Aaron Miller, Johan Reyes, Nate Kirsch, Wesley Muehlhausen
CPSC 224
Card Farkle - Final Report
April 30 2020

Our implementation for the Card Farkle GUI program was rooted mainly in a class called Hand. This class consisted of an array that was made up of objects from a class called Card. This card object had private integer variables corresponding to that card's suit or value. It also had a boolean variable called available that determined whether the card was available for scoring which was used in the score class. These cards were put into the hand through a Deck class that created a virtual deck of cards through an Arraylist of Card objects with all of the values for each suit that you would expect to be in a deck of cards. So, random cards were then put into the Hand Arraylist from the Deck Arraylist eight times to create a new Hand of eight. In every function in our score class there was a Hand class parameter that the function would use to calculate the possible scores for that class.

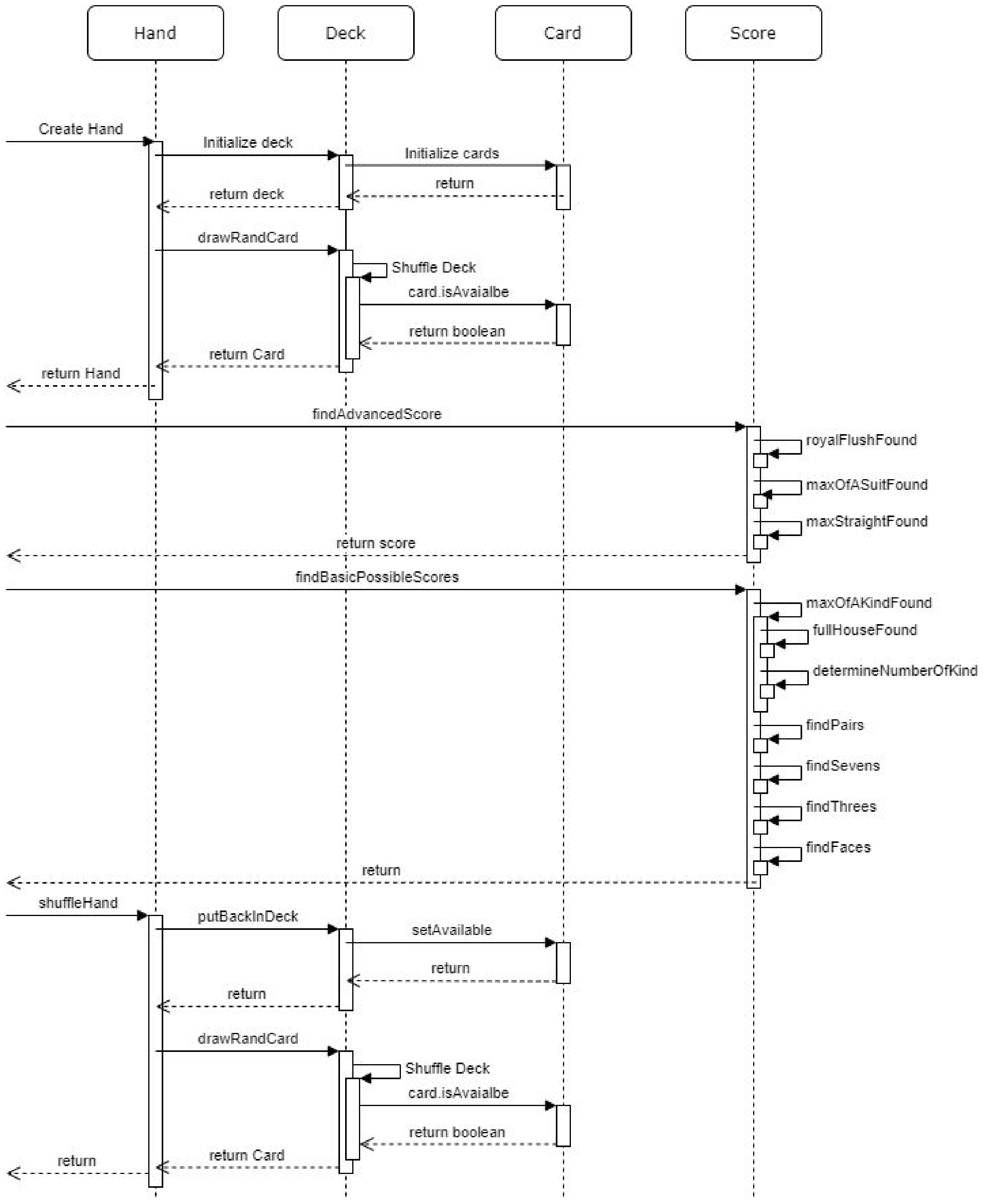
The GUI side of our project started with a Welcome Frame that had a combo box to determine how many players would play the game from one to four and four text fields for each player's name. The names then got passed into a String array that would then get passed into the MainFrame to allow the program to display the player names in the scoreboard and at the top of the screen for each turn.

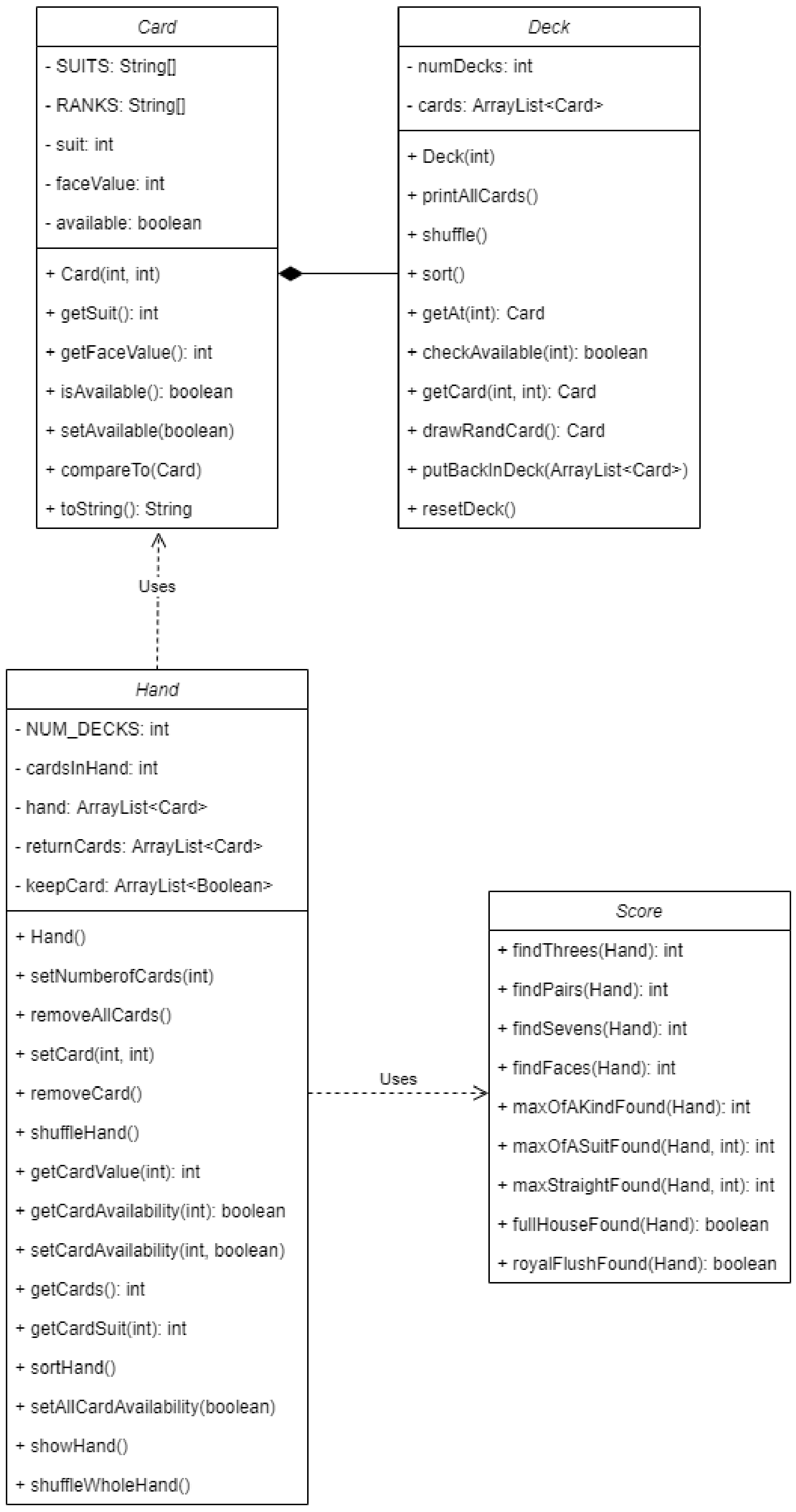
Like I said earlier, the Main Frame was rooted in the Hand class having two Hand Objects that were big parts of every aspect of the game, because they held the values for the cards that each player was playing with. Our first Hand object was just called hand and served as the object that stored the player's current hand. This object's arraylist of Cards matched up with an array of card toggle buttons we used in the program to portray the user's hand through objects on the cards and to allow them to click on the cards they would want to score. The actionPerformed for each toggle button used a toggle function that changed the background of each card clicked on from grey to blue which helped reference which buttons should be scored in other sections of the program. The main button that affected these buttons was the shuffle button which drew new cards for the hand using a shuffle hand function and changed the images of the buttons on each card. Also, this shuffle button removed card buttons based on how many cards were scored before the shuffle.

The second hand object was called scoreHand and was used to score the cards clicked on by the user. This happened after the user hit the score hand button which copied the values and suits of the cards clicked on into the scoreHand object's Arraylist which always resizes based on how many cards were clicked on. This hand is then run through a bunch of score functions starting with the royal flush, flush, and straight functions which were listed as advanced scores. These functions couldn't be scored with the other score functions because of problems that we couldn't figure out. So, if the scoreHand object couldn't be scored in any of these functions, then the program would run the rest of the scoring functions which would change each Card's available member variable to false when that card was scored so that no card would be scored more than once. The score for whatever cards were added to the scoreHand object would be added to an integer private member variable in the main GUI class called turnScore which would reset if the user farkled. After the cards got scored, the program would then remove those cards from the main buttons so the user would be able to know which cards could still be clicked on for scoring. If the user didn’t know what options they might have for scoring, we also had a scoring guide that pulled up a frame that explained all of the scoring options and the respective points affiliated with them.

We also made a Farkle popup that was pulled up whenever the player farkled meaning that there was nothing scorable in their hand. If the player was smart and hit the finish turn button before they farkled, then their turnScore would be added to the scorecard to the location that matched whatever turn they were on. This scorecard could be pulled up by pressing the scorecard button which would bring up a frame displaying the score for every turn played for each player and a total score for each player as well. This finish turn button would also reset the hand Hand object, change the player playing, and if needed update the current turn number. Once either the turn counter hit 21 meaning every player had played their 20th turn or a player reached 10,000 points, then this button would also pull up an end game popup that allowed the players to choose to play again.

(2)(3 **UML/Sequence Diagrams**:)


Hand
Deck
Card
Score
Create Hand
Initialize deck
Initialize cards
return
return deck
drawRandCard
Shuffle Deck
card.isAvaialbe
return boolean
return Card
return Hand
findAdvancedScore
royalFlushFound
maxOfASuitFound
maxStraightFound
return score
findBasicPossibleScores
maxOfAKindFound
fullHouseFound
determineNumberOfKind
findPairs
findSevens
findThrees
findFaces
return
shuffleHand
putBackInDeck
setAvailable
return
return
drawRandCard
Shuffle Deck
card.isAvaialbe
return boolean
return Card
return


Card
- SUITS: String[]
- RANKS: String[]
- suit: int
- faceValue: int
- available: boolean
+ Card(int, int)
+ getSuit(): int
+ getFaceValue(): int
+ isAvailable(): boolean
+ setAvailable(boolean)
+ compareTo(Card)
+ toString(): String
Deck
- numDecks: int
- cards: ArrayList<Card>
+ Deck(int)
+ printAllCards()
+ shuffle()
+ sort()
+ getAt(int): Card
+ checkAvailable(int): boolean
+ getCard(int, int): Card
+ drawRandCard(): Card
+ putBackInDeck(ArrayList<Card>)
+ resetDeck()
Uses
Hand
- NUM_DECKS: int
- cardsInHand: int
- hand: ArrayList<Card>
- returnCards: ArrayList<Card>
- keepCard: ArrayList<Boolean>
+ Hand()
+ setNumberofCards(int)
+ removeAllCards()
+ setCard(int, int)
+ removeCard()
+ shuffleHand()
+ getCardValue(int): int
+ getCardAvailability(int): boolean
+ setCardAvailability(int, boolean)
+ getCards(): int
+ getCardSuit(int): int
+ sortHand()
+ setAllCardAvailability(boolean)
+ showHand()
+ shuffleWholeHand()
Uses
Score
+ findThrees(Hand): int
+ findPairs(Hand): int
+ findSevens(Hand): int
+ findFaces(Hand): int
+ maxOfAKindFound(Hand): int
+ maxOfASuitFound(Hand, int): int
+ maxStraightFound(Hand, int): int
+ fullHouseFound(Hand): boolean
+ royalFlushFound(Hand): boolean

Business Function Modeling

Highest Level Function

- Play a 2 deck game of Card Farkle with multiple (2+) players

General Functionality

- Start a new game of Card Farkle
    - Display a start screen which does the following:
        - Ask the user how many players will be in the game
        - Create an empty running sum for each player
    - Initialize the card decks
- Player takes turn
    - Draws and displays 8 cards to the user
        - Determine if none of the 8 cards are scorable (Farkle)
    - Otherwise, allow the user to
        - Select cards to be kept or reshuffled
    - Check if user wants to score their current points, or to go for another hand
        - Reshuffle cards into hand
- Score hand of player
    - Adds points to players running sum based on hand
    - Display scores of each player on scoreboard
- Switch to next player's turn
    - Loop to Player takes turn protocol
- Finish a game
    - Determine if a player has reached 10,000 points
    - Display final scoreboard
    - Ask the user if they would like to play again
    - Start a new game if appropriate, otherwise quit

Problem Statement

To begin a new game of Card Farkle the first step is to Display the startup welcome screen that will ask the user how many players are playing in this game. Empty running sums will be created for every player, the decks of cards are then initialized and shuffled, and the main game screen is displayed after the user starts the game. The player's turn begins as eight random cards are drawn from the deck and displayed to the user. The displayed cards are first checked that there is a scorable option, if not, it is a farkle and any points that the user had summed up in their turn is discarded and the turn moves to the next player; otherwise, the user is hinted with possible scoring options that they can choose. Once the user selects the scoring options, they may choose to bank their score and end their turn, or they can choose to draw new cards from the deck equal to the number of cards not held for scoring. If the player chooses to bank their points, the game checks if it is the user's first points added to the board, if it is then the sum must be greater or equal to 500 points, otherwise the user must keep playing until the minimum first point threshold is reached. When the points banked, the score is added to the player's running sum, and the turn moves to the next player. Next to the panel displaying the cards in play, there is a scoreboard showing the up-to-date totaled scores for each player, as well as whose turn it currently is. After every player's turn the sums are checked against the winning point threshold of 10,000 points. The player that first crosses this threshold is tentatively elected the victor, as all players besides the player that crossed the point threshold get one more turn to try and upset the victor. The final scoreboard is then displayed and a message prompts the user to either play again, or to quit the game.

In our project there were a lot of parts that worked, many that did not, and different methods that the group could use to make the project better the next time. Some of the primary parts of the project that worked well include a recyclable Hand class, a scoring guide, as well as a user friendly GUI:

- Recyclable Hand class: In this program, the Hand class used the Deck and Card classes. The Hand would take Cards from the Deck when the shuffle button was pressed. Our Hand when shuffled would keep the Cards that the user wanted to be scored and would put the unwanted Cards back in the Deck instead of having to create a new Hand object every shuffle. This made our code very efficient and reusable.
- Scoring Guide: The scoring guide was also one of our best features. When the scoring guide JButton was pressed, a popup would show all of the scoring options in the game which would make it really easy for someone who was not familiar with the game to play.
- User friendly GUI: Our program included a fairly easy to use GUI. Besides the scoring guide, what made our user experience so good was that our program was very hard to break. Like with the startup GUI, the number of string entries would update based on how many players were selected. Or with the buttons in the main game frame. They could not be pressed unless they were activated, so for example you couldn't shuffle the hand 3 times in a row. Another thing that made our program flow was the cards. We chose to use JButtons as cards, rather than for example radio buttons, because they could be toggled very easily and looked the best.

There were also things that didn't work very well such as:

- Scoring and Hand class were hard to mesh together. We had problems with the hand being shuffled while also keeping certain cards that the user selected to keep. We also had a hard time implementing the hand into the scoring class.
- Flush and straight doesn't remove cards from GUI until the user hits the shuffle button. This is a minor problem but was found during testing.
- Buttons sometimes resize when removed and pictures take up more of the buttons depending on how many buttons are available. The buttons were set to be 100x140 pixels but the buttons adjusted to be 145x215 pixels and would go to the smaller size when the hand would be shuffled and change size.
- Problem scoring if the user clicks on a flush or straight and another scorable option.
- Bug with scorecard total if user farkles where it doesn't always set the total to 0.

What would the group do differently next time?

- Aim to finish the program a couple of days before deadline: We got the majority of the work done well before the deadline but the final touches we were scrambling to get done before the deadline.
- Better communication: We had fairly good communication overall but we had problems understanding how we were going to mesh our code together, such as the Scoring/Hand problem mentioned above.
- We would try to fix our problems with cards that would resize when the hand was shuffled.
- Create a better end game screen that displays who won instead of the player having to look at the scoreboard. Right now the end screen works great but it doesn't show the score and name of the person who won.
- Allow more players to play because Farkle can technically be played with as many people as possible. Right now we only have the game to where there are only 1-4 players allowed to play.

System Test Plan

Requirement ID	**Requirement**	**Verified By**
1A	<u>Start a new game of Card Farkle</u>	
1B	To begin a new game of Card Farkle the first step is to Display the startup welcome screen that will ask the user how many players are playing in this game.	Test Case 1 Step 1
1C	Empty running sums will be created for every player, the decks of cards are then initialized and shuffled, and the main game screen is displayed after the user starts the game.	Test Case 1 Step 2
2A	<u>Player takes turn</u>	
2B	The player's turn begins as eight random cards are drawn from the deck and displayed to the user.	Test Case 1 Step 2
2C	The displayed cards are first checked that there is a scorable option, if not, it is a farkle and any points that the user had summed up in their turn is discarded and the turn moves to the next player;	Test Case 1 Step 3
2D	otherwise, the user is hinted with possible scoring options that they can choose.	Test Case 2 Step 3

2E	Once the user selects the scoring options, they may choose to bank their score and end their turn,	Test Case 2 Step 4
2F	or they can choose to draw new cards from the deck equal to the number of cards not held for scoring.	Test Case 3 Step 4
3A	Score hand of player	
3D	When the points banked, the score is added to the player’s running sum, and the turn moves to the next player.	Test Case 2 Step 5
3E	Next to the panel displaying the cards in play, there is a scoreboard showing the up-to-date totaled scores for each player, as well as whose turn it currently is.	Test Case 2 Step 5
5A	Switch to next player’s turn	
5B	as well as whose turn it currently is.	Test Case 2 Step 6
6A	Finish a game	
6B	After every player’s turn the sums are checked against the winning point threshold of 10,000 points.	Test Case 2 Step 7
6C	The player that first crosses this threshold is tentatively elected the victor, as all players besides the player that crossed the point threshold get one more turn to try and upset the victor.	Test Case 2 Step 8
6D	The final scoreboard is then displayed and a message prompts the user to either play again, or to quit the game.	Test Case 2 Step 8

Test Case 1:
1. Starting a new game and enter amount of players in game
2. Start hand for a single user
3. Play hand to where there is no scorable option

Test Case 2:
1. Starting a new game and enter amount of players in game
2. Start hand for a single user

3. Play hand to where there is a scorable option
4. Select scorable option and bank score for that round
5. Switch to next player
6. Play until a player reaches 10,000 points
7. Give other players one more turn to get to 10,000 points

Test Case 3:

1. Starting a new game and enter amount of players in game
2. Start hand for a single user
3. Play hand to where there is a scorable option
4. Select scorable option and redraw unchosen cards