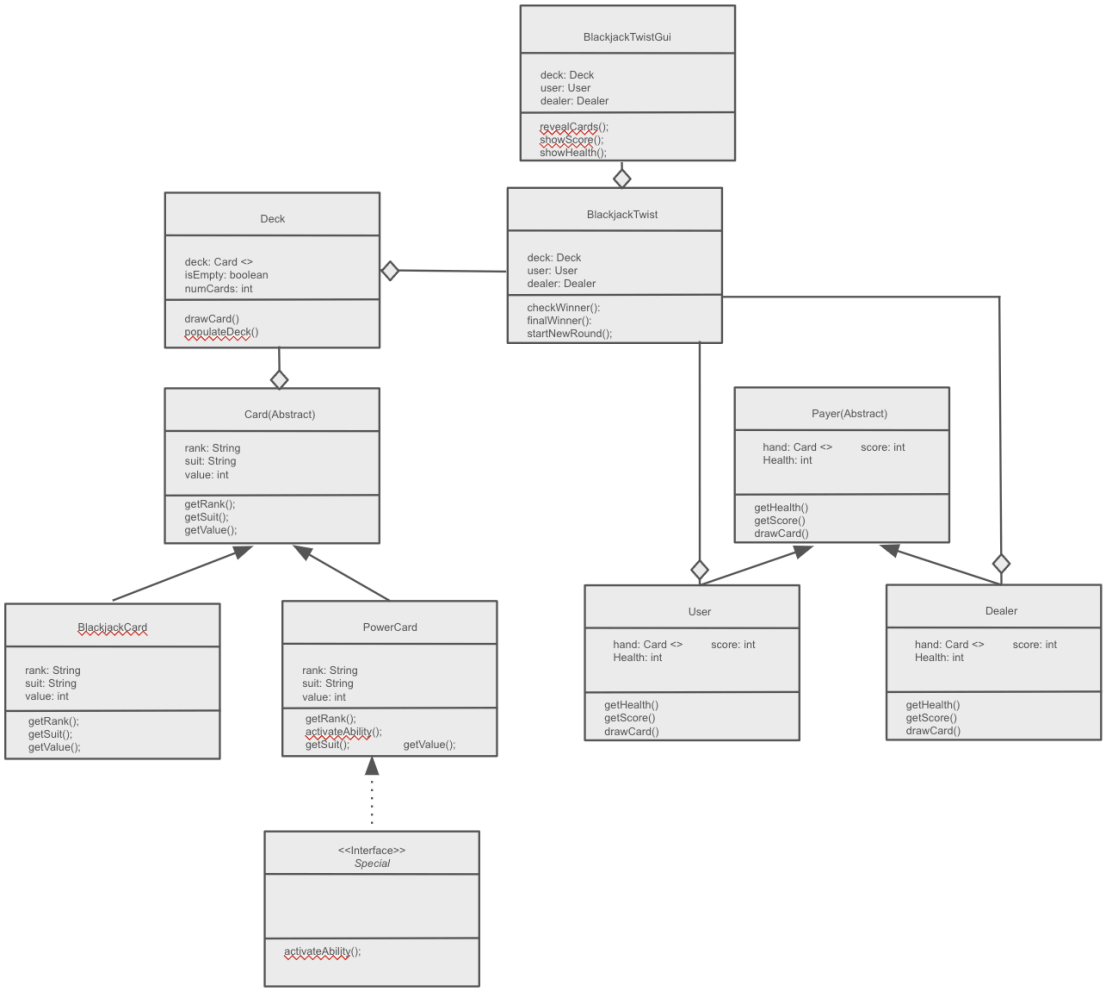
- **Card Dealing:** The player and dealer will each receive two cards at the beginning with the first card in each hand being visible.
- **Hit or Stand:** The player will be given the option to hit (draw a card) or stay (keep their hand).
- **Dealer's Rules:** The dealer must keep choosing hit until they have 17 or more points.
- **Blackjack or Bust:** The game will automatically check for a Blackjack (21 points) or bust (going over 21) and determine the winner.
- **Power Card:** A power card of the winner choice will permanently stay in the winner hand until they use it.



The GUI will have two health bars representing the respective health level of the player and dealer. The Player uses two buttons to indicate whether they want to hit or stay during the game. There are also 3 buttons that pop up when the player wins a round where the player can choose to activate an ability for future rounds.

---

3. UML Diagram Overview



UML Breakdown:

1. **Core Game Classes:**
    - **Card** class to represent a card in the deck.
    - **Deck** class to represent a deck of cards.
    - **Player** class (abstract) to represent both the user and the dealer.
    - **User** class (inherits Player) to represent the player.
    - **Dealer** class (inherits Player) to represent the dealer.
    - **BlackjackGame** class to manage the game state (dealing cards, determining winners).
    - **BlackjackGUI** class to handle user interaction via the graphical interface.

CRC Cards:

Applying CRC Cards

CRC Card for Card:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Deck • Player • BlackjackCard • PowerCard

Applying CRC Cards

CRC Card for BlackjackCard:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Card

Applying CRC Cards

CRC Card for PowerCard:

Responsibilities	Collaborators
• Know the suit(category/vague description) • Know the rank(ability) • Know the value (0)	• Card

Applying CRC Cards

CRC Card for Special(Interface):

Responsibilities	Collaborators
• Generate ability for power cards	• PowerCard

Applying CRC Cards

CRC Card for Deck:

Responsibilities	Collaborators
• Populate the deck of cards • Deal a card	• Card • BlackjackTwist

Applying CRC Cards

CRC Card for Player(abstract):

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• User • Dealer • BlackjackTwist

Applying CRC Cards

CRC Card for User:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for Dealer:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the Dealer • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for BlackjackTwist:

Responsibilities	Collaborators
• Know the rules of the game • Deal cards to players • Determine winner • Be able to start new game	• BlackjackTwistGui • Deck • User • Dealer

Applying CRC Cards

CRC Card for BlackJackTwistGui:

Responsibilities	Collaborators
• Show the interface of the game • Have buttons that do certain things • Display health	• BlackJackTwist

1. Design Principles of Object-Oriented Programming (LO1)

I will use object-oriented programming principles like, abstraction, and inheritance in the Blackjack game. Additionally, I will create classes for the Card, Deck, Player, and Black Jack game.

2. Use Single and Multidimensional Arrays (LO2)

I'll be using array lists for holding the cards in the deck and players' hands. A multidimensional array could be used to hold the game history, such as the player 's previous actions.

3. Object and Classes, Including Aggregation (LO3)

There will be multiple classes:

- Card (to represent a single playing card)
- Deck (to represent the deck of cards)
- Player (to represent the user and dealer, holding hands and score)
- BlackJackTwist this will contain the rules of the game and make sure it is that each round will follow the specified rules

I will use aggregation where the `Deck` class contains a collection of `Card` objects, and a `Player` object has a collection of `Card` objects.

4. Inheritance and Polymorphism (LO4)

- Player and Card will both be abstract and super class of other classes
- EX: User and Dealer will be subclasses of Player
- EX: BlackjackCard and PowerCard will be a subclass of Card

5. Generic Collections and Data Structures (LO5)

I will be using generic collections, such as array lists and stacks, to store cards in the deck and the players' hand.

6. Graphical User Interface (LO6)

I will use GUI for buttons, the background, the health bar, and cards in the users' hand

7. Exception Handling (LO7)

I'll use exception handling to ensure that the game handles invalid inputs. For example the user enters something other than "hit" or "stand" or somehow gets a blank power card and to manage situations like an empty deck or out-of-bounds issues.

8. Text File I/O (LO8)

We'll store game results like player's scores in a text file and read from it to load previous scores when starting the game.

Week 1	● Write the project proposal. ● Plan the object-oriented design, starting with CRC cards. Determine classes (with fields and methods) and interfaces and their responsibilities. (The "model" part of MVC.) ● Create the UML. ● Begin writing project page.
Week 2	● Write code for The Card, BlackjackCard, PowerCard, and Deck Class. ● Develop test cases and test code as it is written and comment on my current code. ● Update project page with progress details. ● Submit code written so far.
Week 3	● Finish writing all my classes except for the Gui. ● Plan where exception handling is needed to ensure the program fails gracefully. ● Design the GUI (sketch it out on paper) - include the design in the Weekly update! ● Update project page with progress details. ● Submit code written so far.
Week 4	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 5	● Write code to create a non-functional GUI (the "view" part of MVC). ● Update project page with progress details. ● Submit code written so far.

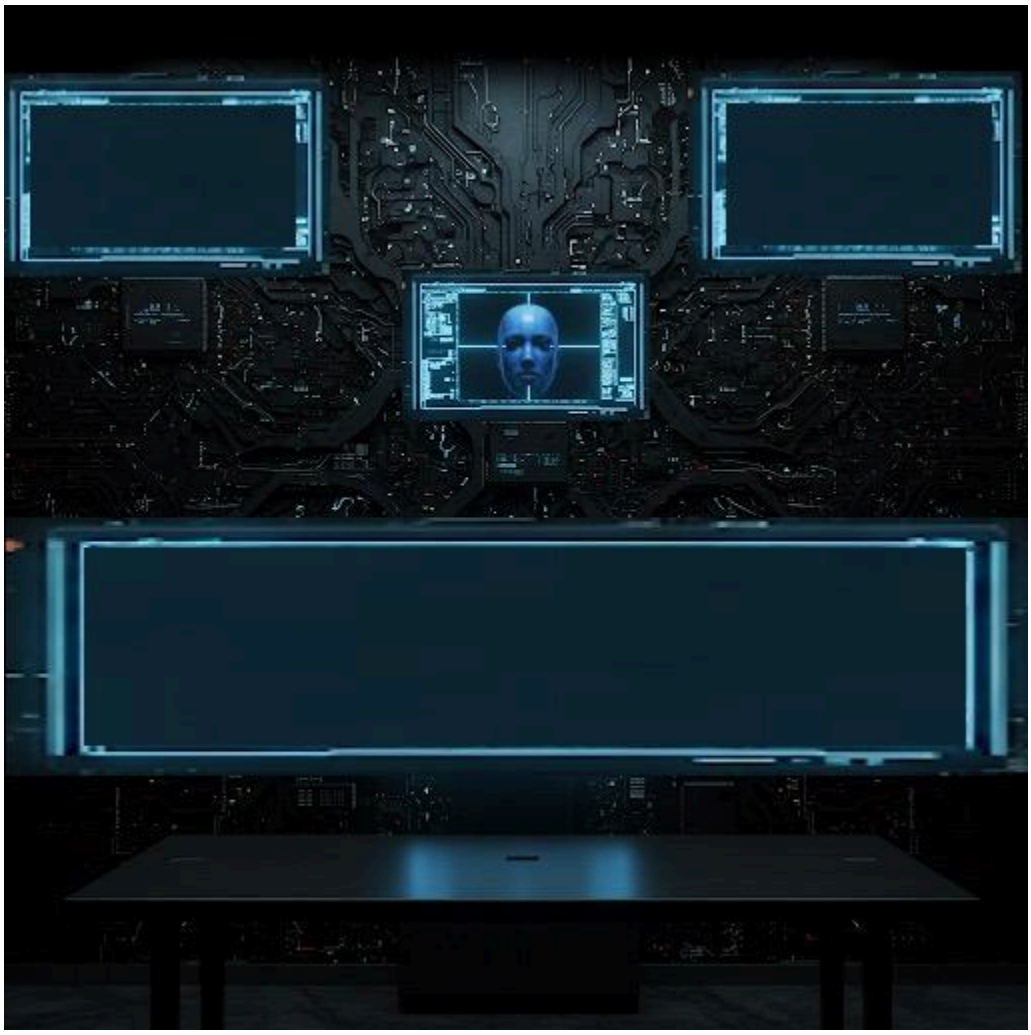
Week 6	● Add event handling to make the GUI functional (the "controller" part of MVC). ● Update project page with progress details. ● Submit code written so far.
Week 7	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 8	● Debug any remaining problems. ● Create project demonstration video, including information about how each LO is used as part of the project. ● Submit final code on Canvas, and add videos to the project page.

Github Repository

https://github.com/GitUser807/Lylecisc191project

Week 2: Updates

Gui Screenshot:



Week 2 Journal Entry:

Writing code this week has been fun. I have been writing the code for my Card, PowerCard, BlackjackCard, Deck, and Special class. When writing for these classes I had to think about how the cards and deck really worked. All standard playing cards have 3 different categories making all 52 cards in a deck. The three things are the suit,rank, and value. So for my cards I implemented these 3 things as instance variables. Since we have our 3 field variables we need to add a corresponding setter and getter methods for these field variables. After writing that I had to think about how I wanted to implement my power card. I want my power card to not add to the players score so I decided for it to have a value of 0. Before the user selects the category

of ability they want. The suit and rank will be Special and None respectively. After the ability is generated the suit and rank hold the category type of the ability and the ability description respectively. After writing for all of these classes I put a little bit more thought into how I want the game to be. This resulted in making some changes to the game.

1st Change

- The player will be given the option to hit (draw a card) or stay (keep their hand). The player can draw a maximum of 12 cards. This prevents the user from drawing an absurd amount of cards because by these rules they can theoretically have an infinite amount of cards in their hand. This will also be implemented to discourage the user from holding on to a lot of power cards since they are carried over from round to round.

2nd Change

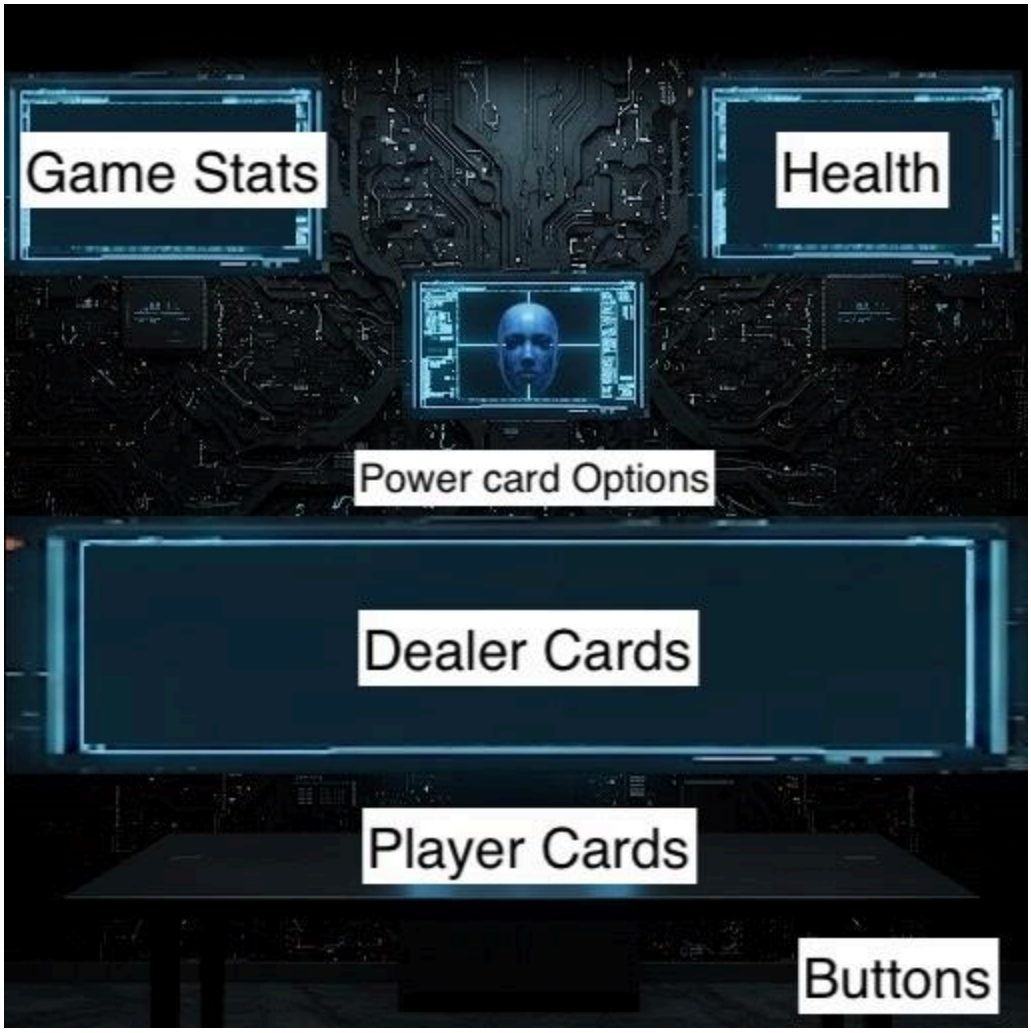
- The dealer must keep choosing hit until they have 17 or more points. The dealer also wins ties to compensate for the fact that they don't get power cards.

3rd Change

- 21 is the best score you can get and you can keep drawing cards even if you bust(not recommended).

4th Change

- The Gui layout will look something like this now:



Week 2 Update Project Presentation: Blackjack With a Twist Game

Week 2: Updated Project Proposal

---

1. Planned Working Time

- **Working Days:** Friday - Sunday

- **Working Hours:** 10 AM - 2pm

---

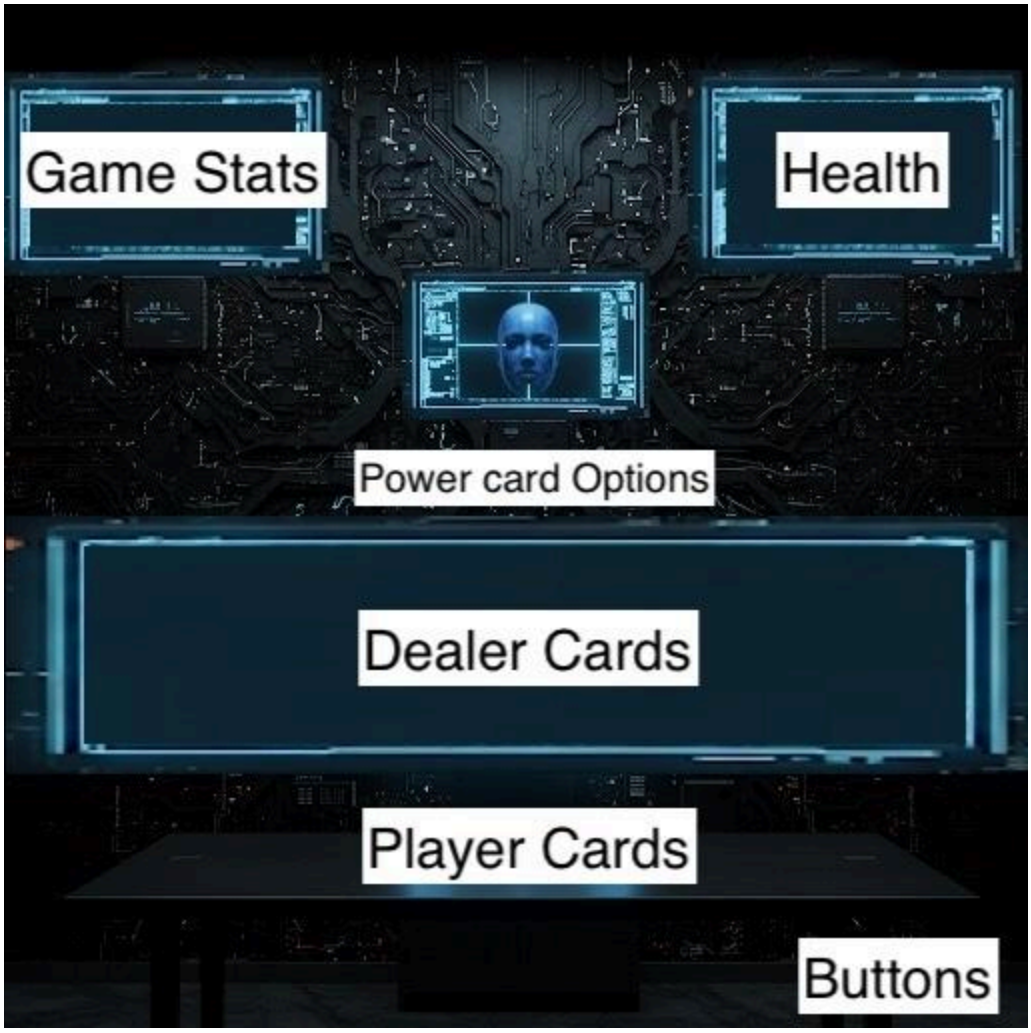
2. Project Pitch

Overview

Blackjack is one of the most popular card games, and the goal is to have a hand value closer to 21 than the dealer without exceeding it. In this project, I will create a playable Blackjack game with a graphical user interface (GUI), where a user can play against the dealer. I decided to base my game off of it but it has a twist. Both the player and the dealer have a health bar and whoever reaches 0 health first loses the game. The dealer will alternate turns as though you are against a non-dealer player. Both the dealer and the player will have multiple turns with each of their turns ending when either choose to hit or stay respectively. The round will end only when both the player and the dealer consecutively choose to stay on back-to-back turns. Additionally, if the player wins the round, they get a power card. This power card can have different effects based on the type you want.

The game will include the following main features:

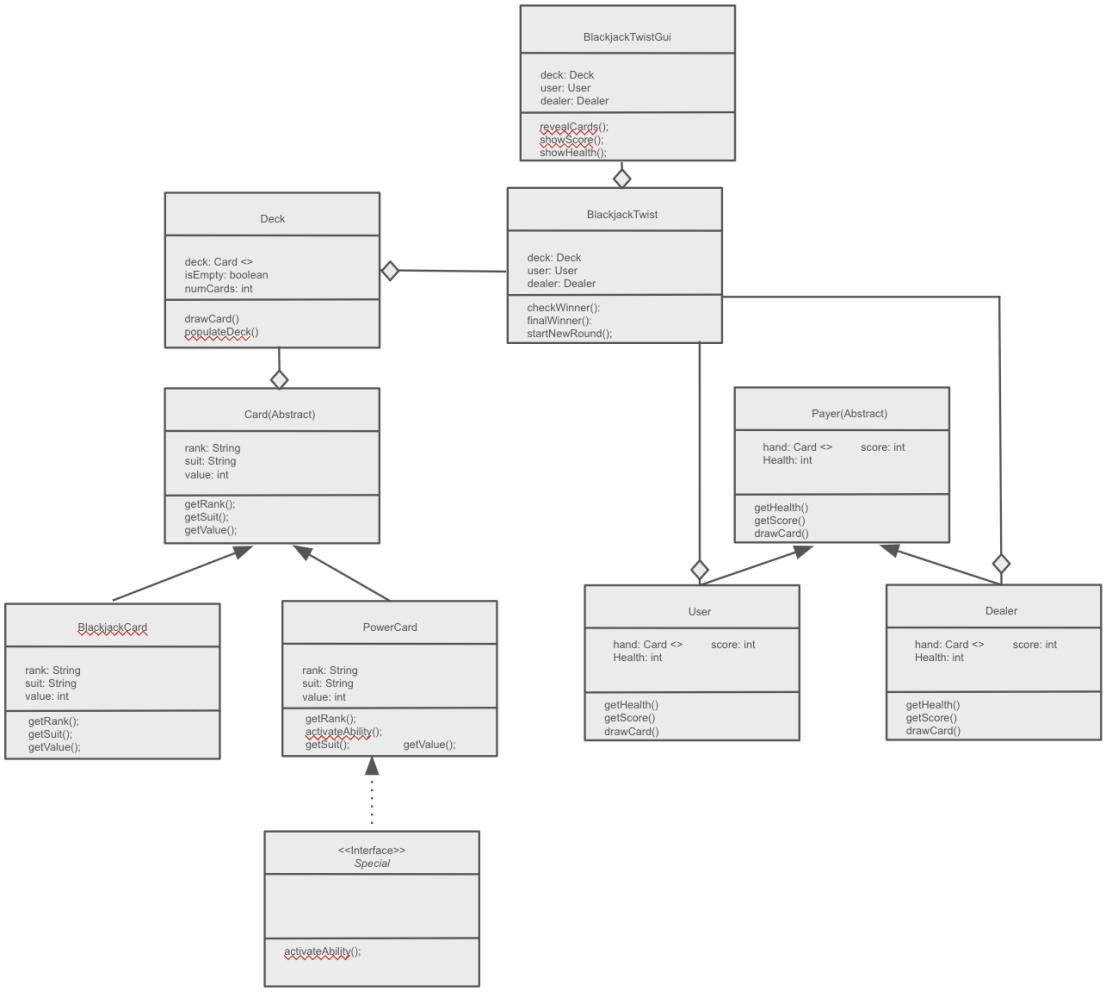
- **Card Dealing:** The player and dealer will each receive two cards at the beginning with the first card in each hand being visible.
- **Hit or Stand:** The player will be given the option to hit (draw a card) or stay (keep their hand). The player can draw a maximum of 12 cards. This prevents the user from drawing an absurd amount of cards because by these rules they can theoretically have an infinite amount of cards in their hand. This will also be implemented to discourage the user from holding on to a lot of power cards since they are carried over from round to round.
- **Dealer's Rules:** The dealer must keep choosing hit until they have 17 or more points. The dealer also wins ties to compensate for the fact that they don't get power cards.
- **Blackjack or Bust:** 21 is the best score you can get and you can keep drawing cards even if you bust(not recommended).
- **Power Card:** A power card of the winner choice will permanently stay in the winner hand until they use it.



The GUI will have two health bars representing the respective health level of the player and dealer. The Player uses two buttons to indicate whether they want to hit or stay during the game. There are also 3 buttons that pop up when the player wins a round where the player can choose to activate an ability for future rounds.

---

3. UML Diagram Overview



UML Breakdown:

1. **Core Game Classes:**
    - **Card** class to represent a card in the deck.
    - **Deck** class to represent a deck of cards.
    - **Player** class (abstract) to represent both the user and the dealer.
    - **User** class (inherits Player) to represent the player.
    - **Dealer** class (inherits Player) to represent the dealer.
    - **BlackjackGame** class to manage the game state (dealing cards, determining winners).
    - **BlackjackGUI** class to handle user interaction via the graphical interface.

CRC Cards:

Applying CRC Cards

CRC Card for Card:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Deck • Player • BlackjackCard • PowerCard

Applying CRC Cards

CRC Card for BlackjackCard:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Card

Applying CRC Cards

CRC Card for PowerCard:

Responsibilities	Collaborators
• Know the suit(category/vague description) • Know the rank(ability) • Know the value (0)	• Card

Applying CRC Cards

CRC Card for Special(Interface):

Responsibilities	Collaborators
• Generate ability for power cards	• PowerCard

Applying CRC Cards

CRC Card for Deck:

Responsibilities	Collaborators
• Populate the deck of cards • Deal a card	• Card • BlackjackTwist

Applying CRC Cards

CRC Card for Player(abstract):

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• User • Dealer • BlackjackTwist

Applying CRC Cards

CRC Card for User:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for Dealer:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the Dealer • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for BlackjackTwist:

Responsibilities	Collaborators
• Know the rules of the game • Deal cards to players • Determine winner • Be able to start new game	• BlackjackTwistGui • Deck • User • Dealer

Applying CRC Cards

CRC Card for BlackJackTwistGui:

Responsibilities	Collaborators
• Show the interface of the game • Have buttons that do certain things • Display health	• BlackJackTwist

1. Design Principles of Object-Oriented Programming (LO1)

I will use object-oriented programming principles like, abstraction, and inheritance in the Blackjack game. Additionally, I will create classes for the Card, Deck, Player, and Black Jack game.

2. Use Single and Multidimensional Arrays (LO2)

I'll be using array lists for holding the cards in the deck and players' hands. A multidimensional array could be used to hold the game history, such as the player 's previous actions.

3. Object and Classes, Including Aggregation (LO3)

There will be multiple classes:

- Card (to represent a single playing card)
- Deck (to represent the deck of cards)
- Player (to represent the user and dealer, holding hands and score)
- BlackJackTwist this will contain the rules of the game and make sure it is that each round will follow the specified rules

I will use aggregation where the `Deck` class contains a collection of `Card` objects, and a `Player` object has a collection of `Card` objects.

4. Inheritance and Polymorphism (LO4)

- Player and Card will both be abstract and super class of other classes
- EX: User and Dealer will be subclasses of Player
- EX: BlackjackCard and PowerCard will be a subclass of Card

5. Generic Collections and Data Structures (LO5)

I will be using generic collections, such as array lists and stacks, to store cards in the deck and the players' hand.

6. Graphical User Interface (LO6)

I will use GUI for buttons, the background, the health bar, and cards in the users' hand

7. Exception Handling (LO7)

I'll use exception handling to ensure that the game handles invalid inputs. For example the user enters something other than "hit" or "stand" or somehow gets a blank power card and to manage situations like an empty deck or out-of-bounds issues.

8. Text File I/O (LO8)

We'll store game results like player's scores in a text file and read from it to load previous scores when starting the game.

Week 1	● Write the project proposal. ● Plan the object-oriented design, starting with CRC cards. Determine classes (with fields and methods) and interfaces and their responsibilities. (The "model" part of MVC.) ● Create the UML. ● Begin writing project page.
Week 2	● Write code for The Card, BlackjackCard, PowerCard, and Deck Class. ● Develop test cases and test code as it is written and comment on my current code. ● Update project page with progress details. ● Submit code written so far.
Week 3	● Finish writing all my classes except for the Gui. ● Plan where exception handling is needed to ensure the program fails gracefully. ● Design the GUI (sketch it out on paper) - include the design in the Weekly update! ● Update project page with progress details. ● Submit code written so far.
Week 4	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 5	● Write code to create a non-functional GUI (the "view" part of MVC). ● Update project page with progress details. ● Submit code written so far.

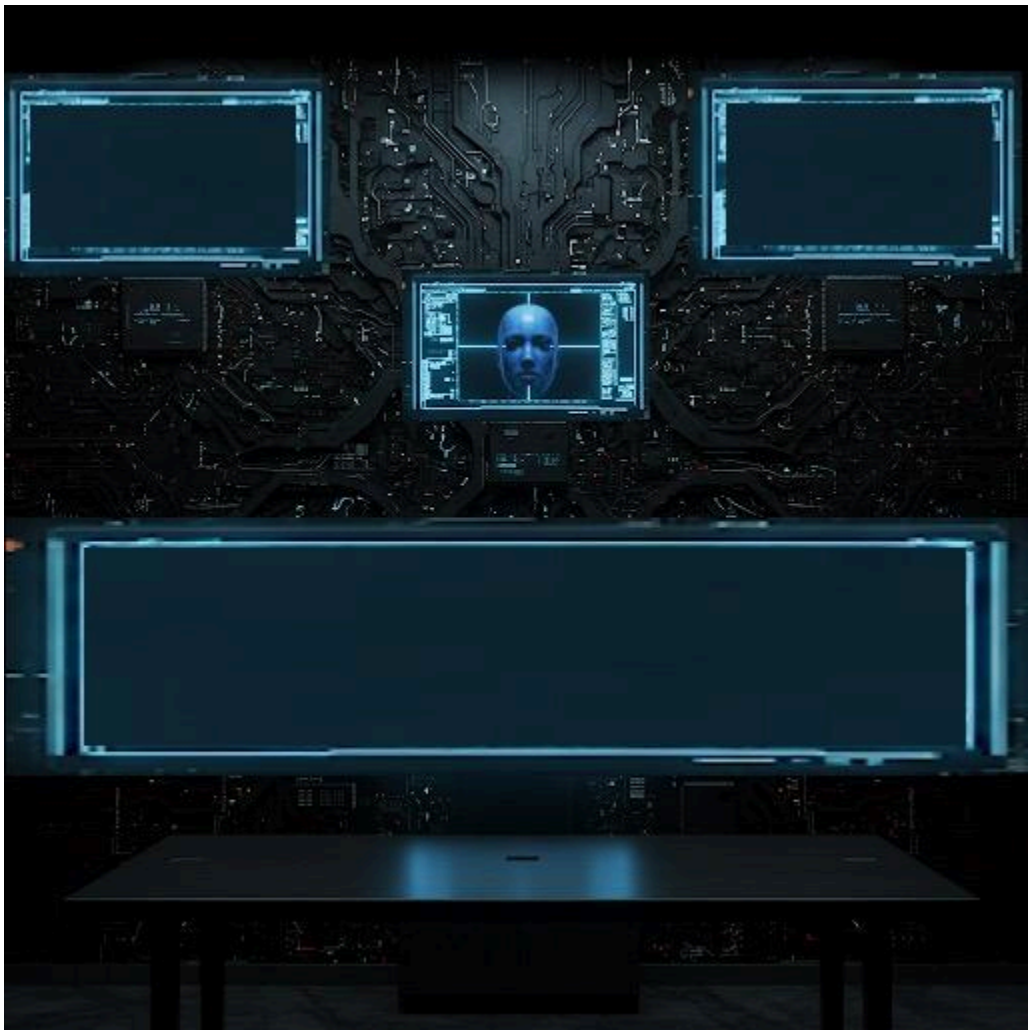
Week 6	● Add event handling to make the GUI functional (the "controller" part of MVC). ● Update project page with progress details. ● Submit code written so far.
Week 7	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 8	● Debug any remaining problems. ● Create project demonstration video, including information about how each LO is used as part of the project. ● Submit final code on Canvas, and add videos to the project page.

Github Repository

https://github.com/GitUser807/Lylecisc191project

Week 3: Updates

Gui Screenshot:



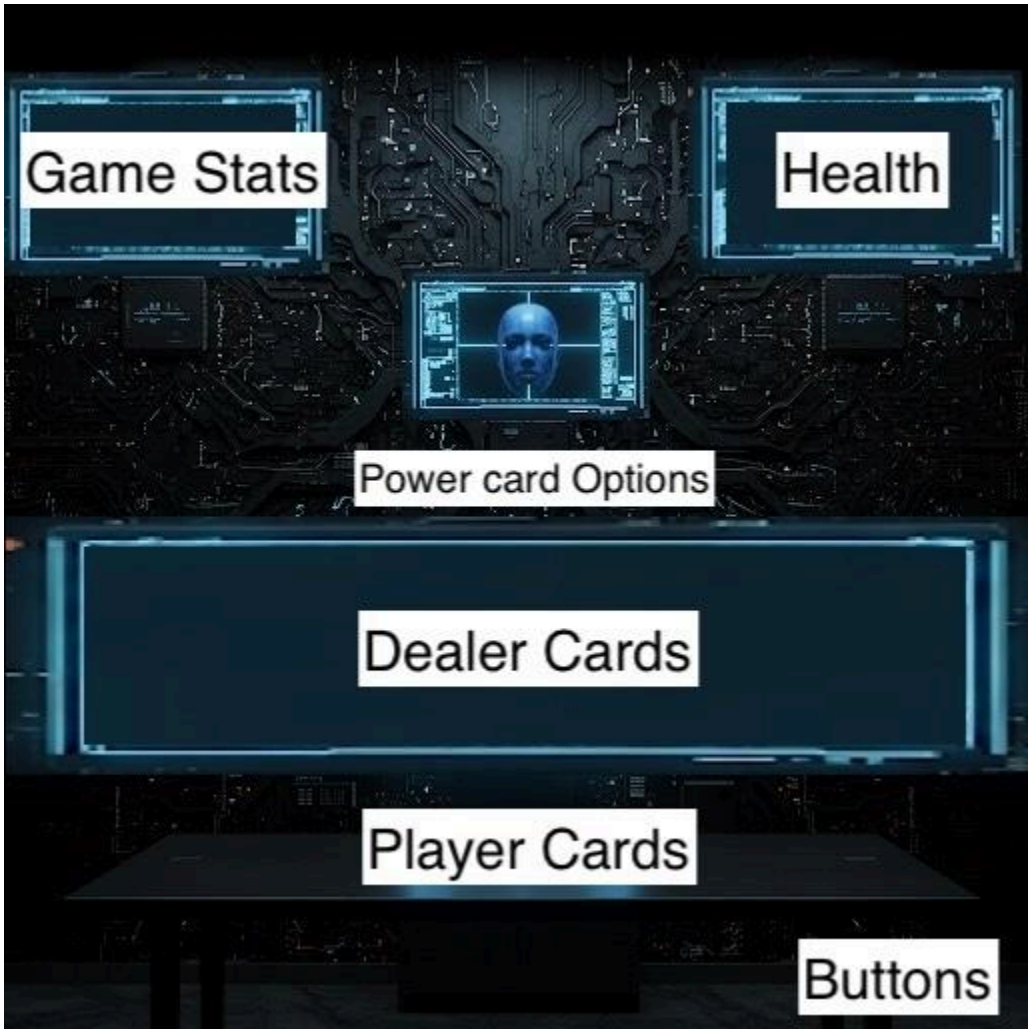
Week 3 Journal Entry:

Writing more code for my project has been enjoyable for me. I have been working on the code for my Player, Dealer, User, and BlackjackTwist class. When writing for the player class I had to think about what makes a person a player in a Black jack game. They have a hand, a score based on their hand, health, and a name. They should also be able to draw a card from the deck. With this I was able to write this class as well as the Dealer and User Class which extends the abstract class Player. The User class is different from the player class because the User has a max amount of cards the player can hold. The Dealer class is different from the player class because a person is not making the decisions for the dealer and they must keep hitting until

they reach at least a score of 17. Based on these differences I added and adjusted the methods accordingly. Next is the BlackjackTwistGame class which contains the logic of the game. While I “finished” writing this class, I could still change up a few things in this class based on how I Design the GUI. I have also decided to work on exceptions during week 4 instead of week 3 because at the time I didn't think I had a correct understanding of exceptions and how to properly implement them. Now that I have completed Module 8 I believe I will be able to properly write and use exceptions in my code. So I have decided to change my project page and move back the implementing exceptions for week4. Working on this project has been fun and I look forward to writing more code for this project.

Change

- Implementing Exception has been moved to week 4



Week 3 Update Project Presentation: Blackjack With a Twist Game

Week 2: Updated Project Proposal

---

1. Planned Working Time

- **Working Days:** Friday - Sunday
- **Working Hours:** 10 AM - 2pm

---

2. Project Pitch

Overview

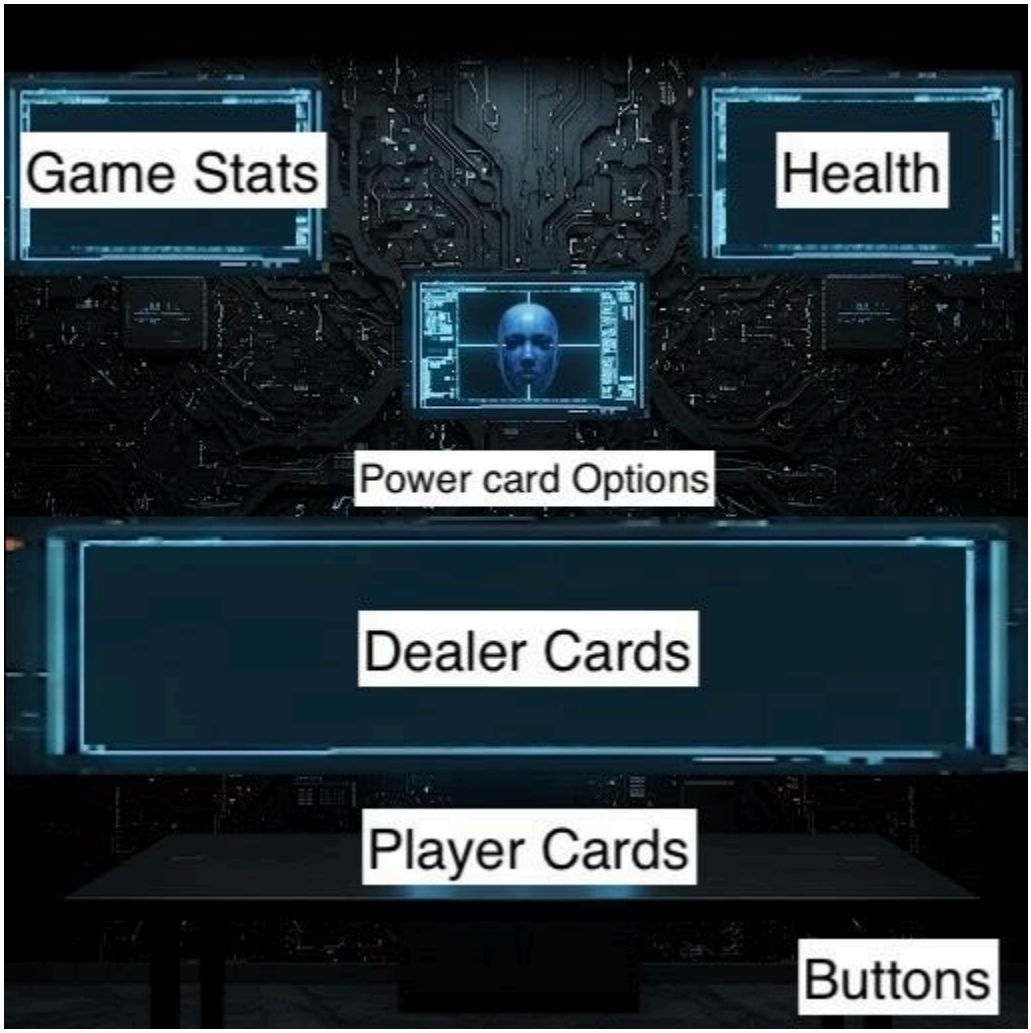
Blackjack is one of the most popular card games, and the goal is to have a hand value closer to 21 than the dealer without exceeding it. In this project, I will create a playable Blackjack game with a graphical user interface (GUI), where a user can play against the dealer. I decided to base my game off of it but it has a twist. Both the player and the dealer have a health bar and whoever reaches 0 health first loses the game. The dealer will alternate turns as though you are against a non-dealer player. Both the dealer and the player will have multiple turns with each of their turns ending when either choose to hit or stay respectively. The round will end only when both the player and the dealer consecutively choose to stay on back-to-back turns. Additionally, if the player wins the round, they get a power card. This power card can have different effects based on the type you want.

The game will include the following main features:

- **Card Dealing:** The player and dealer will each receive two cards at the beginning with the first card in each hand being visible.
- **Hit or Stand:** The player will be given the option to hit (draw a card) or stay (keep their hand). The player can draw a maximum of 12 cards. This prevents the user from drawing an absurd amount of cards because by these rules they can theoretically have an infinite amount of cards in their hand. This will also be implemented to discourage the

user from holding on to a lot of power cards since they are carried over from round to round.

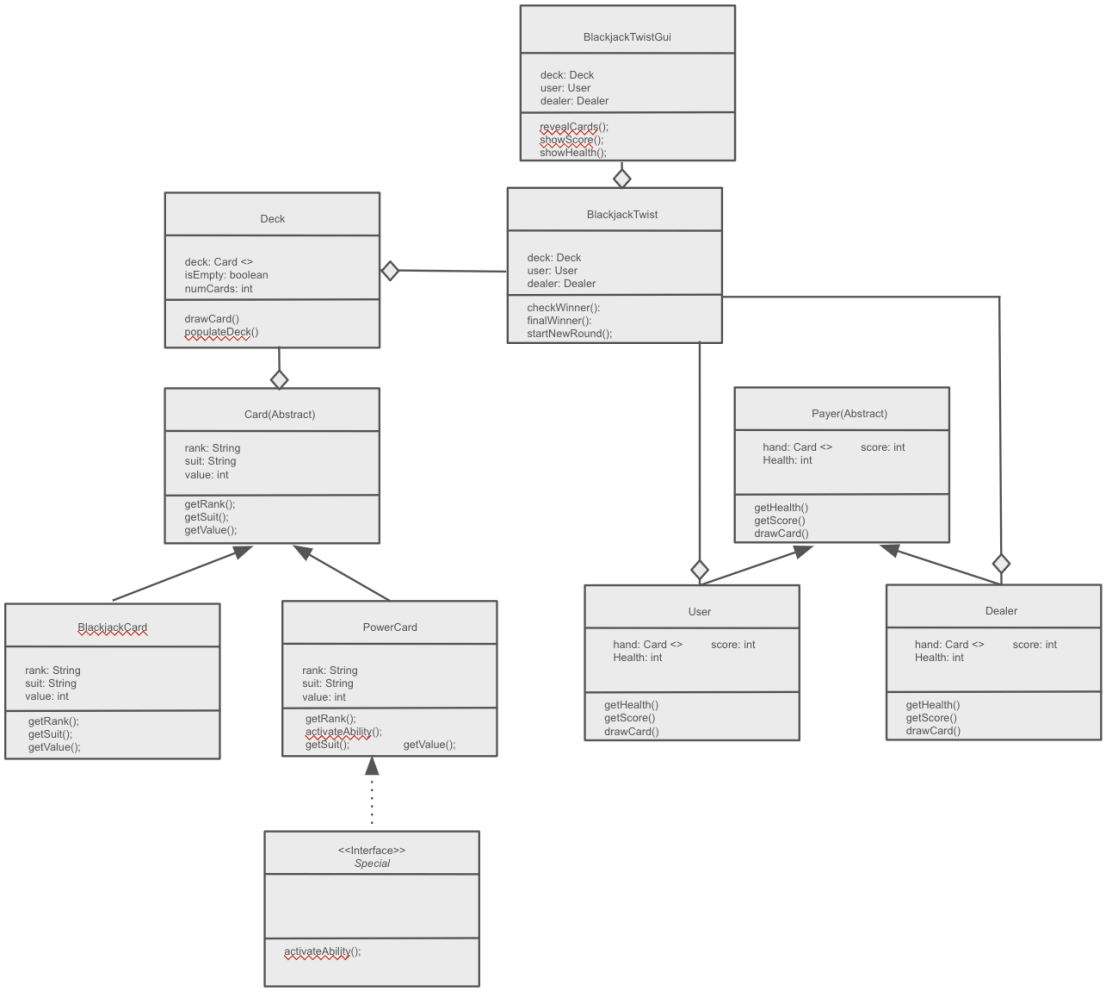
- **Dealer's Rules:** The dealer must keep choosing hit until they have 17 or more points. The dealer also wins ties to compensate for the fact that they don't get power cards.
- **Blackjack or Bust:** 21 is the best score you can get and you can keep drawing cards even if you bust(not recommended).
- **Power Card:** A power card of the winner choice will permanently stay in the winner hand until they use it.



The GUI will have two health bars representing the respective health level of the player and dealer. The Player uses two buttons to indicate whether they want to hit or stay during the game. There are also 3 buttons that pop up when the player wins a round where the player can choose to activate an ability for future rounds.

---

3. UML Diagram Overview



UML Breakdown:

2. **Core Game Classes:**
    - **Card** class to represent a card in the deck.
    - **Deck** class to represent a deck of cards.
    - **Player** class (abstract) to represent both the user and the dealer.
    - **User** class (inherits Player) to represent the player.
    - **Dealer** class (inherits Player) to represent the dealer.
    - **BlackjackGame** class to manage the game state (dealing cards, determining winners).
    - **BlackjackGUI** class to handle user interaction via the graphical interface.

CRC Cards:

Applying CRC Cards

CRC Card for Card:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Deck • Player • BlackjackCard • PowerCard

Applying CRC Cards

CRC Card for BlackjackCard:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Card

Applying CRC Cards

CRC Card for PowerCard:

Responsibilities	Collaborators
• Know the suit(category/vague description) • Know the rank(ability) • Know the value (0)	• Card

Applying CRC Cards

CRC Card for Special(Interface):

Responsibilities	Collaborators
• Generate ability for power cards	• PowerCard

Applying CRC Cards

CRC Card for Deck:

Responsibilities	Collaborators
• Populate the deck of cards • Deal a card	• Card • BlackjackTwist

Applying CRC Cards

CRC Card for Player(abstract):

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• User • Dealer • BlackjackTwist

Applying CRC Cards

CRC Card for User:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for Dealer:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the Dealer • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for BlackjackTwist:

Responsibilities	Collaborators
• Know the rules of the game • Deal cards to players • Determine winner • Be able to start new game	• BlackjackTwistGui • Deck • User • Dealer

Applying CRC Cards

CRC Card for BlackJackTwistGui:

Responsibilities	Collaborators
• Show the interface of the game • Have buttons that do certain things • Display health	• BlackJackTwist

1. Design Principles of Object-Oriented Programming (LO1)

I will use object-oriented programming principles like, abstraction, and inheritance in the Blackjack game. Additionally, I will create classes for the Card, Deck, Player, and Black Jack game.

2. Use Single and Multidimensional Arrays (LO2)

I'll be using array lists for holding the cards in the deck and players' hands. A multidimensional array could be used to hold the game history, such as the player 's previous actions.

3. Object and Classes, Including Aggregation (LO3)

There will be multiple classes:

- Card (to represent a single playing card)
- Deck (to represent the deck of cards)
- Player (to represent the user and dealer, holding hands and score)
- BlackJackTwist this will contain the rules of the game and make sure it is that each round will follow the specified rules

I will use aggregation where the `Deck` class contains a collection of `Card` objects, and a `Player` object has a collection of `Card` objects.

4. Inheritance and Polymorphism (LO4)

- Player and Card will both be abstract and super class of other classes
- EX: User and Dealer will be subclasses of Player
- EX: BlackjackCard and PowerCard will be a subclass of Card

5. Generic Collections and Data Structures (LO5)

I will be using generic collections, such as array lists and stacks, to store cards in the deck and the players' hand.

6. Graphical User Interface (LO6)

I will use GUI for buttons, the background, the health bar, and cards in the users' hand

7. Exception Handling (LO7)

I'll use exception handling to ensure that the game handles invalid inputs. For example the user enters something other than "hit" or "stand" or somehow gets a blank power card and to manage situations like an empty deck or out-of-bounds issues.

8. Text File I/O (LO8)

We'll store game results like player's scores in a text file and read from it to load previous scores when starting the game.

Week 1	● Write the project proposal. ● Plan the object-oriented design, starting with CRC cards. Determine classes (with fields and methods) and interfaces and their responsibilities. (The "model" part of MVC.) ● Create the UML. ● Begin writing project page.
Week 2	● Write code for The Card, BlackjackCard, PowerCard, and Deck Class. ● Develop test cases and test code as it is written and comment on my current code. ● Update project page with progress details. ● Submit code written so far.
Week 3	● Finish writing all my classes except for the Gui. ● Design the GUI (sketch it out on paper) - include the design in the Weekly update! ● Update project page with progress details. ● Submit code written so far.
Week 4	● Test, test, test, debug, and test some more. ● Plan where exception handling is needed to ensure the program fails gracefully. ● Update project page with progress details. ● Submit code written so far.
Week 5	● Write code to create a non-functional GUI (the "view" part of MVC). ● Update project page with progress details. ● Submit code written so far.

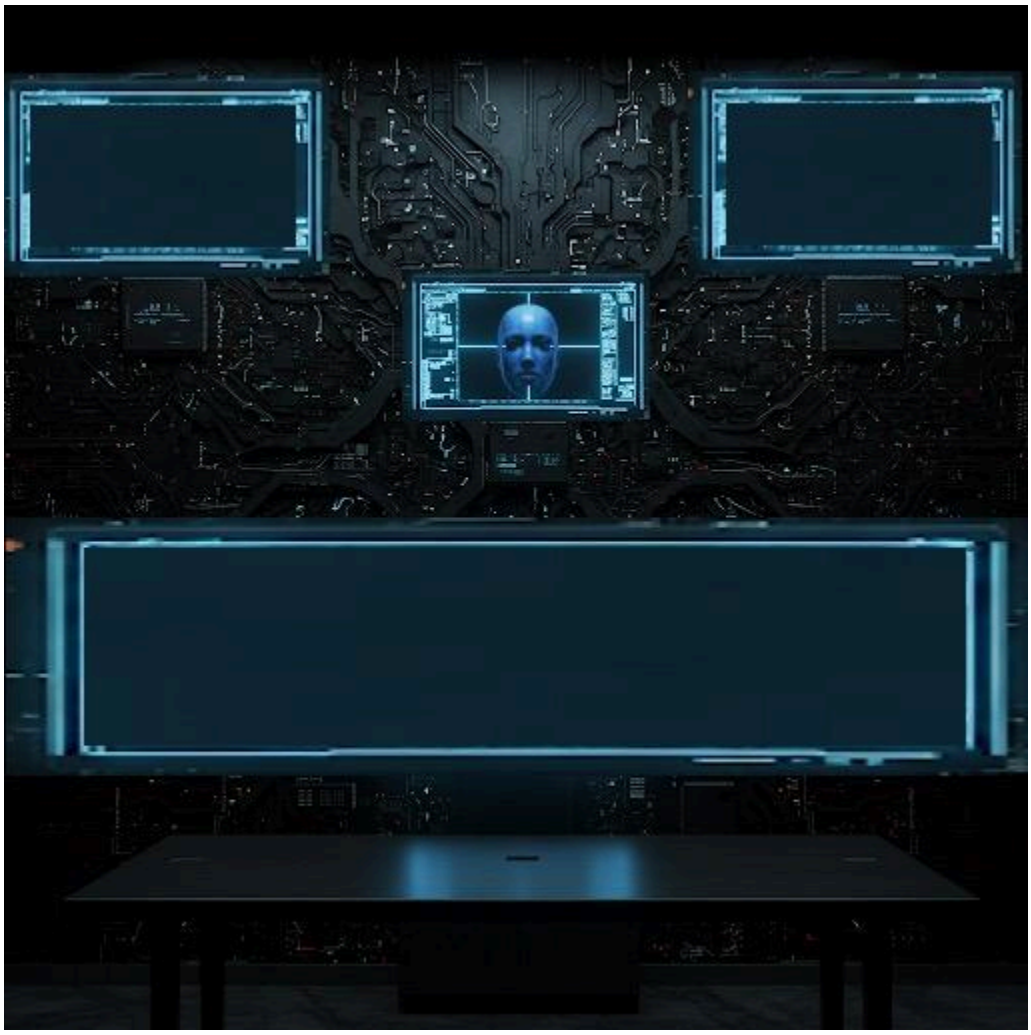
Week 6	● Add event handling to make the GUI functional (the "controller" part of MVC). ● Update project page with progress details. ● Submit code written so far.
Week 7	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 8	● Debug any remaining problems. ● Create project demonstration video, including information about how each LO is used as part of the project. ● Submit final code on Canvas, and add videos to the project page.

Github Repository

https://github.com/GitUser807/Lylecisc191project

Week 4: Updates

Gui Screenshot:

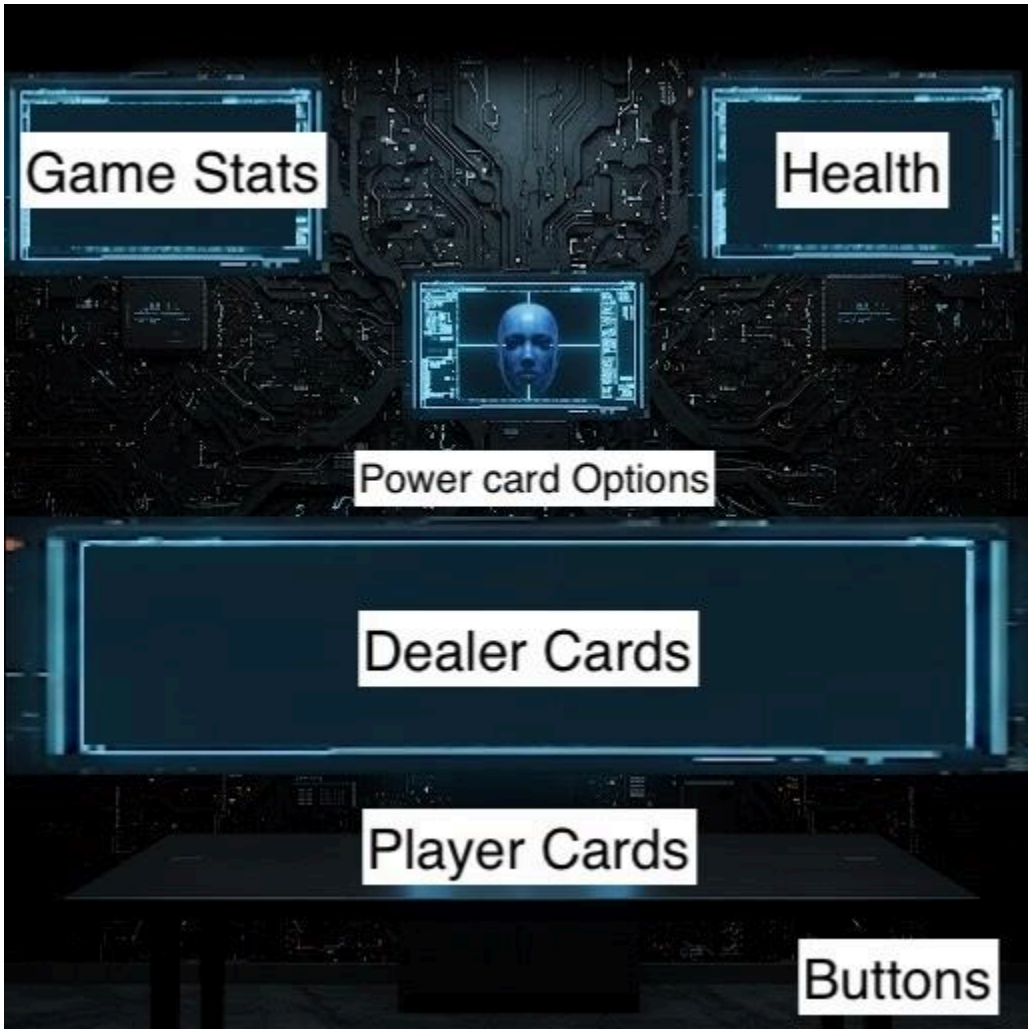


Week 4 Journal Entry:

Writing more code for my project has been enjoyable for me. I have been working on creating exception classes and handling it, as well as adding test classes. I have added 2 different runtime exceptions. One is the InvalidCardRuntimeException, which checks to see if the card added to one of the player's hands is recognizable or not. The other one is the InvalidAmountOfCardsRuntimeException which will be thrown when the user has more cards than the maximum set amount of cards(12). I have also added some test classes to make sure my code is working as intended. As of now, I have not made any changes to my project and I look forward to implementing my GUI soon.

Changes

- None



Week 4 Update Project Presentation: Blackjack With a Twist Game

Week 2: Updated Project Proposal

---

1. Planned Working Time

- **Working Days:** Friday - Sunday
- **Working Hours:** 10 AM - 2pm

---

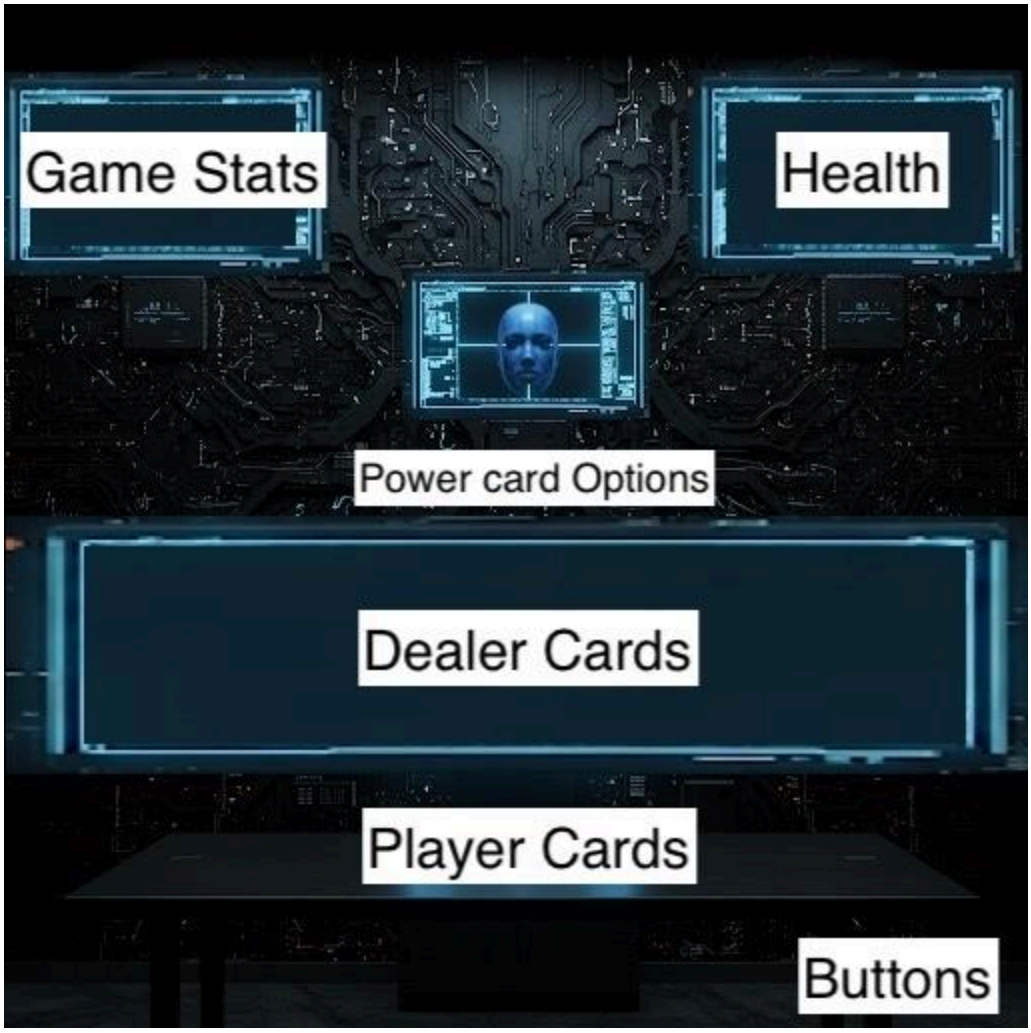
2. Project Pitch

Overview

Blackjack is one of the most popular card games, and the goal is to have a hand value closer to 21 than the dealer without exceeding it. In this project, I will create a playable Blackjack game with a graphical user interface (GUI), where a user can play against the dealer. I decided to base my game off of it but it has a twist. Both the player and the dealer have a health bar and whoever reaches 0 health first loses the game. The dealer will alternate turns as though you are against a non-dealer player. Both the dealer and the player will have multiple turns with each of their turns ending when either choose to hit or stay respectively. The round will end only when both the player and the dealer consecutively choose to stay on back-to-back turns. Additionally, if the player wins the round, they get a power card. This power card can have different effects based on the type you want.

The game will include the following main features:

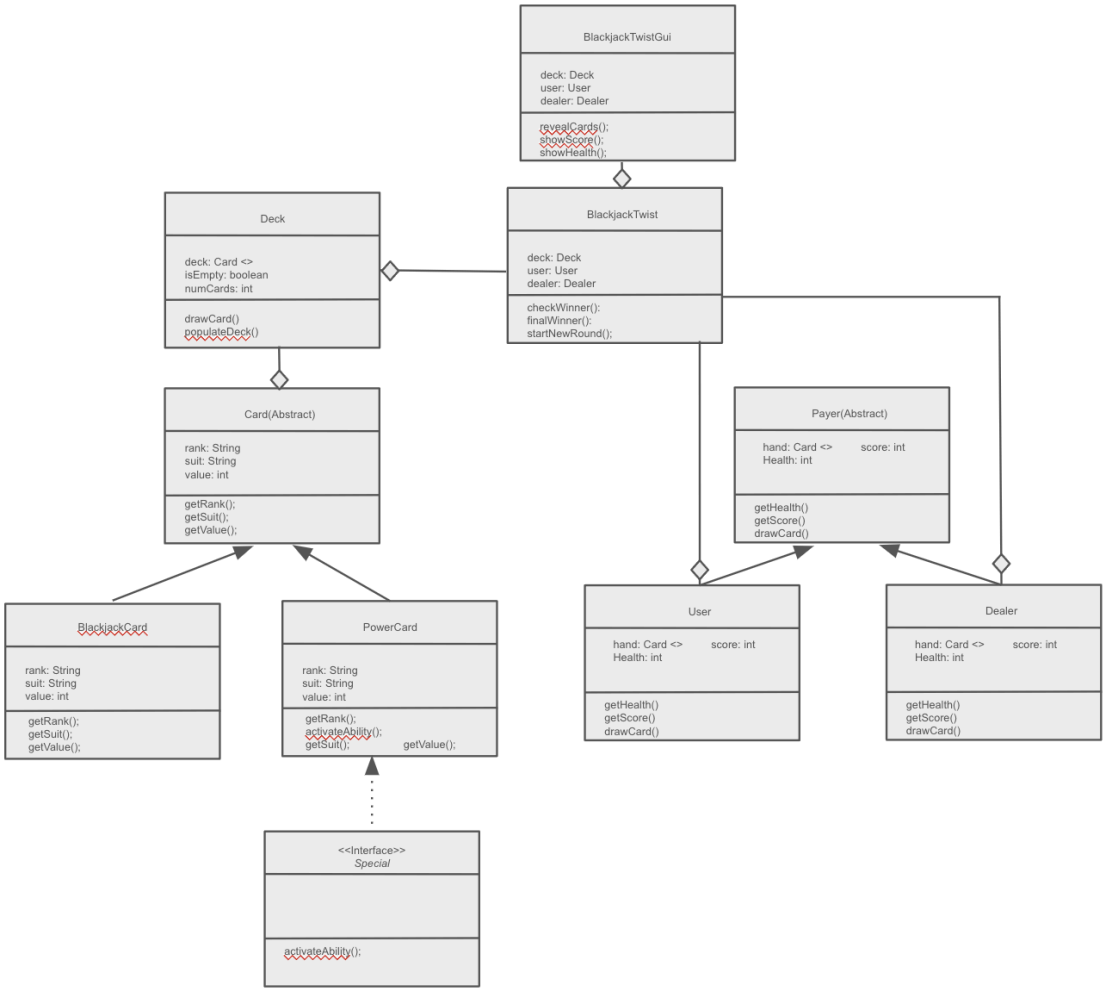
- **Card Dealing:** The player and dealer will each receive two cards at the beginning with the first card in each hand being visible.
- **Hit or Stand:** The player will be given the option to hit (draw a card) or stay (keep their hand). The player can draw a maximum of 12 cards. This prevents the user from drawing an absurd amount of cards because by these rules they can theoretically have an infinite amount of cards in their hand. This will also be implemented to discourage the user from holding on to a lot of power cards since they are carried over from round to round.
- **Dealer's Rules:** The dealer must keep choosing hit until they have 17 or more points. The dealer also wins ties to compensate for the fact that they don't get power cards.
- **Blackjack or Bust:** 21 is the best score you can get and you can keep drawing cards even if you bust(not recommended).
- **Power Card:** A power card of the winner choice will permanently stay in the winner hand until they use it.



The GUI will have two health bars representing the respective health level of the player and dealer. The Player uses two buttons to indicate whether they want to hit or stay during the game. There are also 3 buttons that pop up when the player wins a round where the player can choose to activate an ability for future rounds.

---

3. UML Diagram Overview



UML Breakdown:

3. **Core Game Classes:**
    - **Card** class to represent a card in the deck.
    - **Deck** class to represent a deck of cards.
    - **Player** class (abstract) to represent both the user and the dealer.
    - **User** class (inherits Player) to represent the player.
    - **Dealer** class (inherits Player) to represent the dealer.
    - **BlackjackGame** class to manage the game state (dealing cards, determining winners).
    - **BlackjackGUI** class to handle user interaction via the graphical interface.

CRC Cards:

Applying CRC Cards

CRC Card for Card:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Deck • Player • BlackjackCard • PowerCard

Applying CRC Cards

CRC Card for BlackjackCard:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Card

Applying CRC Cards

CRC Card for PowerCard:

Responsibilities	Collaborators
• Know the suit(category/vague description) • Know the rank(ability) • Know the value (0)	• Card

Applying CRC Cards

CRC Card for Special(Interface):

Responsibilities	Collaborators
• Generate ability for power cards	• PowerCard

Applying CRC Cards

CRC Card for Deck:

Responsibilities	Collaborators
• Populate the deck of cards • Deal a card	• Card • BlackjackTwist

Applying CRC Cards

CRC Card for Player(abstract):

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• User • Dealer • BlackjackTwist

Applying CRC Cards

CRC Card for User:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for Dealer:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the Dealer • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for BlackjackTwist:

Responsibilities	Collaborators
• Know the rules of the game • Deal cards to players • Determine winner • Be able to start new game	• BlackjackTwistGui • Deck • User • Dealer

Applying CRC Cards

CRC Card for BlackJackTwistGui:

Responsibilities	Collaborators
• Show the interface of the game • Have buttons that do certain things • Display health	• BlackJackTwist

1. Design Principles of Object-Oriented Programming (LO1)

I will use object-oriented programming principles like, abstraction, and inheritance in the Blackjack game. Additionally, I will create classes for the Card, Deck, Player, and Black Jack game.

2. Use Single and Multidimensional Arrays (LO2)

I'll be using array lists for holding the cards in the deck and players' hands. A multidimensional array could be used to hold the game history, such as the player 's previous actions.

3. Object and Classes, Including Aggregation (LO3)

There will be multiple classes:

- Card (to represent a single playing card)
- Deck (to represent the deck of cards)
- Player (to represent the user and dealer, holding hands and score)
- BlackJackTwist this will contain the rules of the game and make sure it is that each round will follow the specified rules

I will use aggregation where the `Deck` class contains a collection of `Card` objects, and a `Player` object has a collection of `Card` objects.

4. Inheritance and Polymorphism (LO4)

- Player and Card will both be abstract and super class of other classes
- EX: User and Dealer will be subclasses of Player
- EX: BlackjackCard and PowerCard will be a subclass of Card

5. Generic Collections and Data Structures (LO5)

I will be using generic collections, such as array lists and stacks, to store cards in the deck and the players' hand.

6. Graphical User Interface (LO6)

I will use GUI for buttons, the background, the health bar, and cards in the users' hand

7. Exception Handling (LO7)

I'll use exception handling to ensure that the game handles invalid inputs. For example the user enters something other than "hit" or "stand" or somehow gets a blank power card and to manage situations like an empty deck or out-of-bounds issues.

8. Text File I/O (LO8)

We'll store game results like player's scores in a text file and read from it to load previous scores when starting the game.

Week 1	● Write the project proposal. ● Plan the object-oriented design, starting with CRC cards. Determine classes (with fields and methods) and interfaces and their responsibilities. (The "model" part of MVC.) ● Create the UML. ● Begin writing project page.
Week 2	● Write code for The Card, BlackjackCard, PowerCard, and Deck Class. ● Develop test cases and test code as it is written and comment on my current code. ● Update project page with progress details. ● Submit code written so far.
Week 3	● Finish writing all my classes except for the Gui. ● Design the GUI (sketch it out on paper) - include the design in the Weekly update! ● Update project page with progress details. ● Submit code written so far.
Week 4	● Test, test, test, debug, and test some more. ● Plan where exception handling is needed to ensure the program fails gracefully. ● Update project page with progress details. ● Submit code written so far.
Week 5	● Write code to create a non-functional GUI (the "view" part of MVC). ● Update project page with progress details. ● Submit code written so far.

Week 6	● Add event handling to make the GUI functional (the "controller" part of MVC). ● Update project page with progress details. ● Submit code written so far.
Week 7	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 8	● Debug any remaining problems. ● Create project demonstration video, including information about how each LO is used as part of the project. ● Submit final code on Canvas, and add videos to the project page.

Github Repository

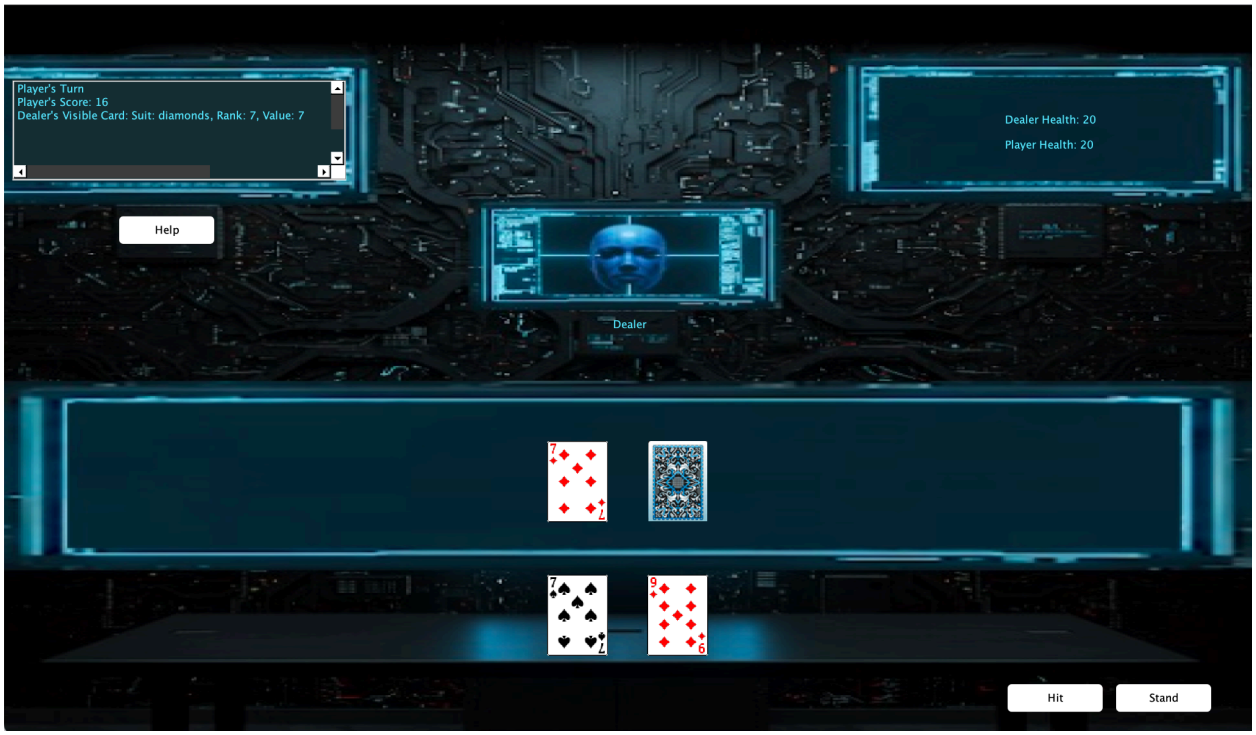
https://github.com/GitUser807/Lylecisc191project

Week 5: Updates

Github Repository

https://github.com/GitUser807/Lylecisc191project

Gui Screenshot:



Week 5 Journal Entry:

This week, I was able to dedicate significantly more time to writing code for my project. Everyday I worked on my project which allowed me to complete the first draft of my game. I have now successfully incorporated a GUI, exception handling, and file reading and writing into the program. Although spending so much time on this project has been tiring, it has been very rewarding to finally see my code come to life.

For my project, I feel the most time consuming part was incorporating the GUI aspect of the game. First, I gathered all the card images and created a background design. Then, I had to carefully choose the right data structure to manage the power cards(special ability) and their corresponding buttons. Since activating a power card removes it from the player's hand, I also needed to delete both the displayed card image and its associated JButton. To solve this, I used

a hash table to link each card to its button, allowing me to efficiently remove both elements when needed.

In addition, I refactored my exception handling to use checked exceptions rather than runtime exceptions to better demonstrate my understanding of proper exception management. I also implemented file reading and writing to record the moves made by both the dealer and the user in every round of the game. I added the "Help" button. So during gameplay when pressed, the users can view this information displayed in the text area on the screen.

Finally, I updated my deck implementation to use a stack instead of an array, making it behave more naturally for drawing cards. Moving forward, I plan to spend the upcoming weeks refining and polishing my code.

Changes

- The rest of the weeks will dedicated to testing and debugging my project

Week 5 Update Project Presentation: Blackjack With a Twist Game

Week 2: Updated Project Proposal

---

1. Planned Working Time

- **Working Days:** Friday - Sunday
- **Working Hours:** 10 AM - 2pm

---

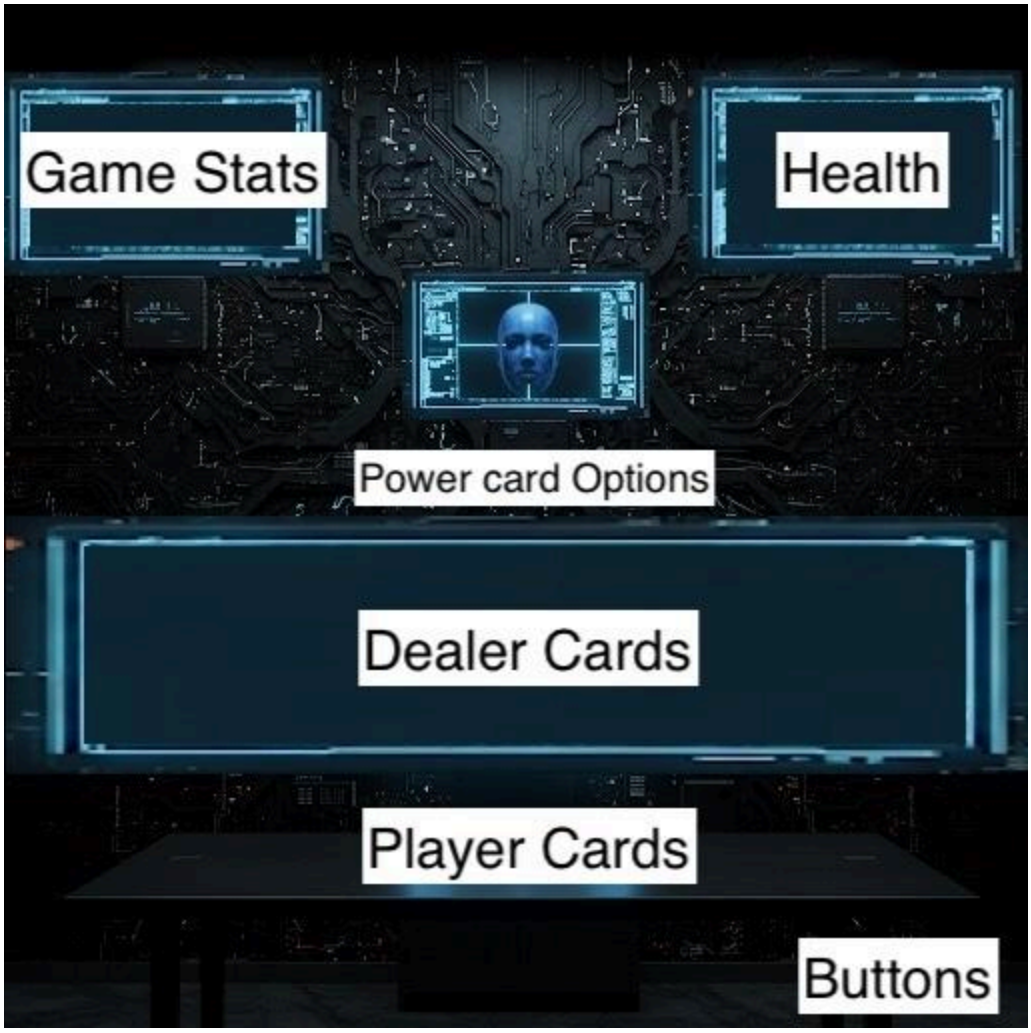
2. Project Pitch

Overview

Blackjack is one of the most popular card games, and the goal is to have a hand value closer to 21 than the dealer without exceeding it. In this project, I will create a playable Blackjack game with a graphical user interface (GUI), where a user can play against the dealer. I decided to base my game off of it but it has a twist. Both the player and the dealer have a health bar and whoever reaches 0 health first loses the game. The dealer will alternate turns as though you are against a non-dealer player. Both the dealer and the player will have multiple turns with each of their turns ending when either choose to hit or stay respectively. The round will end only when both the player and the dealer consecutively choose to stay on back-to-back turns. Additionally, if the player wins the round, they get a power card. This power card can have different effects based on the type you want.

The game will include the following main features:

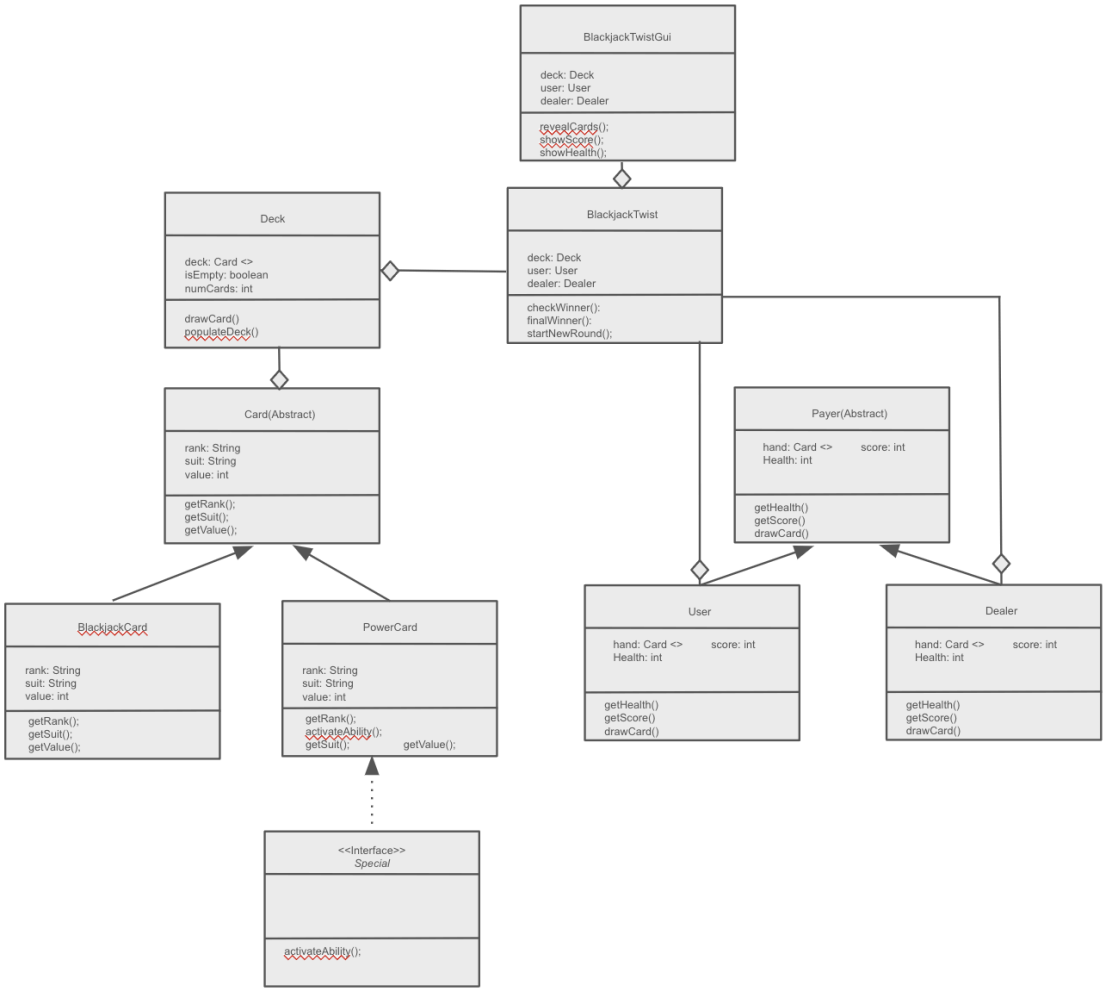
- **Card Dealing:** The player and dealer will each receive two cards at the beginning with the first card in each hand being visible.
- **Hit or Stand:** The player will be given the option to hit (draw a card) or stay (keep their hand). The player can draw a maximum of 12 cards. This prevents the user from drawing an absurd amount of cards because by these rules they can theoretically have an infinite amount of cards in their hand. This will also be implemented to discourage the user from holding on to a lot of power cards since they are carried over from round to round.
- **Dealer's Rules:** The dealer must keep choosing hit until they have 17 or more points. The dealer also wins ties to compensate for the fact that they don't get power cards.
- **Blackjack or Bust:** 21 is the best score you can get and you can keep drawing cards even if you bust(not recommended).
- **Power Card:** A power card of the winner choice will permanently stay in the winner hand until they use it.



The GUI will have two health bars representing the respective health level of the player and dealer. The Player uses two buttons to indicate whether they want to hit or stay during the game. There are also 3 buttons that pop up when the player wins a round where the player can choose to activate an ability for future rounds.

---

3. UML Diagram Overview



UML Breakdown:

1. **Core Game Classes:**
    - **Card** class to represent a card in the deck.
    - **Deck** class to represent a deck of cards.
    - **Player** class (abstract) to represent both the user and the dealer.
    - **User** class (inherits Player) to represent the player.
    - **Dealer** class (inherits Player) to represent the dealer.
    - **BlackjackGame** class to manage the game state (dealing cards, determining winners).
    - **BlackjackGUI** class to handle user interaction via the graphical interface.

CRC Cards:

Applying CRC Cards

CRC Card for Card:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Deck • Player • BlackjackCard • PowerCard

Applying CRC Cards

CRC Card for BlackjackCard:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Card

Applying CRC Cards

CRC Card for PowerCard:

Responsibilities	Collaborators
• Know the suit(category/vague description) • Know the rank(ability) • Know the value (0)	• Card

Applying CRC Cards

CRC Card for Special(Interface):

Responsibilities	Collaborators
• Generate ability for power cards	• PowerCard

Applying CRC Cards

CRC Card for Deck:

Responsibilities	Collaborators
• Populate the deck of cards • Deal a card	• Card • BlackjackTwist

Applying CRC Cards

CRC Card for Player(abstract):

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• User • Dealer • BlackjackTwist

Applying CRC Cards

CRC Card for User:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for Dealer:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the Dealer • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for BlackjackTwist:

Responsibilities	Collaborators
• Know the rules of the game • Deal cards to players • Determine winner • Be able to start new game	• BlackjackTwistGui • Deck • User • Dealer

Applying CRC Cards

CRC Card for BlackJackTwistGui:

Responsibilities	Collaborators
• Show the interface of the game • Have buttons that do certain things • Display health	• BlackJackTwist

1. Design Principles of Object-Oriented Programming (LO1)

I will use object-oriented programming principles like, abstraction, and inheritance in the Blackjack game. Additionally, I will create classes for the Card, Deck, Player, and Black Jack game.

2. Use Single and Multidimensional Arrays (LO2)

I'll be using array lists for holding the cards in the deck and players' hands. A multidimensional array could be used to hold the game history, such as the player 's previous actions.

3. Object and Classes, Including Aggregation (LO3)

There will be multiple classes:

- Card (to represent a single playing card)
- Deck (to represent the deck of cards)
- Player (to represent the user and dealer, holding hands and score)
- BlackJackTwist this will contain the rules of the game and make sure it is that each round will follow the specified rules

I will use aggregation where the `Deck` class contains a collection of `Card` objects, and a `Player` object has a collection of `Card` objects.

4. Inheritance and Polymorphism (LO4)

- Player and Card will both be abstract and super class of other classes
- EX: User and Dealer will be subclasses of Player
- EX: BlackjackCard and PowerCard will be a subclass of Card

5. Generic Collections and Data Structures (LO5)

I will be using generic collections, such as array lists and stacks, to store cards in the deck and the players' hand.

6. Graphical User Interface (LO6)

I will use GUI for buttons, the background, the health bar, and cards in the users' hand

7. Exception Handling (LO7)

I'll use exception handling to ensure that the game handles invalid inputs. For example the user enters something other than "hit" or "stand" or somehow gets a blank power card and to manage situations like an empty deck or out-of-bounds issues.

8. Text File I/O (LO8)

We'll store game results like player's scores in a text file and read from it to load previous scores when starting the game.

Week 1	● Write the project proposal. ● Plan the object-oriented design, starting with CRC cards. Determine classes (with fields and methods) and interfaces and their responsibilities. (The "model" part of MVC.) ● Create the UML. ● Begin writing project page.
Week 2	● Write code for The Card, BlackjackCard, PowerCard, and Deck Class. ● Develop test cases and test code as it is written and comment on my current code. ● Update project page with progress details. ● Submit code written so far.
Week 3	● Finish writing all my classes except for the Gui. ● Design the GUI (sketch it out on paper) - include the design in the Weekly update! ● Update project page with progress details. ● Submit code written so far.
Week 4	● Test, test, test, debug, and test some more. ● Plan where exception handling is needed to ensure the program fails gracefully. ● Update project page with progress details. ● Submit code written so far.
Week 5	● Finish First Draft of Game ○ Implement Exception Handling ○ Write the GUI for the game ○ Incorporate File reading and writing to the game

Week 6	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 7	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 8	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.

Github Repository

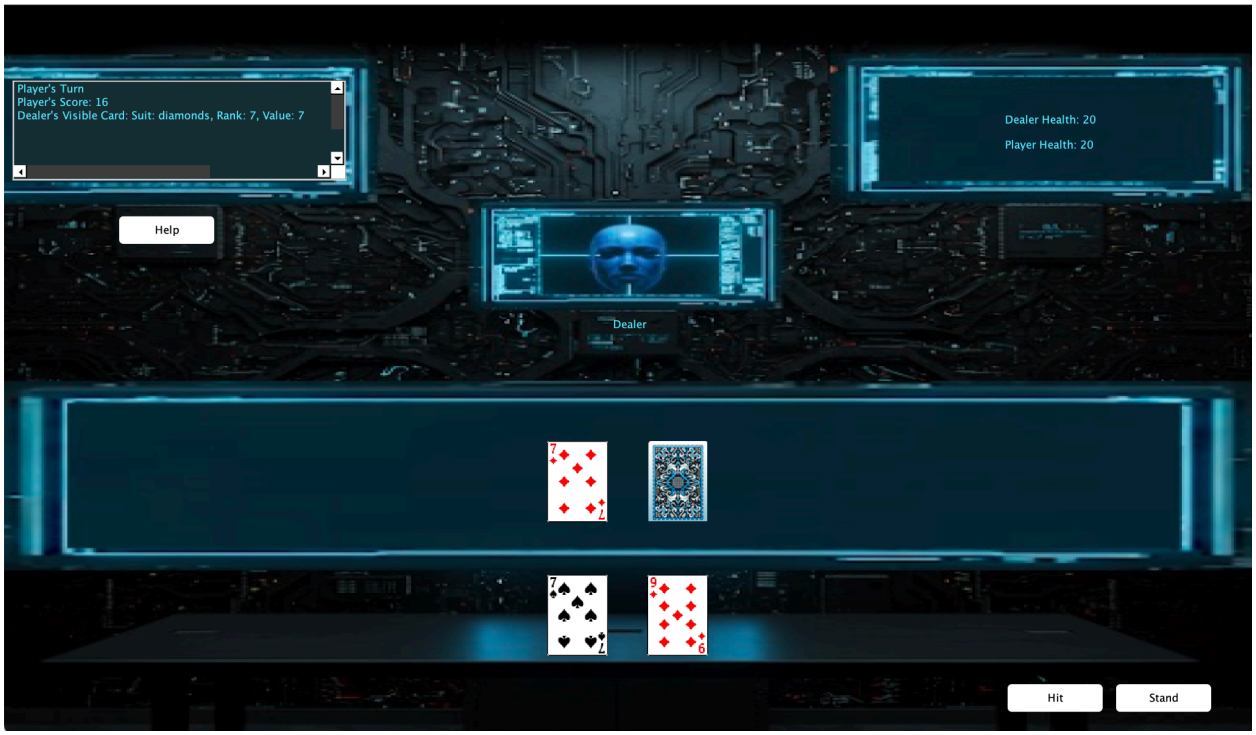
https://github.com/GitUser807/Lylecisc191project

Week 6: Updates

Github Repository

https://github.com/GitUser807/Lylecisc191project

Gui Screenshot:



Week 6 Journal Entry:

This week, I primarily focused on debugging and improving code clarity through method and variable renaming. One significant issue I encountered involved the Dealer's health not updating in the GUI during gameplay. Initially, I considered several potential causes, so I began isolating what was working to help narrow down the source of the bug.

First, I checked the game stats file displayed via the help button. The dealer's health was updating correctly there, which indicated that the game logic was functioning as expected — the issue likely stemmed from the GUI layer. I then observed that the user's health was updating correctly in the GUI throughout each round, suggesting that I had correctly placed the update call for the user's health.

This led me to suspect that the dealer's health JLabel simply wasn't being updated in the relevant method. Using search (Command + F), I scanned for where the dealer's health label was manipulated and discovered that I had forgotten to include the update in one key method. Once I added the necessary line to update the dealer's health JLabel, the issue was resolved.

In addition to debugging, I improved code clarity by renaming variables to be more descriptive. I also standardized terminology across the codebase — replacing all uses of "player" with "user" wherever the user-specific data was involved. Since both User and Dealer extend the Player class, this distinction makes the code easier to read and maintain.

Changes

- None

Week 6 Update Project Presentation: Blackjack With a Twist Game

Week 6: Updated Project Proposal

---

1. Planned Working Time

- **Working Days:** Friday - Sunday
- **Working Hours:** 10 AM - 2pm

---

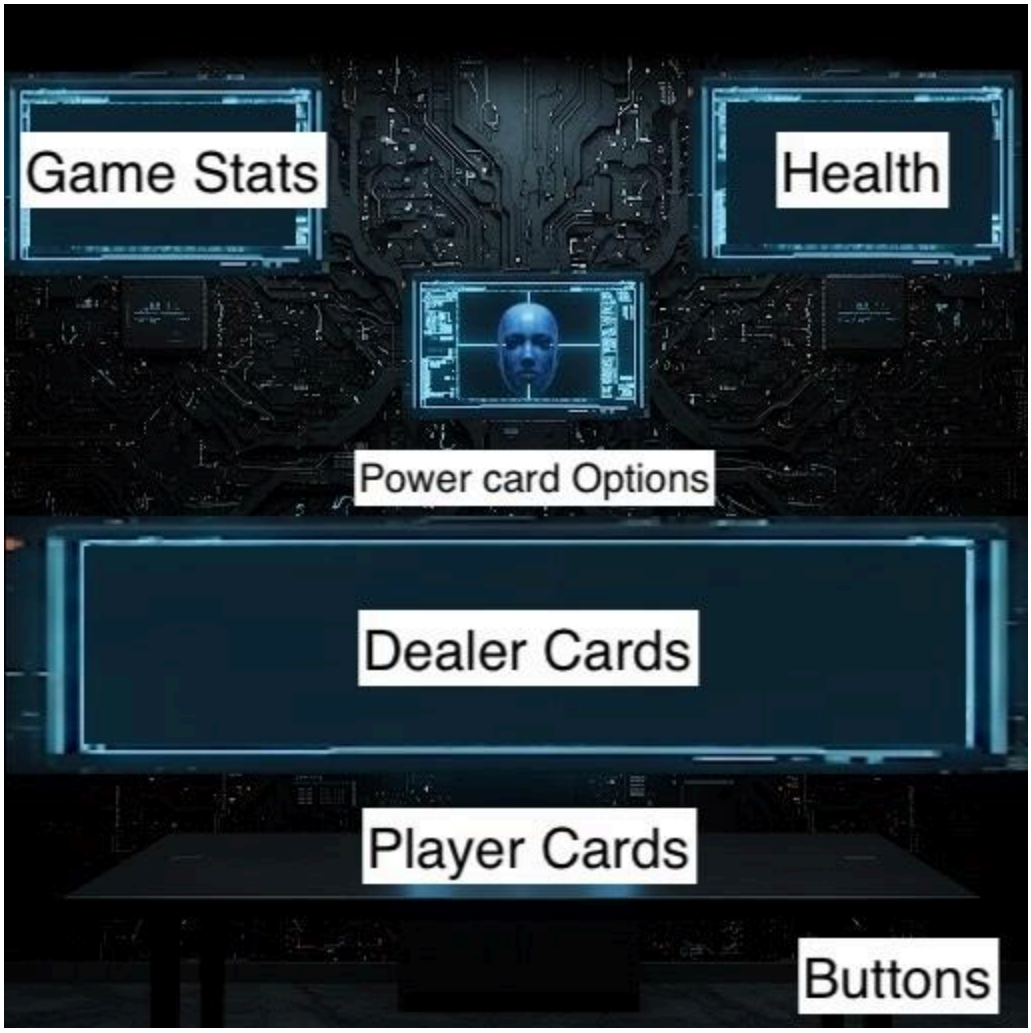
2. Project Pitch

Overview

Blackjack is one of the most popular card games, and the goal is to have a hand value closer to 21 than the dealer without exceeding it. In this project, I will create a playable Blackjack game with a graphical user interface (GUI), where a user can play against the dealer. I decided to base my game off of it but it has a twist. Both the player and the dealer have a health bar and whoever reaches 0 health first loses the game. The dealer will alternate turns as though you are against a non-dealer player. Both the dealer and the player will have multiple turns with each of their turns ending when either choose to hit or stay respectively. The round will end only when both the player and the dealer consecutively choose to stay on back-to-back turns. Additionally, if the player wins the round, they get a power card. This power card can have different effects based on the type you want.

The game will include the following main features:

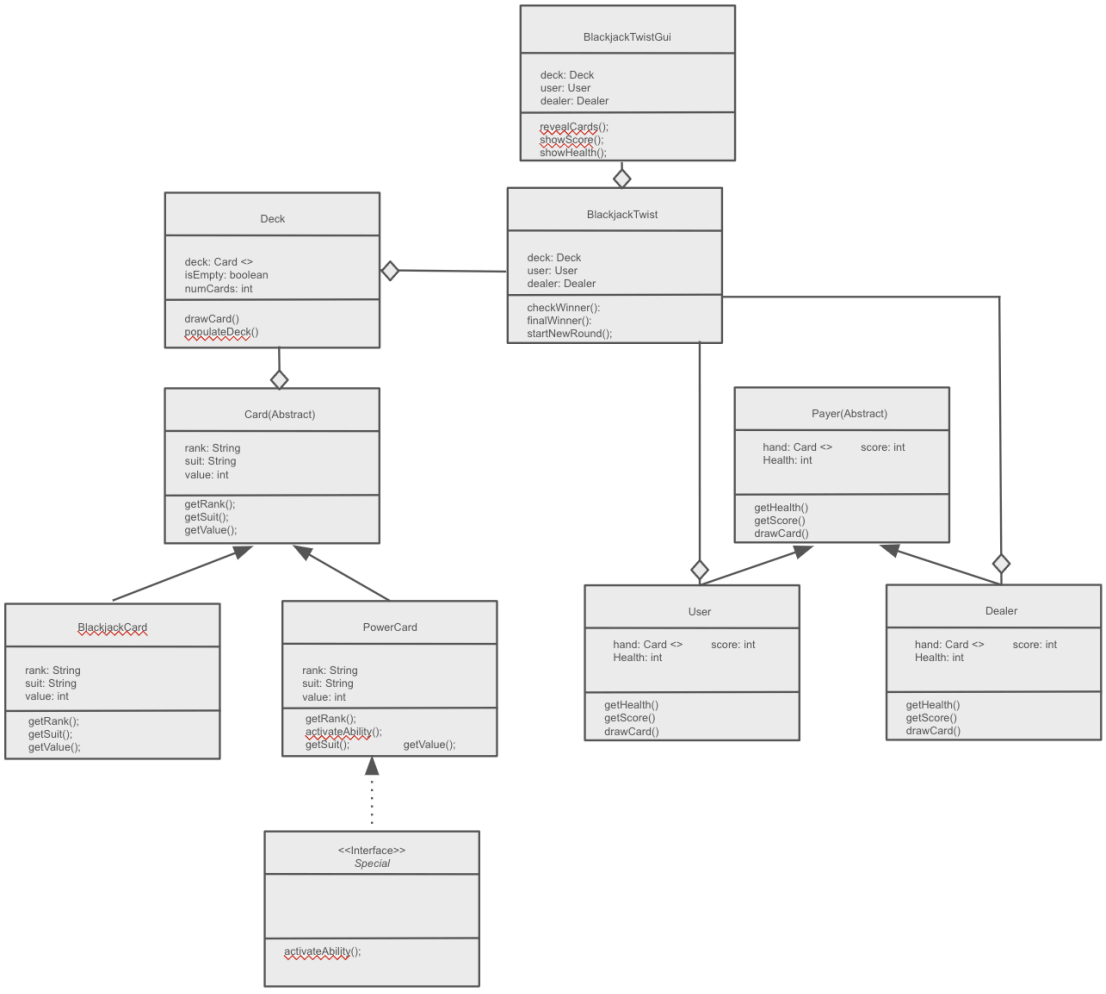
- **Card Dealing:** The player and dealer will each receive two cards at the beginning with the first card in each hand being visible.
- **Hit or Stand:** The player will be given the option to hit (draw a card) or stay (keep their hand). The player can draw a maximum of 12 cards. This prevents the user from drawing an absurd amount of cards because by these rules they can theoretically have an infinite amount of cards in their hand. This will also be implemented to discourage the user from holding on to a lot of power cards since they are carried over from round to round.
- **Dealer's Rules:** The dealer must keep choosing hit until they have 17 or more points. The dealer also wins ties to compensate for the fact that they don't get power cards.
- **Blackjack or Bust:** 21 is the best score you can get and you can keep drawing cards even if you bust(not recommended).
- **Power Card:** A power card of the winner choice will permanently stay in the winner hand until they use it.



The GUI will have two health bars representing the respective health level of the player and dealer. The Player uses two buttons to indicate whether they want to hit or stay during the game. There are also 3 buttons that pop up when the player wins a round where the player can choose to activate an ability for future rounds.

---

3. UML Diagram Overview



UML Breakdown:

2. **Core Game Classes:**
    - **Card** class to represent a card in the deck.
    - **Deck** class to represent a deck of cards.
    - **Player** class (abstract) to represent both the user and the dealer.
    - **User** class (inherits Player) to represent the player.
    - **Dealer** class (inherits Player) to represent the dealer.
    - **BlackjackGame** class to manage the game state (dealing cards, determining winners).
    - **BlackjackGUI** class to handle user interaction via the graphical interface.

CRC Cards:

Applying CRC Cards

CRC Card for Card:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Deck • Player • BlackjackCard • PowerCard

Applying CRC Cards

CRC Card for BlackjackCard:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Card

Applying CRC Cards

CRC Card for PowerCard:

Responsibilities	Collaborators
• Know the suit(category/vague description) • Know the rank(ability) • Know the value (0)	• Card

Applying CRC Cards

CRC Card for Special(Interface):

Responsibilities	Collaborators
• Generate ability for power cards	• PowerCard

Applying CRC Cards

CRC Card for Deck:

Responsibilities	Collaborators
• Populate the deck of cards • Deal a card	• Card • BlackjackTwist

Applying CRC Cards

CRC Card for Player(abstract):

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• User • Dealer • BlackjackTwist

Applying CRC Cards

CRC Card for User:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for Dealer:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the Dealer • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for BlackjackTwist:

Responsibilities	Collaborators
• Know the rules of the game • Deal cards to players • Determine winner • Be able to start new game	• BlackjackTwistGui • Deck • User • Dealer

Applying CRC Cards

CRC Card for BlackJackTwistGui:

Responsibilities	Collaborators
• Show the interface of the game • Have buttons that do certain things • Display health	• BlackJackTwist

1. Design Principles of Object-Oriented Programming (LO1)

I will use object-oriented programming principles like, abstraction, and inheritance in the Blackjack game. Additionally, I will create classes for the Card, Deck, Player, and Black Jack game.

2. Use Single and Multidimensional Arrays (LO2)

I'll be using array lists for holding the cards in the deck and players' hands. A multidimensional array could be used to hold the game history, such as the player 's previous actions.

3. Object and Classes, Including Aggregation (LO3)

There will be multiple classes:

- Card (to represent a single playing card)
- Deck (to represent the deck of cards)
- Player (to represent the user and dealer, holding hands and score)
- BlackJackTwist this will contain the rules of the game and make sure it is that each round will follow the specified rules

I will use aggregation where the `Deck` class contains a collection of `Card` objects, and a `Player` object has a collection of `Card` objects.

4. Inheritance and Polymorphism (LO4)

- Player and Card will both be abstract and super class of other classes
- EX: User and Dealer will be subclasses of Player
- EX: BlackjackCard and PowerCard will be a subclass of Card

5. Generic Collections and Data Structures (LO5)

I will be using generic collections, such as array lists and stacks, to store cards in the deck and the players' hand.

6. Graphical User Interface (LO6)

I will use GUI for buttons, the background, the health bar, and cards in the users' hand

7. Exception Handling (LO7)

I'll use exception handling to ensure that the game handles invalid inputs. For example the user enters something other than "hit" or "stand" or somehow gets a blank power card and to manage situations like an empty deck or out-of-bounds issues.

8. Text File I/O (LO8)

We'll store game results like player's scores in a text file and read from it to load previous scores when starting the game.

Week 1	● Write the project proposal. ● Plan the object-oriented design, starting with CRC cards. Determine classes (with fields and methods) and interfaces and their responsibilities. (The "model" part of MVC.) ● Create the UML. ● Begin writing project page.
Week 2	● Write code for The Card, BlackjackCard, PowerCard, and Deck Class. ● Develop test cases and test code as it is written and comment on my current code. ● Update project page with progress details. ● Submit code written so far.
Week 3	● Finish writing all my classes except for the Gui. ● Design the GUI (sketch it out on paper) - include the design in the Weekly update! ● Update project page with progress details. ● Submit code written so far.
Week 4	● Test, test, test, debug, and test some more. ● Plan where exception handling is needed to ensure the program fails gracefully. ● Update project page with progress details. ● Submit code written so far.
Week 5	● Finish First Draft of Game ○ Implement Exception Handling ○ Write the GUI for the game ○ Incorporate File reading and writing to the game

Week 6	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 7	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 8	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.

Github Repository

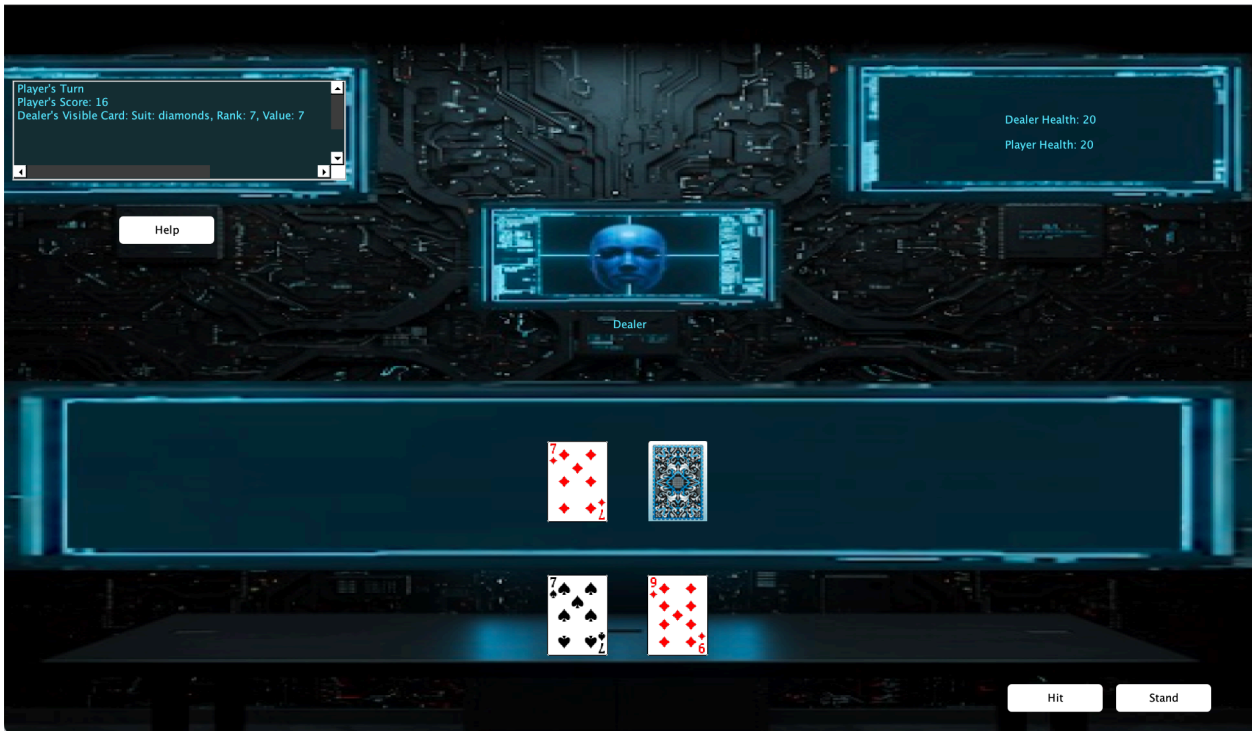
https://github.com/GitUser807/Lylecisc191project

Week 7: Updates

Github Repository

https://github.com/GitUser807/Lylecisc191project

Gui Screenshot:



Week 7 Journal Entry:

This week, I mainly focused on debugging my code and ensuring proper citation of images used in my project. One issue I encountered involved the .txt file where I wrote the history of the game. I noticed that the rounds were being recorded starting from round 0 instead of round 1. After reviewing the relevant portions of my code, I discovered that the round counter was being updated after the current round value was written to the file. This meant that every entry in the txt file was consistently one round behind the actual state of the game. To resolve this, I modified the code so that the round number is incremented before it is written to the .txt file. After implementing this fix, the round tracking in the history log was correctly synched with the gameplay, and the issue was successfully resolved. This was the only bug I found this week.

In addition to debugging, I also took time to properly cite the image of my cards that I used in the project. This involved documenting the source of the image and ensuring that it met the citation guidelines appropriate for my work.

Overall, this week involved a mix of troubleshooting a subtle but important bug and handling the administrative task of attribution, both of which are crucial aspects of maintaining a functional and responsible software project.

Changes

- None

Week 7 Update Project Presentation: Blackjack With a Twist Game

Week 7: Updated Project Proposal

---

1. Planned Working Time

- **Working Days:** Friday - Sunday
- **Working Hours:** 10 AM - 2pm

---

2. Project Pitch

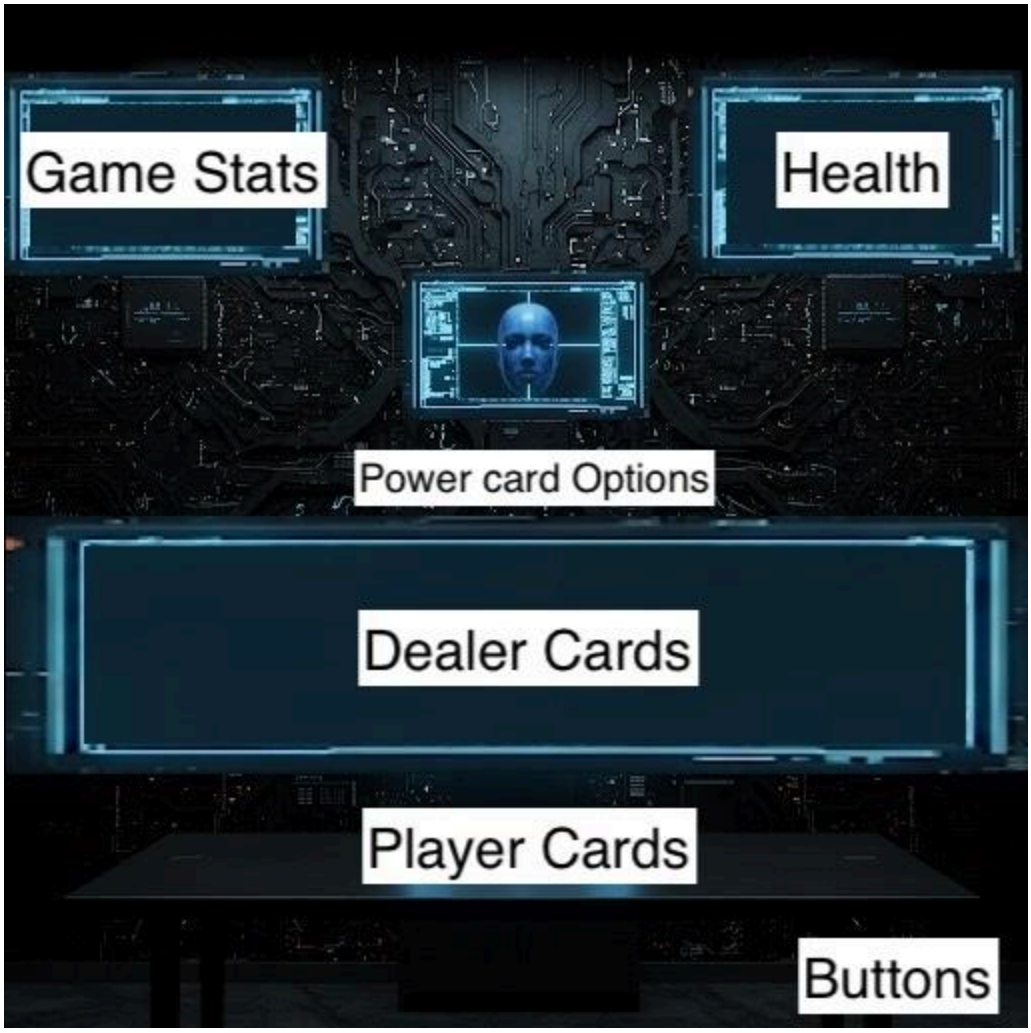
Overview

Blackjack is one of the most popular card games, and the goal is to have a hand value closer to 21 than the dealer without exceeding it. In this project, I will create a playable Blackjack game with a graphical user interface (GUI), where a user can play against the dealer. I decided to

base my game off of it but it has a twist. Both the player and the dealer have a health bar and whoever reaches 0 health first loses the game. The dealer will alternate turns as though you are against a non-dealer player. Both the dealer and the player will have multiple turns with each of their turns ending when either choose to hit or stay respectively. The round will end only when both the player and the dealer consecutively choose to stay on back-to-back turns. Additionally, if the player wins the round, they get a power card. This power card can have different effects based on the type you want.

The game will include the following main features:

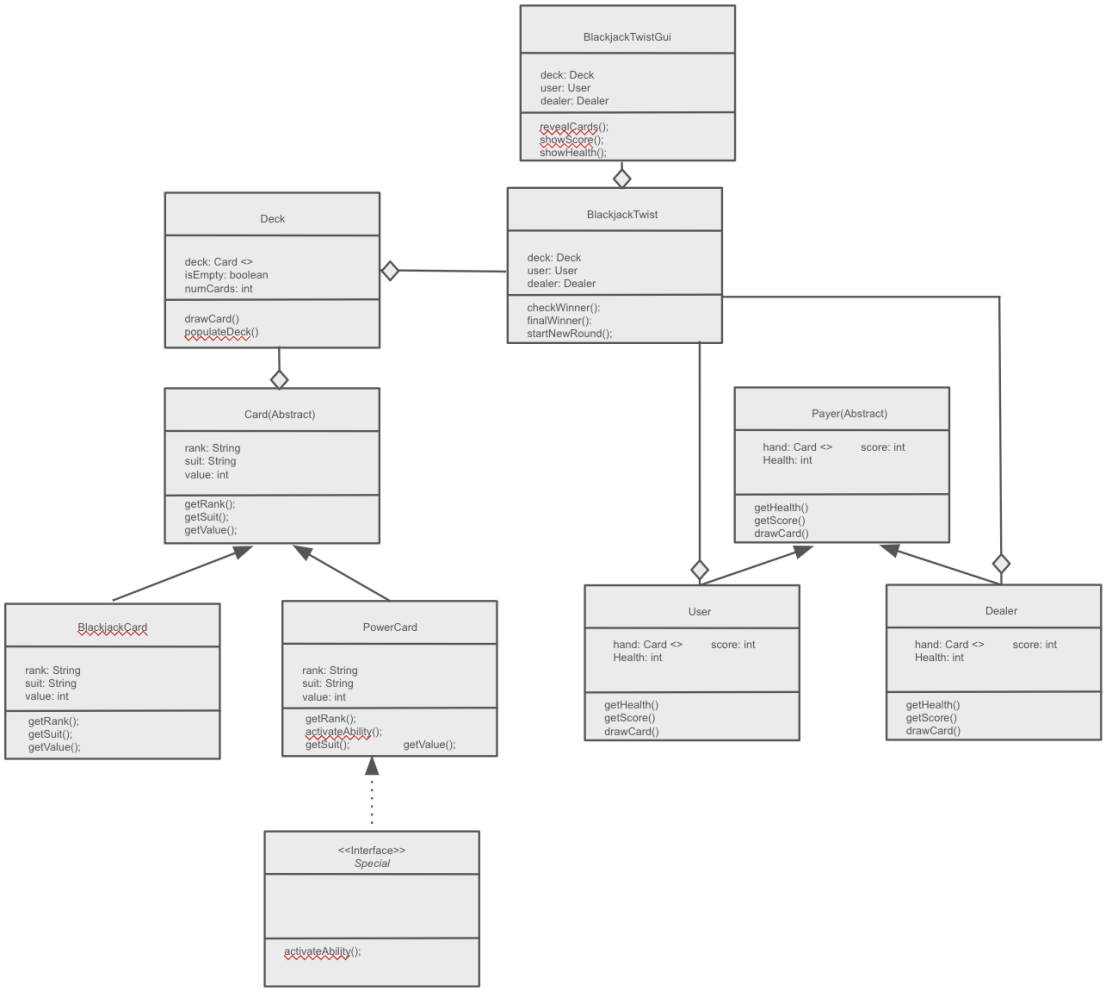
- **Card Dealing:** The player and dealer will each receive two cards at the beginning with the first card in each hand being visible.
- **Hit or Stand:** The player will be given the option to hit (draw a card) or stay (keep their hand). The player can draw a maximum of 12 cards. This prevents the user from drawing an absurd amount of cards because by these rules they can theoretically have an infinite amount of cards in their hand. This will also be implemented to discourage the user from holding on to a lot of power cards since they are carried over from round to round.
- **Dealer's Rules:** The dealer must keep choosing hit until they have 17 or more points. The dealer also wins ties to compensate for the fact that they don't get power cards.
- **Blackjack or Bust:** 21 is the best score you can get and you can keep drawing cards even if you bust(not recommended).
- **Power Card:** A power card of the winner choice will permanently stay in the winner hand until they use it.



The GUI will have two health bars representing the respective health level of the player and dealer. The Player uses two buttons to indicate whether they want to hit or stay during the game. There are also 3 buttons that pop up when the player wins a round where the player can choose to activate an ability for future rounds.

---

3. UML Diagram Overview



UML Breakdown:

3. **Core Game Classes:**
    - **Card** class to represent a card in the deck.
    - **Deck** class to represent a deck of cards.
    - **Player** class (abstract) to represent both the user and the dealer.
    - **User** class (inherits Player) to represent the player.
    - **Dealer** class (inherits Player) to represent the dealer.
    - **BlackjackGame** class to manage the game state (dealing cards, determining winners).
    - **BlackjackGUI** class to handle user interaction via the graphical interface.

CRC Cards:

Applying CRC Cards

CRC Card for Card:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Deck • Player • BlackjackCard • PowerCard

Applying CRC Cards

CRC Card for BlackjackCard:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Card

Applying CRC Cards

CRC Card for PowerCard:

Responsibilities	Collaborators
• Know the suit(category/vague description) • Know the rank(ability) • Know the value (0)	• Card

Applying CRC Cards

CRC Card for Special(Interface):

Responsibilities	Collaborators
• Generate ability for power cards	• PowerCard

Applying CRC Cards

CRC Card for Deck:

Responsibilities	Collaborators
• Populate the deck of cards • Deal a card	• Card • BlackjackTwist

Applying CRC Cards

CRC Card for Player(abstract):

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• User • Dealer • BlackjackTwist

Applying CRC Cards

CRC Card for User:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for Dealer:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the Dealer • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for BlackjackTwist:

Responsibilities	Collaborators
• Know the rules of the game • Deal cards to players • Determine winner • Be able to start new game	• BlackjackTwistGui • Deck • User • Dealer

Applying CRC Cards

CRC Card for BlackJackTwistGui:

Responsibilities	Collaborators
• Show the interface of the game • Have buttons that do certain things • Display health	• BlackJackTwist

1. Design Principles of Object-Oriented Programming (LO1)

I will use object-oriented programming principles like, abstraction, and inheritance in the Blackjack game. Additionally, I will create classes for the Card, Deck, Player, and Black Jack game.

2. Use Single and Multidimensional Arrays (LO2)

I'll be using array lists for holding the cards in the deck and players' hands. A multidimensional array could be used to hold the game history, such as the player 's previous actions.

3. Object and Classes, Including Aggregation (LO3)

There will be multiple classes:

- Card (to represent a single playing card)
- Deck (to represent the deck of cards)
- Player (to represent the user and dealer, holding hands and score)
- BlackJackTwist this will contain the rules of the game and make sure it is that each round will follow the specified rules

I will use aggregation where the `Deck` class contains a collection of `Card` objects, and a `Player` object has a collection of `Card` objects.

4. Inheritance and Polymorphism (LO4)

- Player and Card will both be abstract and super class of other classes
- EX: User and Dealer will be subclasses of Player
- EX: BlackjackCard and PowerCard will be a subclass of Card

5. Generic Collections and Data Structures (LO5)

I will be using generic collections, such as array lists and stacks, to store cards in the deck and the players' hand.

6. Graphical User Interface (LO6)

I will use GUI for buttons, the background, the health bar, and cards in the users' hand

7. Exception Handling (LO7)

I'll use exception handling to ensure that the game handles invalid inputs. For example the user enters something other than "hit" or "stand" or somehow gets a blank power card and to manage situations like an empty deck or out-of-bounds issues.

8. Text File I/O (LO8)

We'll store game results like player's scores in a text file and read from it to load previous scores when starting the game.

Week 1	● Write the project proposal. ● Plan the object-oriented design, starting with CRC cards. Determine classes (with fields and methods) and interfaces and their responsibilities. (The "model" part of MVC.) ● Create the UML. ● Begin writing project page.
Week 2	● Write code for The Card, BlackjackCard, PowerCard, and Deck Class. ● Develop test cases and test code as it is written and comment on my current code. ● Update project page with progress details. ● Submit code written so far.
Week 3	● Finish writing all my classes except for the Gui. ● Design the GUI (sketch it out on paper) - include the design in the Weekly update! ● Update project page with progress details. ● Submit code written so far.
Week 4	● Test, test, test, debug, and test some more. ● Plan where exception handling is needed to ensure the program fails gracefully. ● Update project page with progress details. ● Submit code written so far.
Week 5	● Finish First Draft of Game ○ Implement Exception Handling ○ Write the GUI for the game ○ Incorporate File reading and writing to the game

Week 6	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 7	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 8	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.

Github Repository

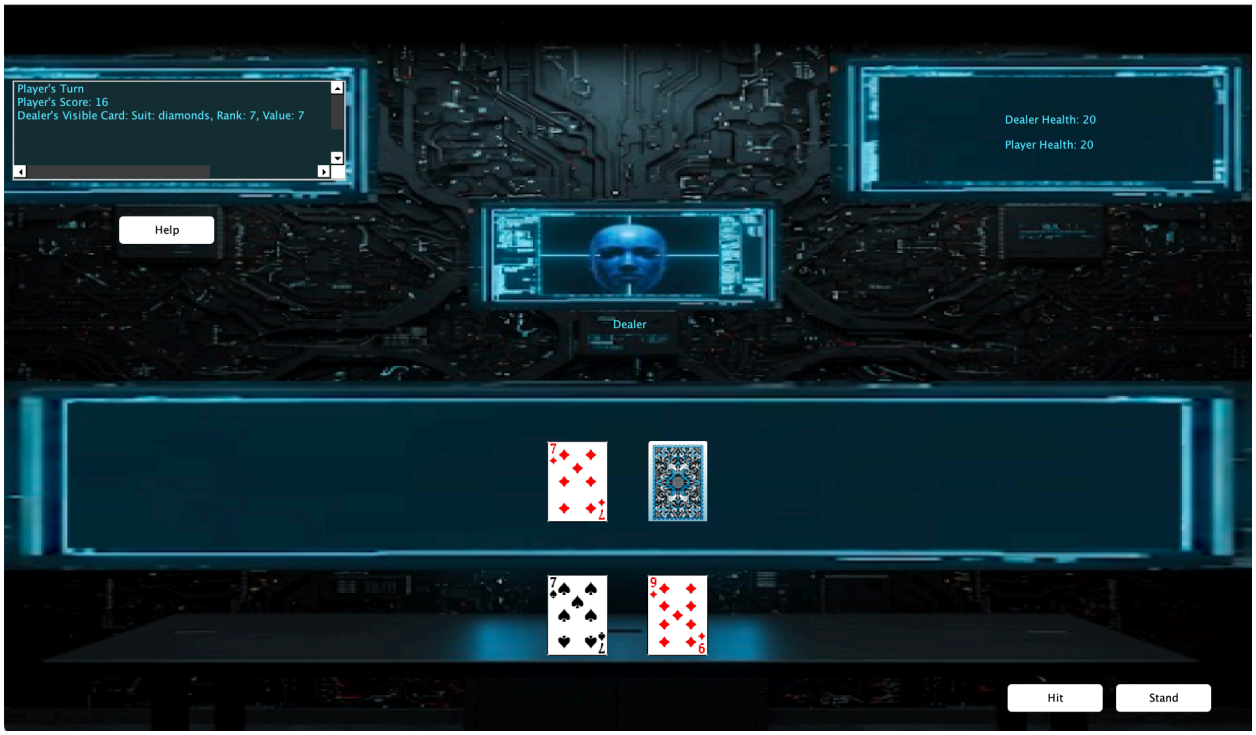
https://github.com/GitUser807/Lylecisc191project

Week 8: Updates

Github Repository

https://github.com/GitUser807/Lylecisc191project

Gui Screenshot:



Week 8 Journal Entry:

This week, I focused primarily on implementing the feedback provided by my instructor.

The first change I made was updating all my instance variables to be private. Initially, I had left some of them with default access while debugging and overlooked changing them afterward. Making them private improves encapsulation and demonstrates good practices in object-oriented programming.

Next, I added comments clarifying the IS-A and HAS-A relationships among my classes and instance variables. This was done to enhance readability and help anyone reviewing my code better understand the design structure.

I also added a brief explanation in my code comments about why I use getClass().getResource(picUrl) to load images. This method is ideal for GUI applications using Swing, as it loads resources from the classpath. I first learned about it during a hackathon, where a teammate explained that it's the preferred approach for loading bundled images—whether running the app inside an IDE or from a JAR file. It increases portability by avoiding hard coded file paths, which can break when the project is moved to another system.

Finally, I removed SwingUtilities.invokeLater from my code. After reviewing its purpose, I realized that since my application does not involve multithreaded UI updates, using it was unnecessary.

Changes

- None

Week 8 Update Project Presentation: Blackjack With a Twist Game

Week 8: Updated Project Proposal

---

1. Planned Working Time

- **Working Days:** Friday - Sunday
- **Working Hours:** 10 AM - 2pm

---

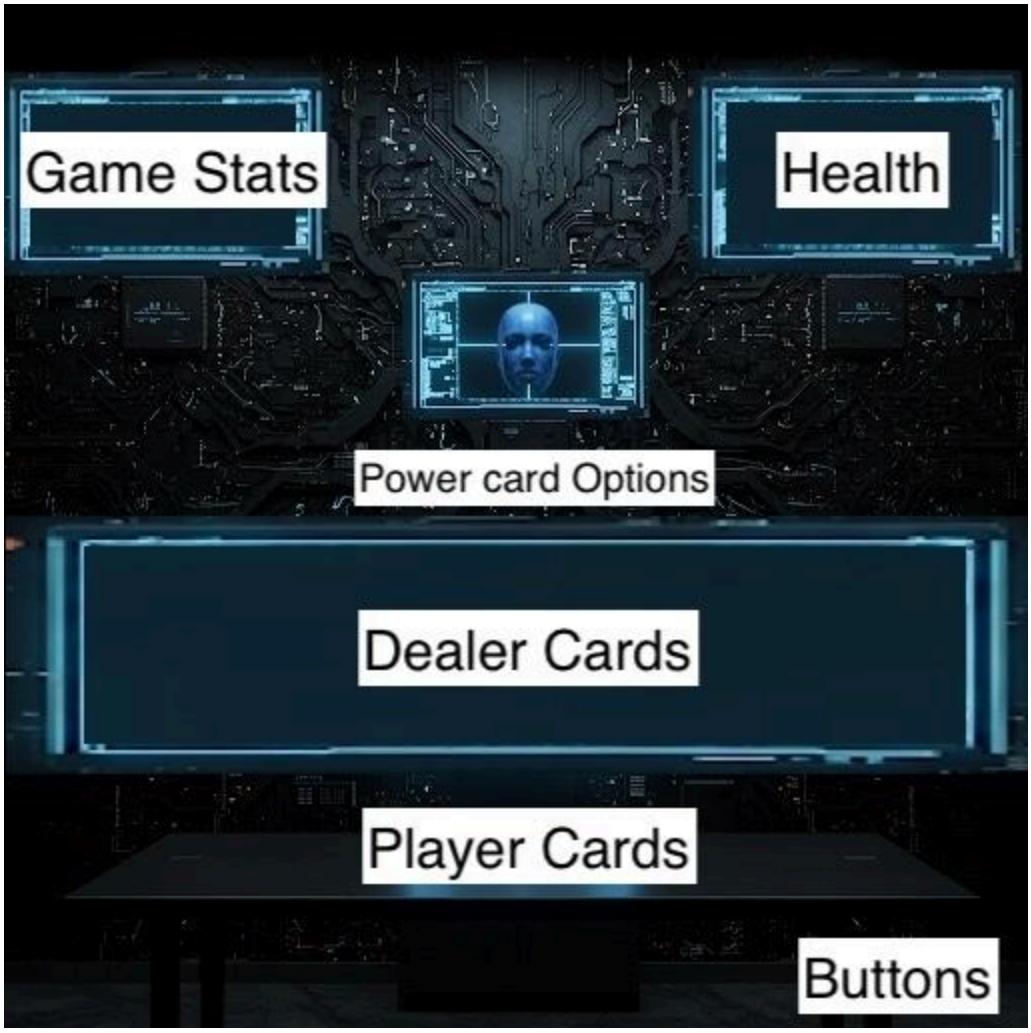
2. Project Pitch

Overview

Blackjack is one of the most popular card games, and the goal is to have a hand value closer to 21 than the dealer without exceeding it. In this project, I will create a playable Blackjack game with a graphical user interface (GUI), where a user can play against the dealer. I decided to base my game off of it but it has a twist. Both the player and the dealer have a health bar and whoever reaches 0 health first loses the game. The dealer will alternate turns as though you are against a non-dealer player. Both the dealer and the player will have multiple turns with each of their turns ending when either choose to hit or stay respectively. The round will end only when both the player and the dealer consecutively choose to stay on back-to-back turns. Additionally, if the player wins the round, they get a power card. This power card can have different effects based on the type you want.

The game will include the following main features:

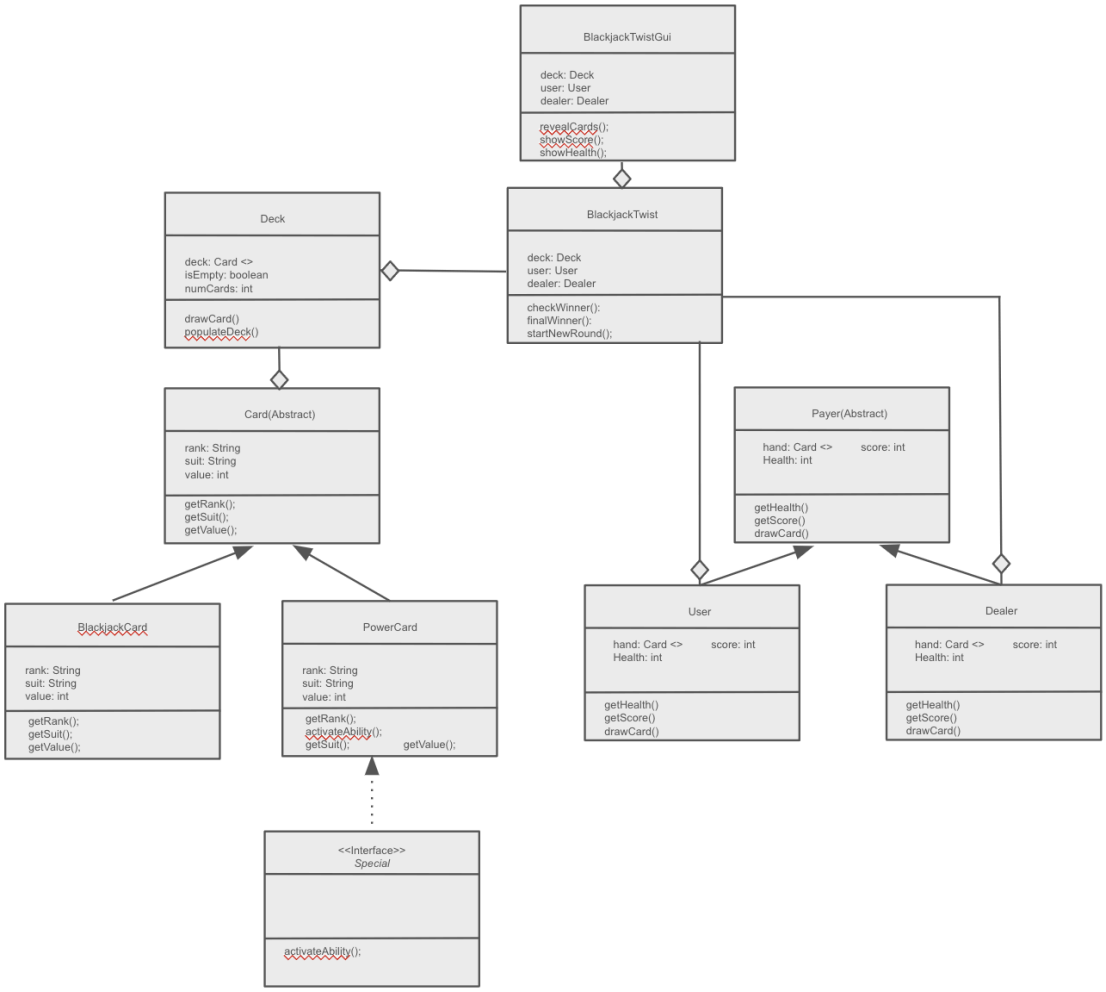
- **Card Dealing:** The player and dealer will each receive two cards at the beginning with the first card in each hand being visible.
- **Hit or Stand:** The player will be given the option to hit (draw a card) or stay (keep their hand). The player can draw a maximum of 12 cards. This prevents the user from drawing an absurd amount of cards because by these rules they can theoretically have an infinite amount of cards in their hand. This will also be implemented to discourage the user from holding on to a lot of power cards since they are carried over from round to round.
- **Dealer's Rules:** The dealer must keep choosing hit until they have 17 or more points. The dealer also wins ties to compensate for the fact that they don't get power cards.
- **Blackjack or Bust:** 21 is the best score you can get and you can keep drawing cards even if you bust(not recommended).
- **Power Card:** A power card of the winner choice will permanently stay in the winner hand until they use it.



The GUI will have two health bars representing the respective health level of the player and dealer. The Player uses two buttons to indicate whether they want to hit or stay during the game. There are also 3 buttons that pop up when the player wins a round where the player can choose to activate an ability for future rounds.

---

3. UML Diagram Overview



UML Breakdown:

4. **Core Game Classes:**
    - **Card** class to represent a card in the deck.
    - **Deck** class to represent a deck of cards.
    - **Player** class (abstract) to represent both the user and the dealer.
    - **User** class (inherits Player) to represent the player.
    - **Dealer** class (inherits Player) to represent the dealer.
    - **BlackjackGame** class to manage the game state (dealing cards, determining winners).
    - **BlackjackGUI** class to handle user interaction via the graphical interface.

CRC Cards:

Applying CRC Cards

CRC Card for Card:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Deck • Player • BlackjackCard • PowerCard

Applying CRC Cards

CRC Card for BlackjackCard:

Responsibilities	Collaborators
• Know the suit • Know the rank • Know the value	• Card

Applying CRC Cards

CRC Card for PowerCard:

Responsibilities	Collaborators
• Know the suit(category/vague description) • Know the rank(ability) • Know the value (0)	• Card

Applying CRC Cards

CRC Card for Special(Interface):

Responsibilities	Collaborators
• Generate ability for power cards	• PowerCard

Applying CRC Cards

CRC Card for Deck:

Responsibilities	Collaborators
• Populate the deck of cards • Deal a card	• Card • BlackjackTwist

Applying CRC Cards

CRC Card for Player(abstract):

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• User • Dealer • BlackjackTwist

Applying CRC Cards

CRC Card for User:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the player • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for Dealer:

Responsibilities	Collaborators
• Know when to add card to hand • Know the total value of hand • Know the health of the Dealer • Be able to reveal cards	• Player • BlackjackTwist

Applying CRC Cards

CRC Card for BlackjackTwist:

Responsibilities	Collaborators
• Know the rules of the game • Deal cards to players • Determine winner • Be able to start new game	• BlackjackTwistGui • Deck • User • Dealer

Applying CRC Cards

CRC Card for BlackJackTwistGui:

Responsibilities	Collaborators
• Show the interface of the game • Have buttons that do certain things • Display health	• BlackJackTwist

1. Design Principles of Object-Oriented Programming (LO1)

I will use object-oriented programming principles like, abstraction, and inheritance in the Blackjack game. Additionally, I will create classes for the Card, Deck, Player, and Black Jack game.

2. Use Single and Multidimensional Arrays (LO2)

I'll be using array lists for holding the cards in the deck and players' hands. A multidimensional array could be used to hold the game history, such as the player 's previous actions.

3. Object and Classes, Including Aggregation (LO3)

There will be multiple classes:

- Card (to represent a single playing card)
- Deck (to represent the deck of cards)
- Player (to represent the user and dealer, holding hands and score)
- BlackJackTwist this will contain the rules of the game and make sure it is that each round will follow the specified rules

I will use aggregation where the `Deck` class contains a collection of `Card` objects, and a `Player` object has a collection of `Card` objects.

4. Inheritance and Polymorphism (LO4)

- Player and Card will both be abstract and super class of other classes
- EX: User and Dealer will be subclasses of Player
- EX: BlackjackCard and PowerCard will be a subclass of Card

5. Generic Collections and Data Structures (LO5)

I will be using generic collections, such as array lists and stacks, to store cards in the deck and the players' hand.

6. Graphical User Interface (LO6)

I will use GUI for buttons, the background, the health bar, and cards in the users' hand

7. Exception Handling (LO7)

I'll use exception handling to ensure that the game handles invalid inputs. For example the user enters something other than "hit" or "stand" or somehow gets a blank power card and to manage situations like an empty deck or out-of-bounds issues.

8. Text File I/O (LO8)

We'll store game results like player's scores in a text file and read from it to load previous scores when starting the game.

Week 1	● Write the project proposal. ● Plan the object-oriented design, starting with CRC cards. Determine classes (with fields and methods) and interfaces and their responsibilities. (The "model" part of MVC.) ● Create the UML. ● Begin writing project page.
Week 2	● Write code for The Card, BlackjackCard, PowerCard, and Deck Class. ● Develop test cases and test code as it is written and comment on my current code. ● Update project page with progress details. ● Submit code written so far.
Week 3	● Finish writing all my classes except for the Gui. ● Design the GUI (sketch it out on paper) - include the design in the Weekly update! ● Update project page with progress details. ● Submit code written so far.
Week 4	● Test, test, test, debug, and test some more. ● Plan where exception handling is needed to ensure the program fails gracefully. ● Update project page with progress details. ● Submit code written so far.
Week 5	● Finish First Draft of Game ○ Implement Exception Handling ○ Write the GUI for the game ○ Incorporate File reading and writing to the game

Week 6	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 7	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.
Week 8	● Test, test, test, debug, and test some more. ● Update project page with progress details. ● Submit code written so far.

Github Repository

https://github.com/GitUser807/Lylecisc191project