THE BEST ENTERTAINMENT

Estd

2022

BRISCOLA

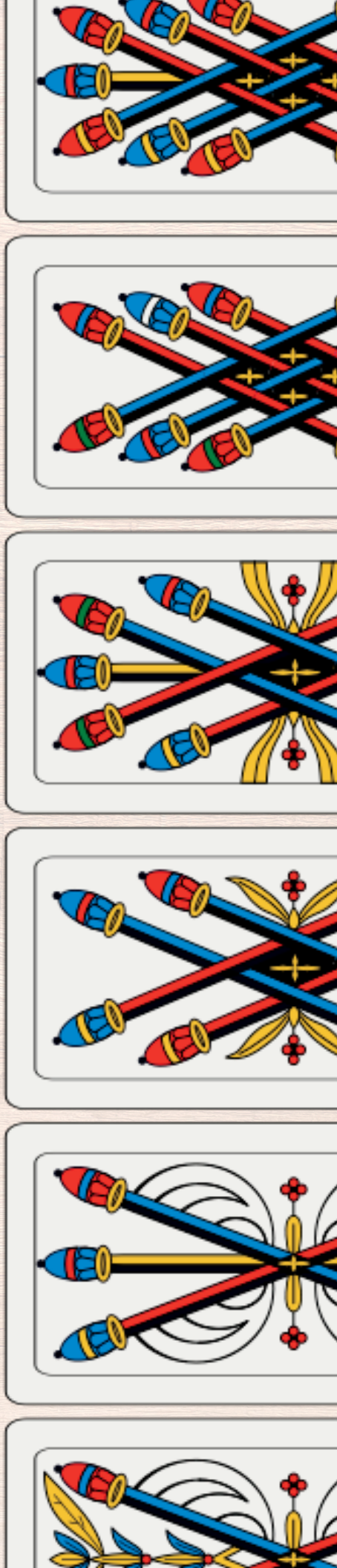
ITALIAN CARD GAME

BY DAVID MACY AND ABIGAIL BROWN

TABLE OF CONTENTS

OUR VISION

Our goal is to create a user-friendly application that allows people to play Briscola, a popular Italian card game, at any time and from anywhere in the world. By offering an engaging alternative to traditional card games like War and Go Fish, our app aims to promote cultural diversity and introduce players to a new gaming experience. We are committed to designing an intuitive and enjoyable platform that makes learning Briscola easy and accessible to everyone.



HOW TO PLAY

Briscola is a fast-paced card game that can now be played as an electronic game! This two player game features you trying to beat a computer opponent. The object of the game is to win more points than your opponent. There are a total of 120 points in the deck. Whoever scores more than 60 points will win! In the event of a tie, both players will end up with 60 points (for more information on points, see 'Card Points').

To begin the game, the 40 card deck is shuffled and both players receive three cards each. The top card on the deck is then flipped over and represents the trump suit for the rest of the game. Playing a trump card will automatically win the round!

During each round of the game, both players play one card. The cards are then compared to determine the winner. Below outlines some basic scenarios (see more game play scenarios on page 9).

- The cards played have different point values: the higher point value wins
- One card played is a trump suit: the trump card wins regardless of it's associated point value
- The point values are equal and the suits match: we look at the associated face value of the card (i.e., 5 of Coins versus 7 of Coins). The higher face value wins
- The point values are equal and the suits DON'T match: we look at the what card was played first (i.e., 5 of Coins vs 6 of Swords). The first card played becomes the automatic winner

POINT VALUES

In order to better understand the game, you must know what the cards' point values are! Below is a handy-dandy table filled with the associated point values for each card. If you're used to American Playing Cards, you'll notice that the points for an Italian Card Deck are quite different...

Point Values									
Ace	Two	Three	Four	Five	Six	Seven	Jack	Horse	King
11	0	10	0	0	0	0	2	3	4

As seen above, the points associated with the card do not correlate with the actual face value of a card. Look at the Four, for instance. The Four has a point value of zero! Yet the Three has a point value of ten! Keep this in mind while playing!

THE SUITS

Just like with American cards, Italian cards have 4 suits: Coins, Sticks, Cups, and Swords!

Coins

Sticks

Cups

Swords

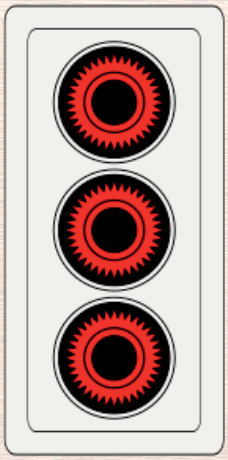
Now these suits are certainly different than American cards' suits: Hearts, Spades, Clubs, and Diamonds. However, with a little practice, you'll get used to the Italian cards in no time!

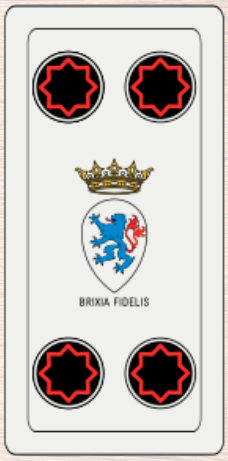
THE FACE NAMES

Briscola is played with the Ace, Two, Three, Four, Five, Six, Seven, Jack, Horse, and King! There are 10 cards per suit, thus giving us a 40 card deck.

Ace

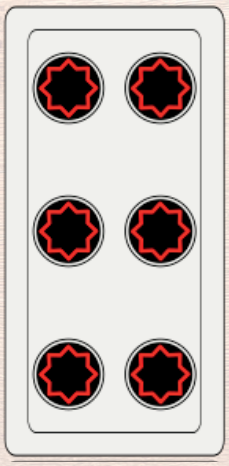

Two

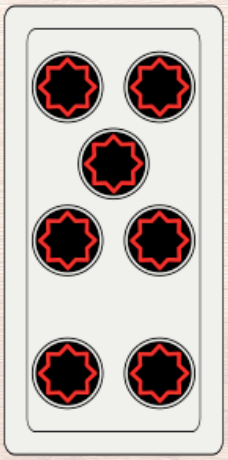

Three



Four


Five


Six


Seven

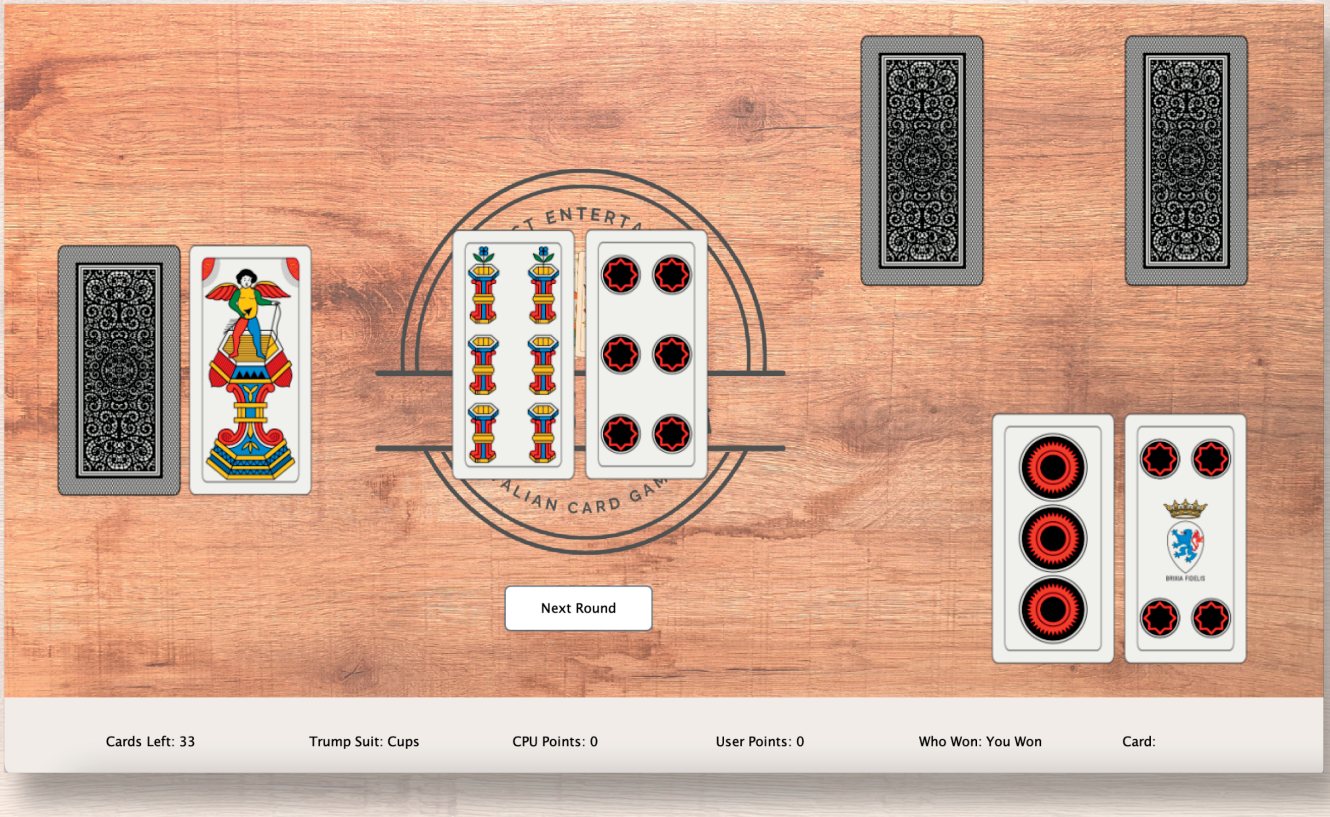

Jack

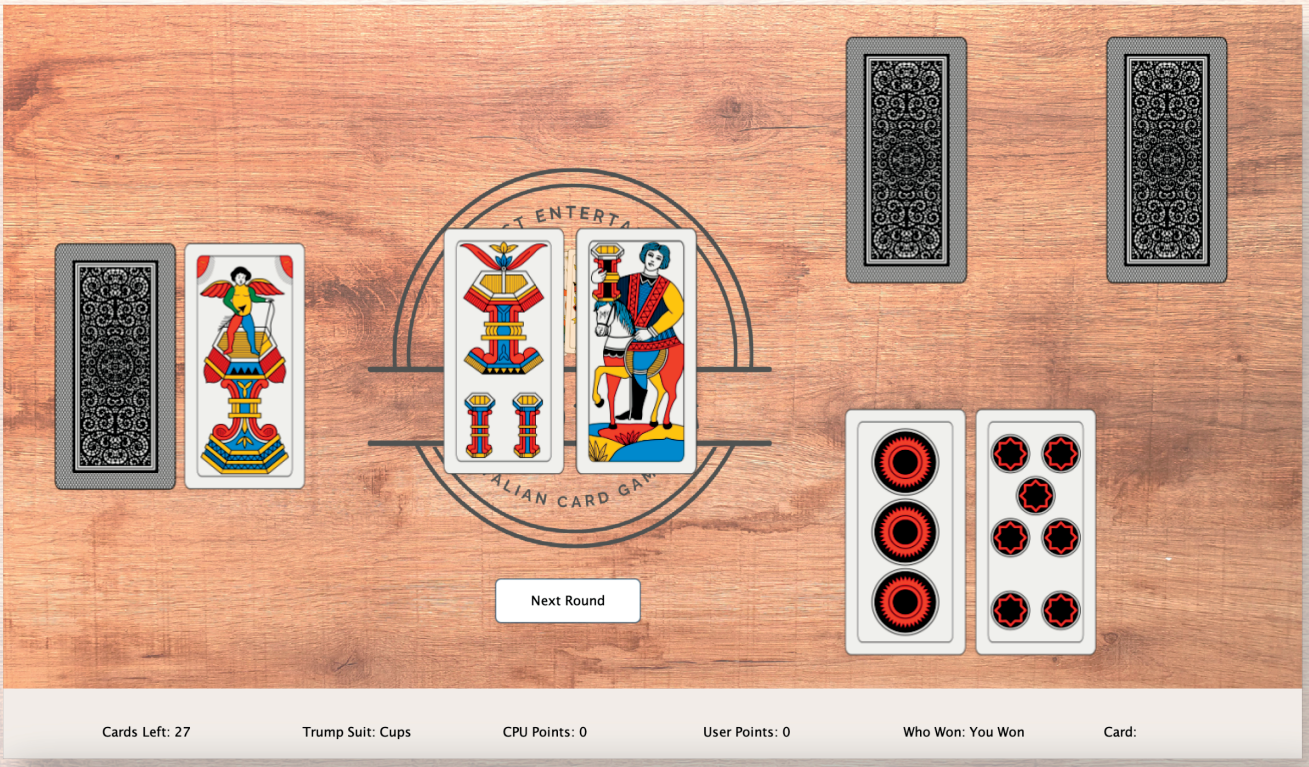
Horse


King

SCENARIOS

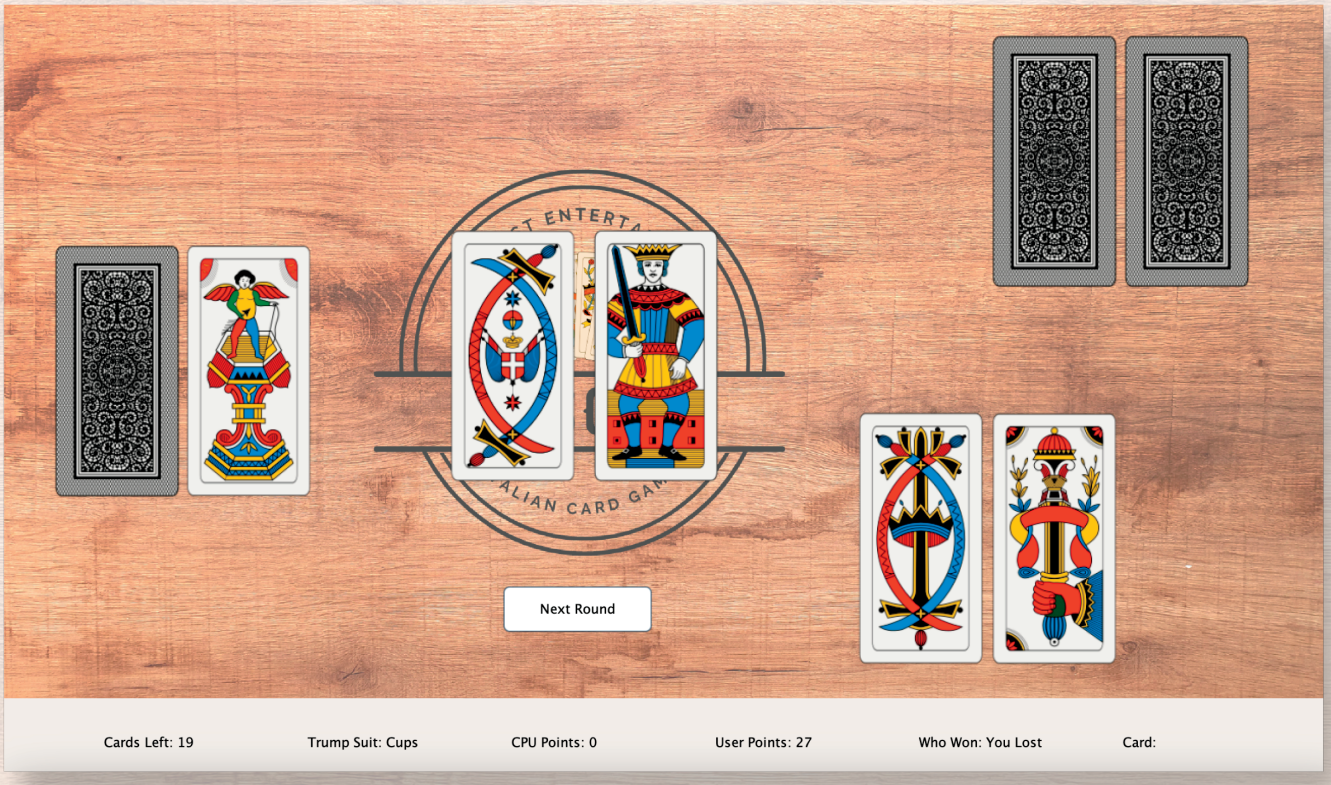
In the scenario pictured below, the player led off with the Six of Cups (the card in the center on the left). The computer opponent played the Six of Coins (pictured center right). Both the six of Coins and Six of cups have no associated point value. However, because the trump suit was Cups, the player won.





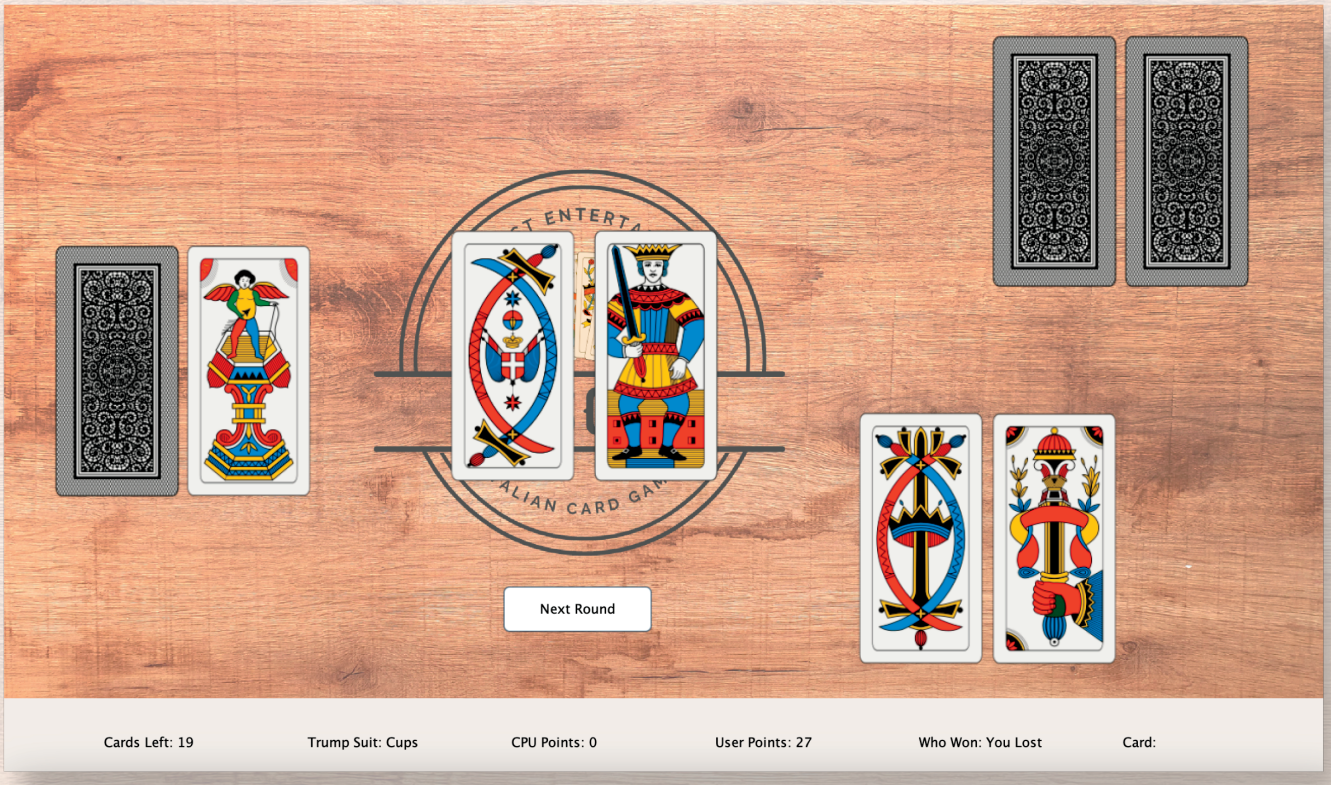
In the scenario pictured above, the computer opponent played the first card, which was the Horse of Cups. The player then played the Three of Cups. Both players matched the trump suit resulting in a need to look at the worth of the card (remember, when the suits match, look at the points). The Three of Coins has a point value of 10 points while the Horse of Cups is worth only 3 points. Because the Three has a higher point value, the player who played the Three will win the round and receive all 13 points.

Pictured below is a very common game scenario where the suits match and no trump cards are played. The player led off with the Two of Swords while the computer opponent played the King of Swords (the user's card played is pictured center left and the computer player's card is pictured center right). In this case, the point value for the King is far stronger, thus resulting in the computer winning the overall hand and receiving a total of 4 points for the round.



CONTROLS

To play, simply select either easy mode or hard mode (these buttons can be found on the main menu). Then, hit the deal cards button. You should see a game board like the one pictured below. Click on a card to play it. You can also hover over cards to see the card name. Plus, on easy mode, you can get strategy hints!

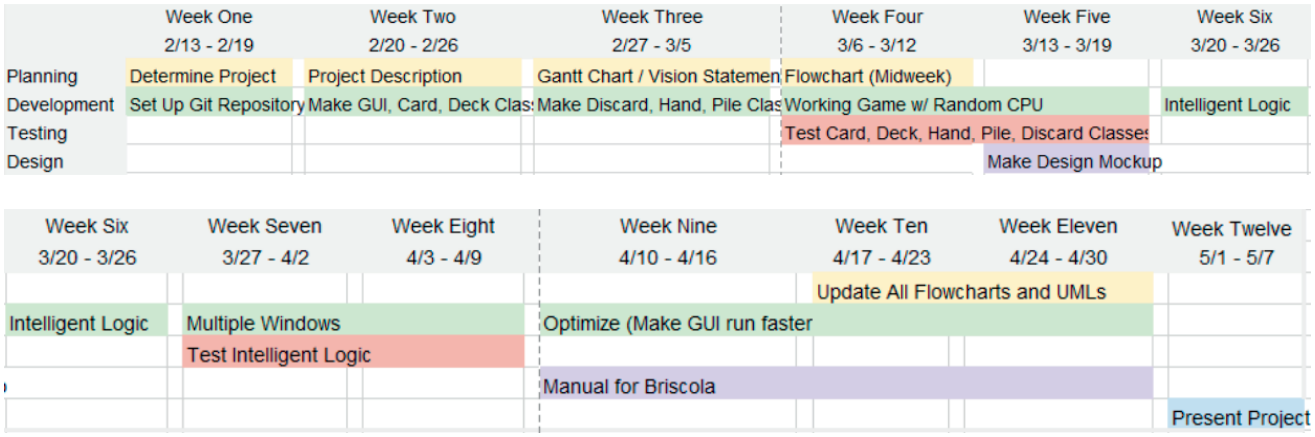


DEVELOPER SECTION

where we dive into the nitty gritty details of the code...

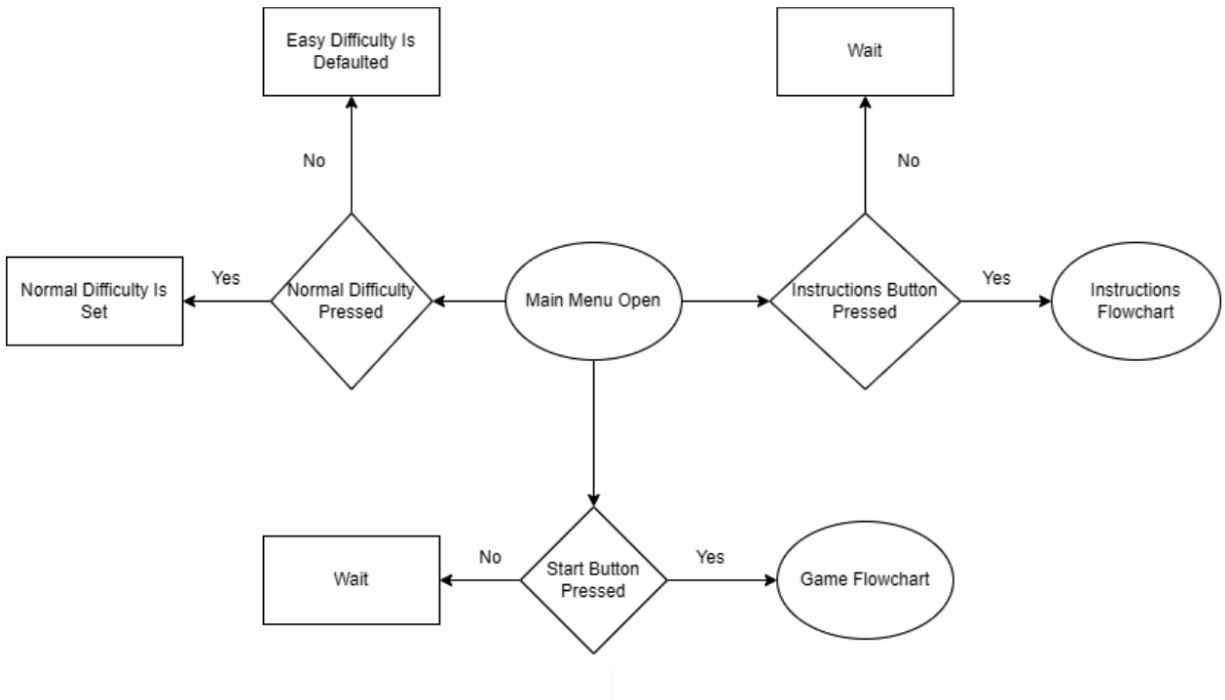
GANTT CHART

We stayed relatively on schedule for most of our game development. We followed through with proper testing protocol (i.e., unit testing and game play testing). Further, we were able to optimize our game by immensely reducing the file sizes of our cards. We even have multiple game window frames (i.e., instructions frame, game frame, and menu frame).



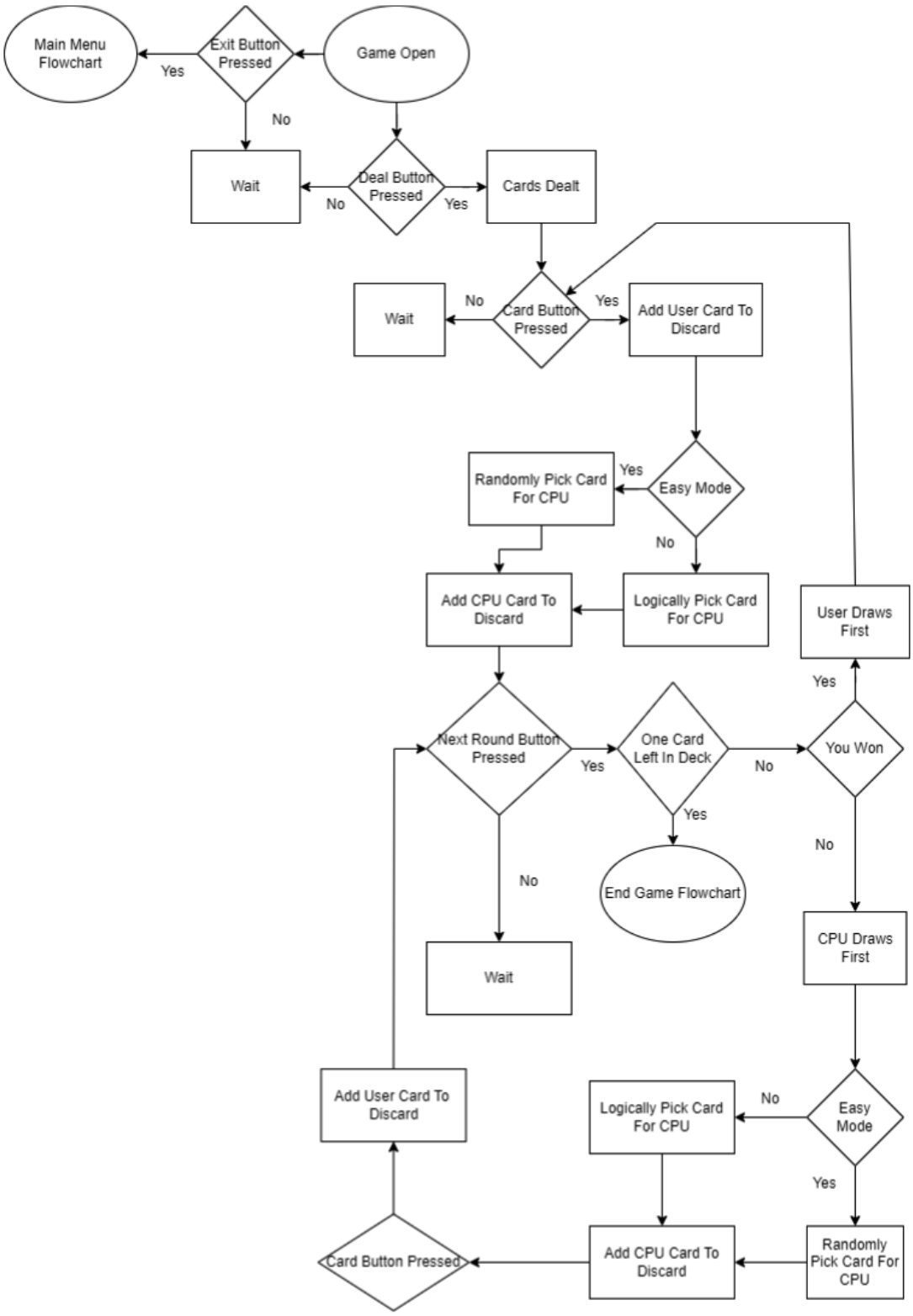
In the future, we would look into developing a way to save the win statistics for the user. This was a goal we were unable to reach.

MAIN MENU FLOWCHART

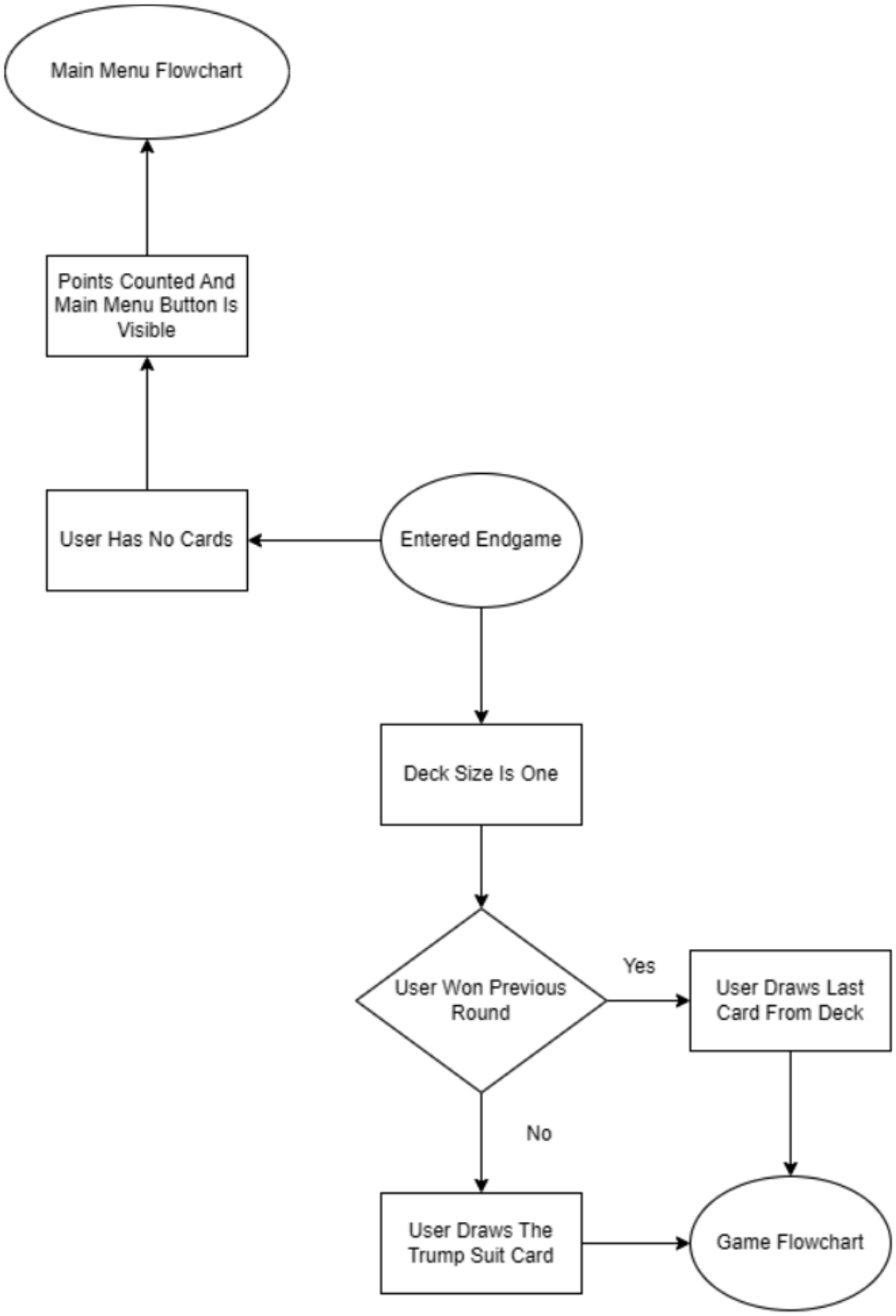


The flowchart above outlines our main menu operation. On the main menu, we check to see whether the easy or normal difficulty is selected. Then, we check if the user selected the instructions button. If they selected the start button, the game menu will pop up, allowing the user to begin playing the game.

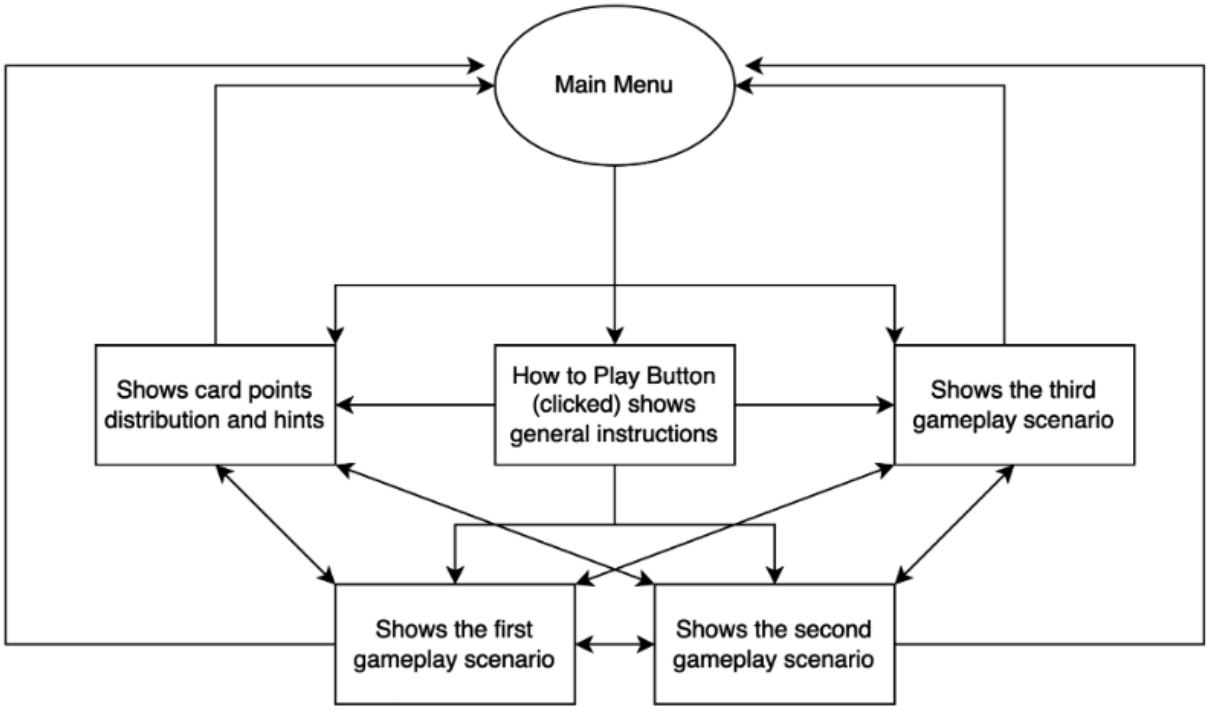
GAME FLOWCHART



END OF THE GAME FLOWCHART

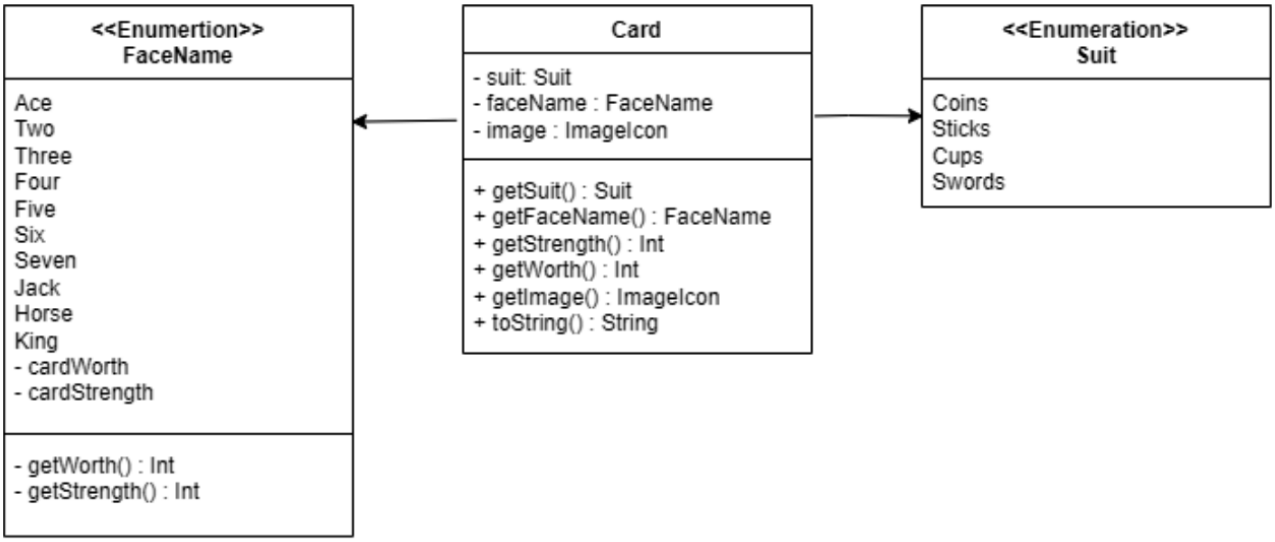


INSTRUCTIONS FLOWCHART



The flowchart above outlines our instructions operations. Every page that loads on the instructions main menu will link to each other. In essence, you can toggle back and forth throughout the instructions as you please.

UML: CARD

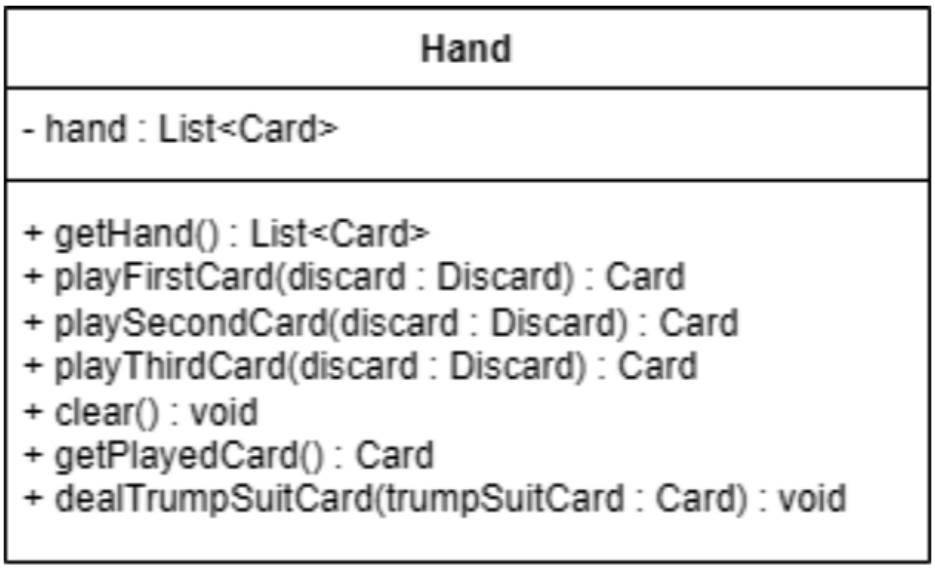


Creates a card object that will have a unique suit and faceName associated with it based on the Italian card game Briscola. Each card can be a Coin, Stick, Sword, or Cup with faceNames 2-7, jack, horse, king, and ace. Each faceName will have a worth and strength associated with the card. Worth is used to count up points at the end of the game whereas strength decides which card wins when the cards are played during the game.

UML: DECK

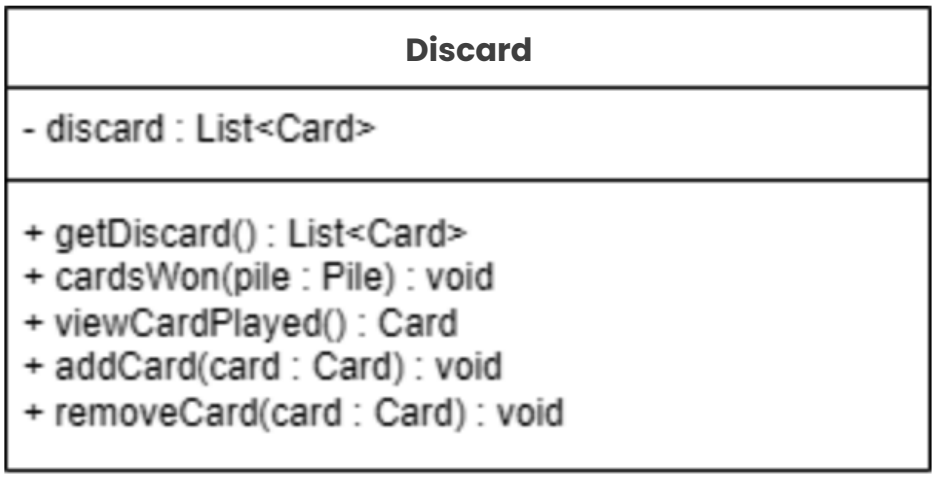
Creates a collection of cards that will represent the deck for the card game Briscola. Each deck contains 40 cards, 10 cards from each suit. The deck is shuffled every time it's created so it does not have to be done during the game. The functionality for this class will include dealing and drawing cards from the deck to the players hands.

UML: HAND

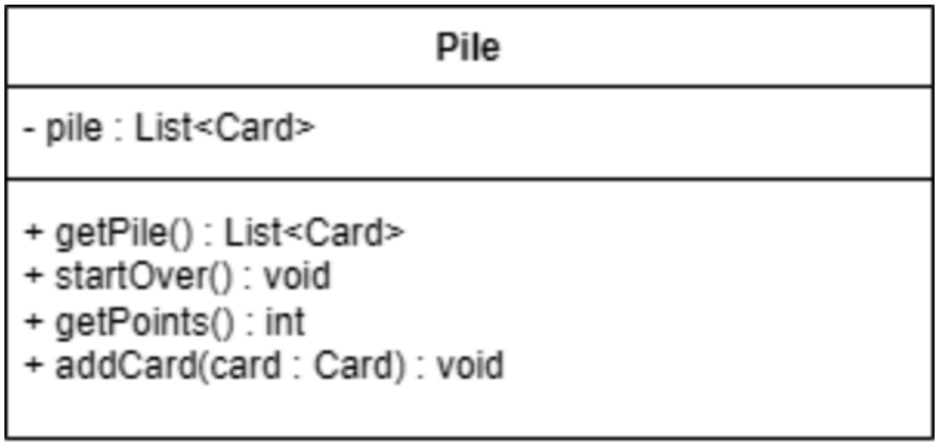


Creates a list of cards that will represent the hand for each player. The functionality of this class will include playing one of the three cards in the player's hands. Each player will only play one card per round. The user and the CPU will always have three cards in their hands unless the deck runs out of cards to draw from.

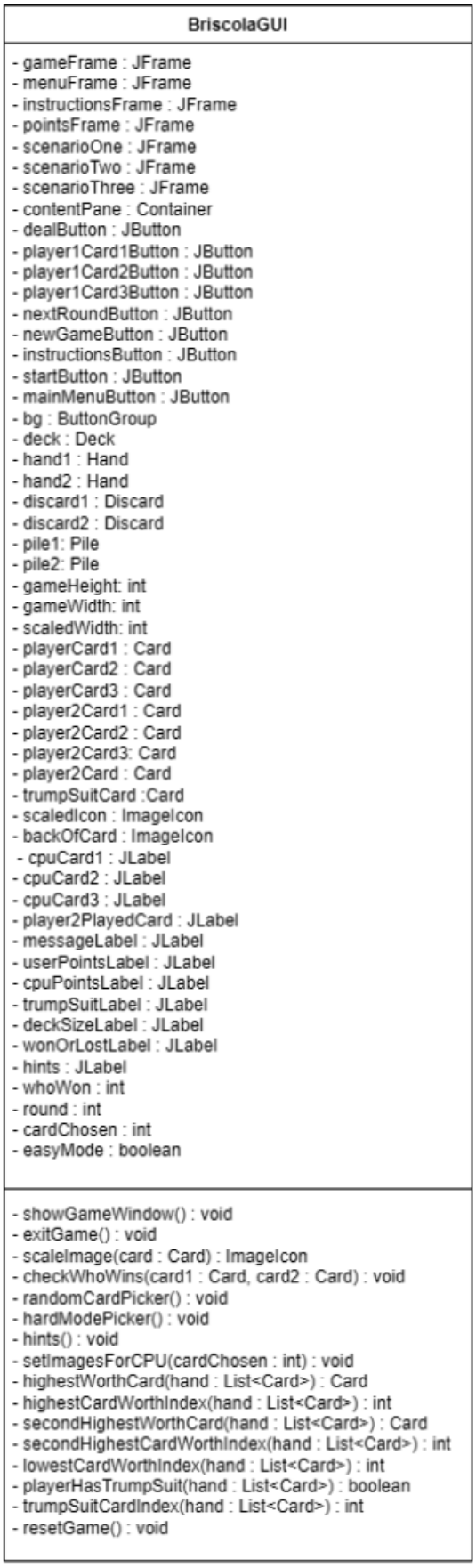
UML: DISCARD/PILE



Creates a list of cards that will represent the discard for each player. This is used to compare the two cards played by each player and evaluate who wins the round. Once a winner is determined cards are sent to the winner's pile.



Creates a list of cards that will represent the pile where all cards won will end up. Once the game is finished all card values are counted up to see who scores more points. More than 60 points guarantees a win while 60 is a draw.

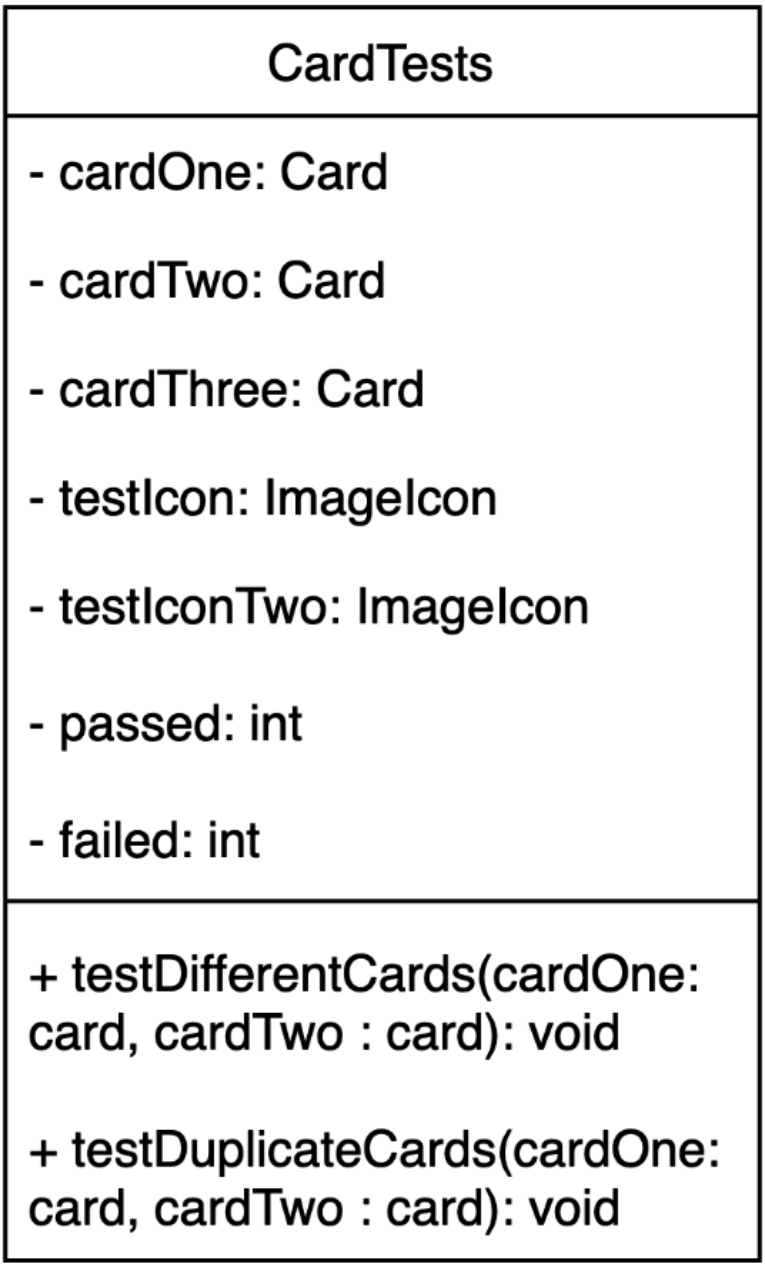


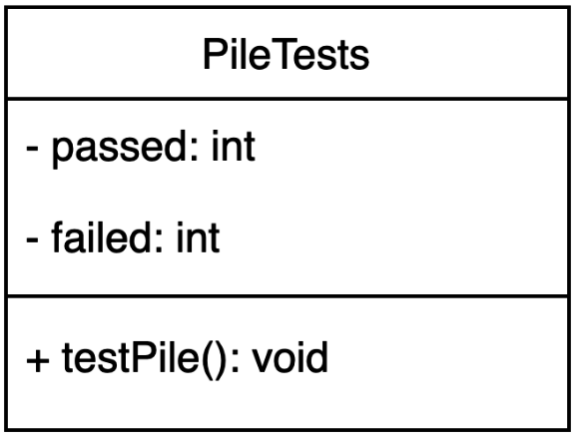
UML: THE GUI

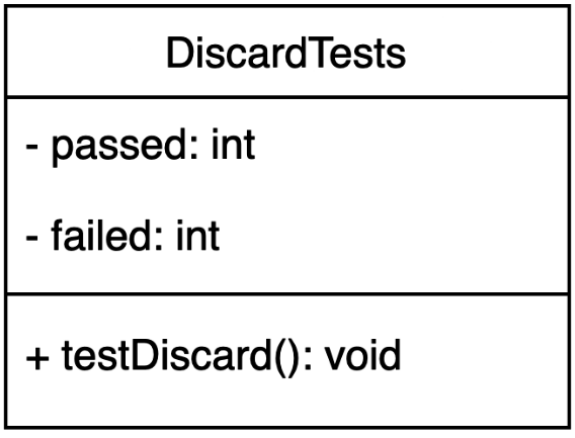
Controller class that creates the GUI and logic for the entire game of Briscola. This is going to be a 2P version of the game where the user begins every game. This controller controls three windows, the gameFrame, menuFrame, and the instructionsFrame. The menuFrame is where the user starts every time the application starts. In the menuFrame the user can select which difficulty they would like to play on, visit the instructions frame, or to start the game.

THE TEST UMLS...

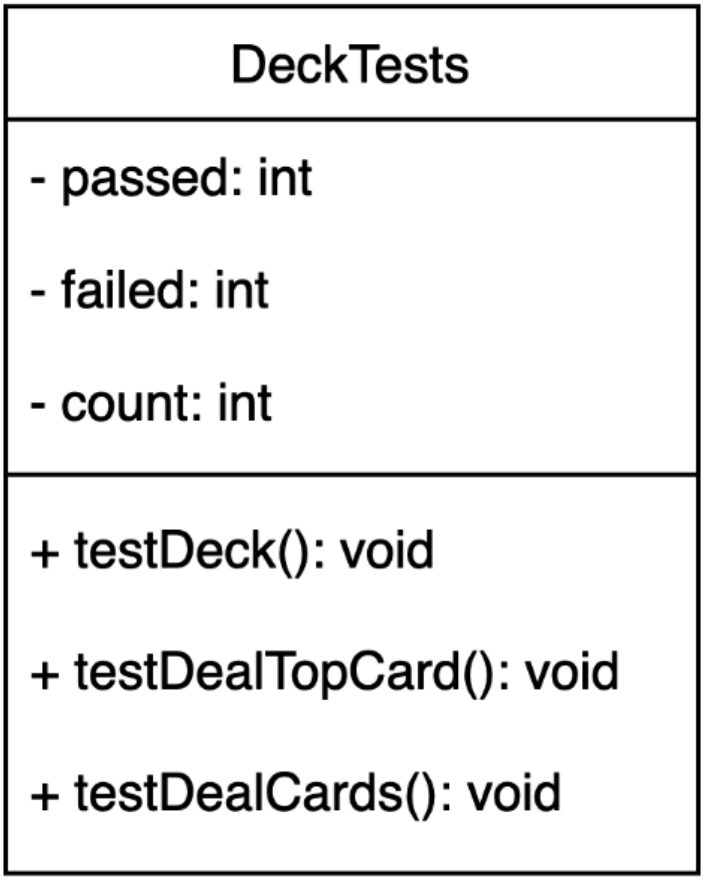
The following UML Diagrams depict the various testing classes. These classes extensively tested the Hand, Discard, Pile, Deck, and Card classes.

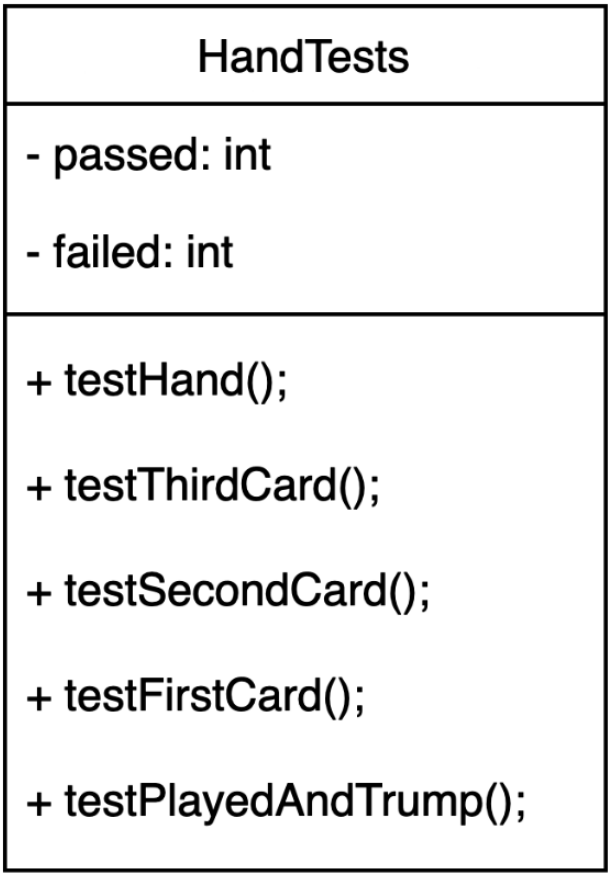






TEST UMLS CONT...





These test classes will show to the user the number of passed and failed cases. Should one of the cases fail, it will be noted which case(s) failed. Roughly 50 unit tests were completed, ensuring that our code is fit to be used!

IMPORTANT METHODS

checkWhoWins Method

```
private void checkWhoWins(Card card1, Card card2) {
   //if the user's card has the same suit as the CPU then whoever has the
stronger card wins.
   if (card1.getSuit().equals(card2.getSuit())) {
       if (card1.getStrength() > card2.getStrength()) {
           whoWon = 1;
       } else {
           whoWon =  2;
       }
   //if the user plays a card with the same suit as the trump suit card and
the CPU did not match the trump suit then
   //the user wins.
   } else if (card1.getSuit().equals(trumpSuitCard.getSuit()) &&
!card2.getSuit().equals(trumpSuitCard.getSuit())) {
       whoWon = 1;
   //The opposite applies here. If the user does not play a card matching
trump suit but the CPU does, the CPU wins.
   } else if (!card1.getSuit().equals(trumpSuitCard.getSuit()) &&
card2.getSuit().equals(trumpSuitCard.getSuit())) {
       whoWon = 2;
   } else {
       //if the user won the previous round and the CPU does not play a suit
matching the user or a trump suit card
       //then the user wins.
       if (whoWon == 1) {
           if (!card1.getSuit().equals(card2.getSuit())) {
               whoWon = 1;
           }
       //if it's the other way around then the CPU wins.
       } else {
           whoWon = 2;
       }
   }
}
```

Method for checking who wins a round. It compares two cards and changes the whoWon variable so the game understands which player won. If whoWon changes to 1 the user won. If whoWon changes to 2 then the CPU won. The parameters it takes are the two cards that will be compared to each other. Card 1 is always the user's card and Card 2 is always the CPU's card.

randomCardPickerMethod

```
private void randomCardPicker() {
   //if the hand size is one, only let the CPU choose its first card
   if (hand2.getHand().size() == 1) {
       cardChosen = 0;
   }
   //if the hand size is 2, let the CPU choose 0 or 1
   if (hand2.getHand().size() == 2) {
       Random random = new Random();
       cardChosen = random.nextInt(2);

   }
   //if the hand is full, let the CPU choose either 0, 1, 2.
   if (hand2.getHand().size() == 3) {
       Random random = new Random();
       cardChosen = random.nextInt(3);
   }
}
```

Method used for randomly choosing a number 0-2 when the game is set on easyMode. These three integers represent the possible choices a random opponent can choose from. Once it has chosen a variable it reassigns the global cardChosen variable. The cardChosen variable is used to set the images on the board for the CPU's played card.

hardModePicker

```
private void hardModePicker(Card player1Card) {
   //if CPU is going first then choose the card worth least in your hand
   if (player1Card == null) {
       cardChosen = lowestCardWorthIndex(hand2.getHand());
   } else {
       //If player one has played a king or a card worth more (assuming they
didn't play a trump card), and CPU has
       //a trump card in their hand, play the trump card.
       if (player1Card.getWorth() > 3 && playerHasTrumpSuit(hand2.getHand())
&& !player1Card.getSuit().equals(trumpSuitCard.getSuit())) {
           cardChosen = trumpSuitCardIndex(hand2.getHand());
       //If player one played a trump card, the CPU will play a card worth the
least from its hand.
       } else if (player1Card.getSuit().equals(trumpSuitCard.getSuit())) {
           cardChosen = lowestCardWorthIndex(hand2.getHand());
```

```
        //If the suit of the card that player one played equals the suit of the
card worth the most in the CPU's hand
        //and the cpu will win by playing that card, play that card.
        } else if
(player1Card.getSuit().equals(highestWorthCard(hand2.getHand()).getSuit()) &&
player1Card.getStrength() < highestWorthCard(hand2.getHand()).getStrength()) {
            cardChosen = highestCardWorthIndex(hand2.getHand());
        //same logic as above but if the cpu has two valuable cards it won't
ignore the second one and play it.
        } else if (hand2.getHand().size() > 1 &&
player1Card.getSuit().equals(secondHighestWorthCard(hand2.getHand()).getSuit()
) && player1Card.getStrength() <
secondHighestWorthCard(hand2.getHand()).getStrength()) {
            cardChosen = secondHighestCardWorthIndex(hand2.getHand());
        }
        //else play the lowest worth card in your hand
        else {
            cardChosen = lowestCardWorthIndex(hand2.getHand());
        }
    }
}
```

Method used for logically picking cards to play as the CPU on normal/hard mode. This is all done after looking to see what card has been played by the user. If the user is going second (which implies the CPU is going first) the CPU will choose to play the card worth least in its hand. Otherwise, it will see what logically makes sense to play.

hints

This method is only called when playing on easy mode. It modifies a JLabel on easy mode to give suggestions to the user who may be a little unsure on how to play. The logic is almost identical to how the CPU would play on normal mode since it's an effective strategy to play the game.

```
private void hints() {
   //if player two has not played a card suggest to play the card worth least
in the user's hand
   if (player2Card == null) {
       hints.setText("Try playing the card worth least");
   } else {
       //If player two played a trump suit card suggest to play the card worth
least in your hand
        if (player2Card.getSuit().equals(trumpSuitCard.getSuit())) {
           hints.setText("Try playing the card worth least");
       //If the CPU plays a card that is not of trump suit and your highest
worth card is greater then the card
       //the cpu played, suggest to play that card
       } else if
(player2Card.getSuit().equals(highestWorthCard(hand1.getHand()).getSuit()) &&
player2Card.getStrength() < highestWorthCard(hand1.getHand()).getStrength()) {
           hints.setText("Try playing the " +
hand1.getHand().get(highestCardWorthIndex(hand1.getHand())));
       //in any other scenario, suggest to play the card worth least
       } else if (hand1.getHand().size() > 1 &&
player2Card.getSuit().equals(secondHighestWorthCard(hand1.getHand()).getSuit()
) && player2Card.getStrength() <
secondHighestWorthCard(hand1.getHand()).getStrength()) {
           hints.setText("Try playing the " +
hand1.getHand().get(secondHighestCardWorthIndex(hand1.getHand())));
           //If the CPU plays a king, three, or ace, it's not trump suits, and
the user has a trump card in its hand
            // suggest to play the trump card
        } else if (player2Card.getWorth() > 3 &&
playerHasTrumpSuit(hand1.getHand()) &&
!player2Card.getSuit().equals(trumpSuitCard.getSuit())) {
            hints.setText("Try playing your trump suit card");
        } else {
           hints.setText("Try playing the card worth least");
       }
   }
}
```

UNIT TESTING

It was very important to test the logic of our code. It is necessary to have bug-free code in order to prove the best user experience possible. With that in mind, we created 5 testing classes to thoroughly test the Hand, Deck, Discard, Pile, and Card classes. Below are some notable snippets of code.

```
public void testThirdCard() {
    Card cardA = new Card(Card.Suit.Coins, Card.FaceName.Ace, "test.png");
    Card cardB = new Card(Card.Suit.Swords, Card.FaceName.Four, "test.png");
    Card cardC = new Card(Card.Suit.Sticks, Card.FaceName.Horse, "test2.png");

    // create an instance of discard so the cards in hand have a container to
go to
    Discard discard = new Discard();
    // create an instance of hand so the cards have a hand to go to
    Hand hand = new Hand();

    hand.getHand().add(cardA);
    hand.getHand().add(cardB);
    hand.getHand().add(cardC);

    System.out.println("Testing playThirdCard...");
    // play the third card and check that it was played
    Card third = hand.playThirdCard(discard);
    if (third == cardC) {
        System.out.println("   pass");
        passed++;
    } else {
        System.err.println("   failed playThirdCard");
        failed++;
    }

    // check that the hand size decreased
    if (hand.getHand().size() == 2) {
        System.out.println("   pass");
        passed++;
    } else {
        System.err.println("   failed playThirdCard");
        failed++;
    }

    // check that the discard size increased
    if (discard.getDiscard().size() == 1) {
        System.out.println("   pass");
        passed++;
    } else {
        System.err.println("   failed playThirdCard");
        failed++;
    }
}
```

The testThirdCard method fully tested that the correct card is being played. That is to say, if the user selected the third card in their hand to play, the program must recognize where the third card is in their hand and play that card. Very similar test cases exist for both testFirstCard and testSecondCard.

testDealTopCard

```
public void testDealTopCard() {
   // create two decks and a hand
   Deck deckA = new Deck();
   Deck deckB = new Deck();
   Hand hand = new Hand();

   System.out.println("Testing dealTopCard...");

   // Save the top card in a variable
   Card topCardA = deckA.lookTopCard();

   // deal the top card (removes it from deck)
   deckA.drawTopCard(hand);

   // check that the played card is the true top card
   if (hand.getPlayedCard() == topCardA) {
       System.out.println("   pass");
       passed++;
   } else {
       System.err.println("   failed dealTopCard");
       failed++;
   }
   hand.clear();

   // save the top card from deck be and check that the played card is the
true top card
   Card topCardB = deckB.lookTopCard();
   deckB.drawTopCard(hand);
   if (hand.getPlayedCard() == topCardB) {
       System.out.println("   pass");
       passed++;
   } else {
       System.err.println("   failed dealTopCard");
       failed++;
   }

   hand.clear();

   // check that the new top of the deck does not equal the played card
   deckA.drawTopCard(hand);

   if (hand.getPlayedCard() != deckA.lookTopCard()) {
       System.out.println("   pass");
       passed++;
   } else {
       System.err.println("   failed dealTopCard");
       failed++;
   }
}
```

The testDealTopCard method extensively tests whether the top card is the true top card of the deck. This pulls many methods from the Deck class to test, including lookTopCard and drawTopCard. We were able to determine that our deck was working as expected and the correct cards were being dealt!

Test Enumeration

```
public void testDifferentCards(Card cardA, Card cardB) {
    System.out.println("Testing High Level Cards (getStrength)...");
    // check that the test cards actually have the correct strength (based on face value)
    if (cardA.getStrength() == 9 && cardB.getStrength() == 8) {
        System.out.println("   pass");
        passed++;
    } else {
        System.err.println("   failed getStrength");
        failed++;
    }

    System.out.println("Testing High Level Cards (getWorth)...");
    // check that the card worth (point value) is correct
    if (cardA.getWorth() == 11 && cardB.getWorth() == 10) {
        System.out.println("   pass");
        passed++;
    } else {
        System.err.println("   failed getWorth");
        failed++;
    }
```

Finally, we made sure that the correct strengths and worths were assigned to each card. This means that the point value (the worth) of a card needed to be accurate and the strength (the faceValue) of a card must be tested. For instance, a Three must have a point value of 10 and a faceValue worth of 8 (meaning it is the 2nd highest valued card in the whole game).

CONCLUSION

Challenges

We had a very slow game performance due to large image file sizes. To fix this, we reduced the image sizes and simplified our designs.

We had some difficulty managing and sharing code in a group, which we quickly overcame by using Git as our source code manager.

In the Future...

We would love to add the ability to play against multiple CPUs for more variety and challenge. The physical card game version of Briscola allows for more than two players to play at any given time. Having more CPUs would make this game far more realistic.

We would also like to create another popular Italian card game, Scopa, to add more content and options for users.

It would be incredibly fun to allow the user to login to the game and be able to keep track of their stats. Then the user can see how they improve over time!

Animations to the cards would be very fun and bump our game to the next level.

FOR THE SOURCE CODE, VISIT:

HTTPS://GITHUB.COM/DJMACY/BRISCOLA PROJECT

THANK YOU! ENJOY BRISCOLA!