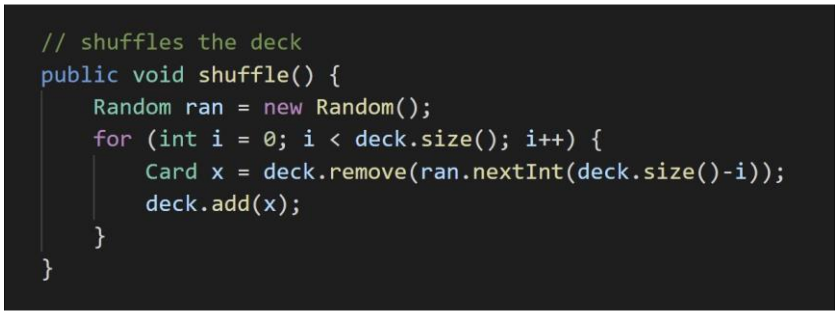
Linn Cao Nguyen Phuong
Project 1: Monte-Carlo Simulation: Blackjack
CS231
2/22/2021

Abstract

The main purpose of this project is to help students get used to Java, including learning Java's documentation and libraries, how to build classes and methods, use array lists, etc. by simulating a simple version of the card game Blackjack. Java is an object-oriented programming language, and everything in Java is associated with classes and objects, along with its attributes and methods. Thus, classes are very important in Java. For this project, there must be at least 5 classes – Card class, Hand class, Deck class, Blackjack class, and Simulation Class – that are connected together as appropriate. It requires students to import at least two classes – the Random (to shuffle the deck, pick random cards, etc.) and the ArrayList (to store cards in hands and in the deck) library classes.
For this project, other than the required classes, I also created Interactive class as an extension that allows the player to play the game in the terminal. To do this, I created a few more methods in the required classes and imported the Scanner class which allows for user input. This extension is connected to another extension – a method that automatically changes the value of Ace from 11 to 1 in both hands (for normal, automatic mode) and only in dealer's hands (for interactive mode).

Results

The Card class holds the value of the card, which must be in the range of 2-11 (or 1-11 if there are option values of 1 or 11 for the Aces). The Hand class creates an array list of type Card (ArrayList<Card>) to hold the Card objects. The Deck class also initializes an array list of type Card to hold a deck of 52 cards, 4 each of cards with values 2-9 and 11, and 16 cards with the value 10. One thing I would like to mention about my Deck class is my shuffle method that puts the deck in random order.

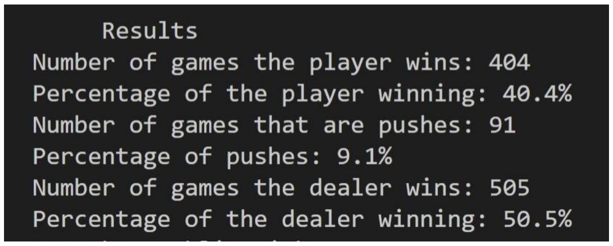


```java
// shuffles the deck
public void shuffle() {
    Random ran = new Random();
    for (int i = 0; i < deck.size(); i++) {
        Card x = deck.remove(ran.nextInt(deck.size()-i));
        deck.add(x);
    }
}
```

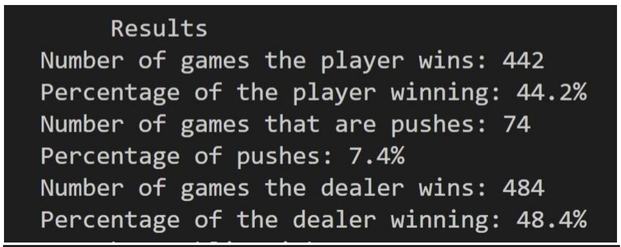
Blackjack class's shuffle method

Instead of building a fresh second deck which shall be added with the randomly removed cards from the first deck, I randomly remove cards from the range from 0 to (deck.size()-i). Specifically, the removed card will be added to the end of the deck and the already-removed-and-re-added cards are out of the range of the next cards to be removed. The main methods of the Card, Hand and Deck classes are only used to test the methods of these 3 classes.
On the other hand, running the Blackjack class shall give us a simulation of 3 games of blackjack in the terminal, each prints out the player's and dealer's initial hands, final hands and the result (player wins/dealer wins/push). Given the hands of the player's and the dealer's, I can check whether my blackjack game is running correctly. The result of the games is then sent to the text file mygames.txt.

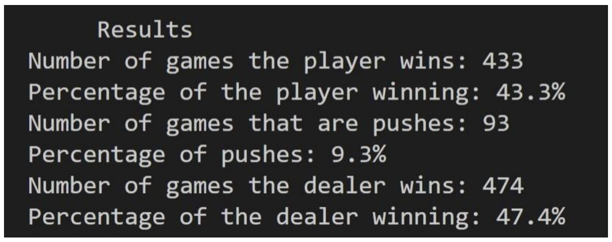
The Simulation class executes 1000 games of Blackjack in its main function, creating and reusing a single Blackjack object. After running the games, it prints out the results at the end that tells the user how many games the player wins, how many the dealer wins, and how many are pushes. It also includes a calculation of the percentages of the three types of results. The results I have after running the Simulation class 10 times are:



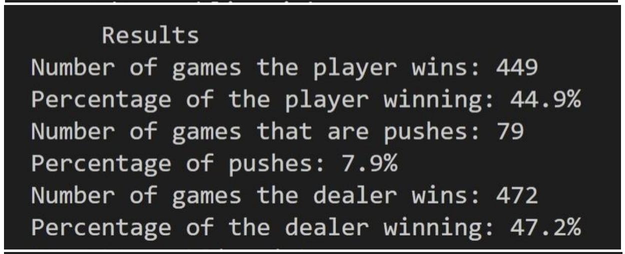
```
     Results
Number of games the player wins: 404
Percentage of the player winning: 40.4%
Number of games that are pushes: 91
Percentage of pushes: 9.1%
Number of games the dealer wins: 505
Percentage of the dealer winning: 50.5%
```



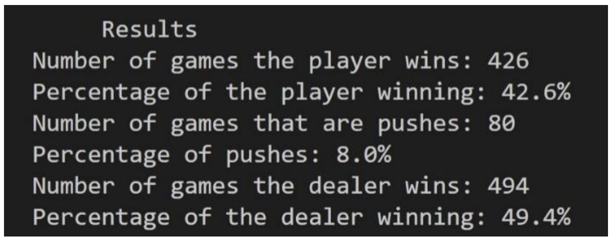
```
     Results
Number of games the player wins: 442
Percentage of the player winning: 44.2%
Number of games that are pushes: 74
Percentage of pushes: 7.4%
Number of games the dealer wins: 484
Percentage of the dealer winning: 48.4%
```



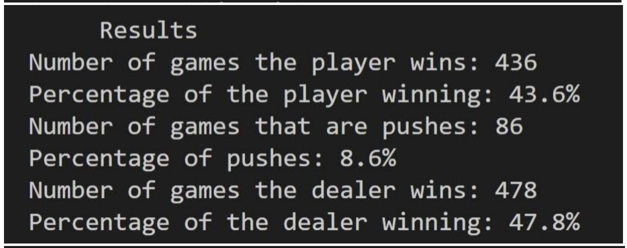
```
     Results
Number of games the player wins: 433
Percentage of the player winning: 43.3%
Number of games that are pushes: 93
Percentage of pushes: 9.3%
Number of games the dealer wins: 474
Percentage of the dealer winning: 47.4%
```



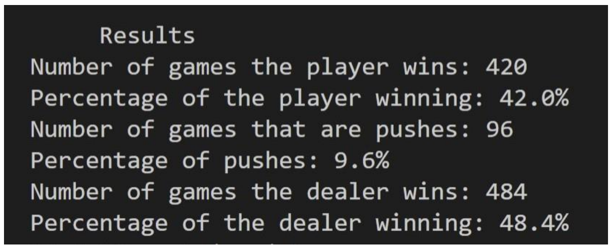
```
     Results
Number of games the player wins: 449
Percentage of the player winning: 44.9%
Number of games that are pushes: 79
Percentage of pushes: 7.9%
Number of games the dealer wins: 472
Percentage of the dealer winning: 47.2%
```



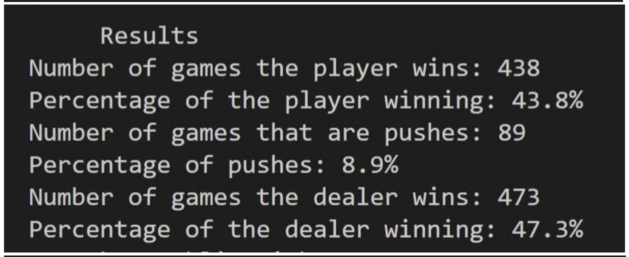
```
     Results
Number of games the player wins: 426
Percentage of the player winning: 42.6%
Number of games that are pushes: 80
Percentage of pushes: 8.0%
Number of games the dealer wins: 494
Percentage of the dealer winning: 49.4%
```



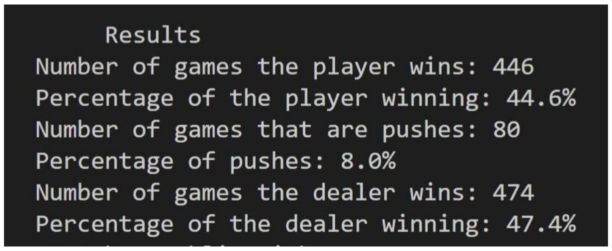
```
     Results
Number of games the player wins: 436
Percentage of the player winning: 43.6%
Number of games that are pushes: 86
Percentage of pushes: 8.6%
Number of games the dealer wins: 478
Percentage of the dealer winning: 47.8%
```



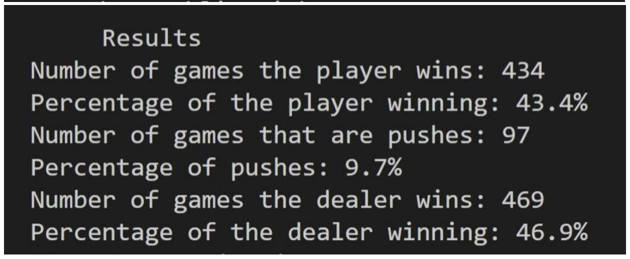
```
     Results
Number of games the player wins: 420
Percentage of the player winning: 42.0%
Number of games that are pushes: 96
Percentage of pushes: 9.6%
Number of games the dealer wins: 484
Percentage of the dealer winning: 48.4%
```



```
     Results
Number of games the player wins: 438
Percentage of the player winning: 43.8%
Number of games that are pushes: 89
Percentage of pushes: 8.9%
Number of games the dealer wins: 473
Percentage of the dealer winning: 47.3%
```



```
     Results
Number of games the player wins: 446
Percentage of the player winning: 44.6%
Number of games that are pushes: 80
Percentage of pushes: 8.0%
Number of games the dealer wins: 474
Percentage of the dealer winning: 47.4%
```



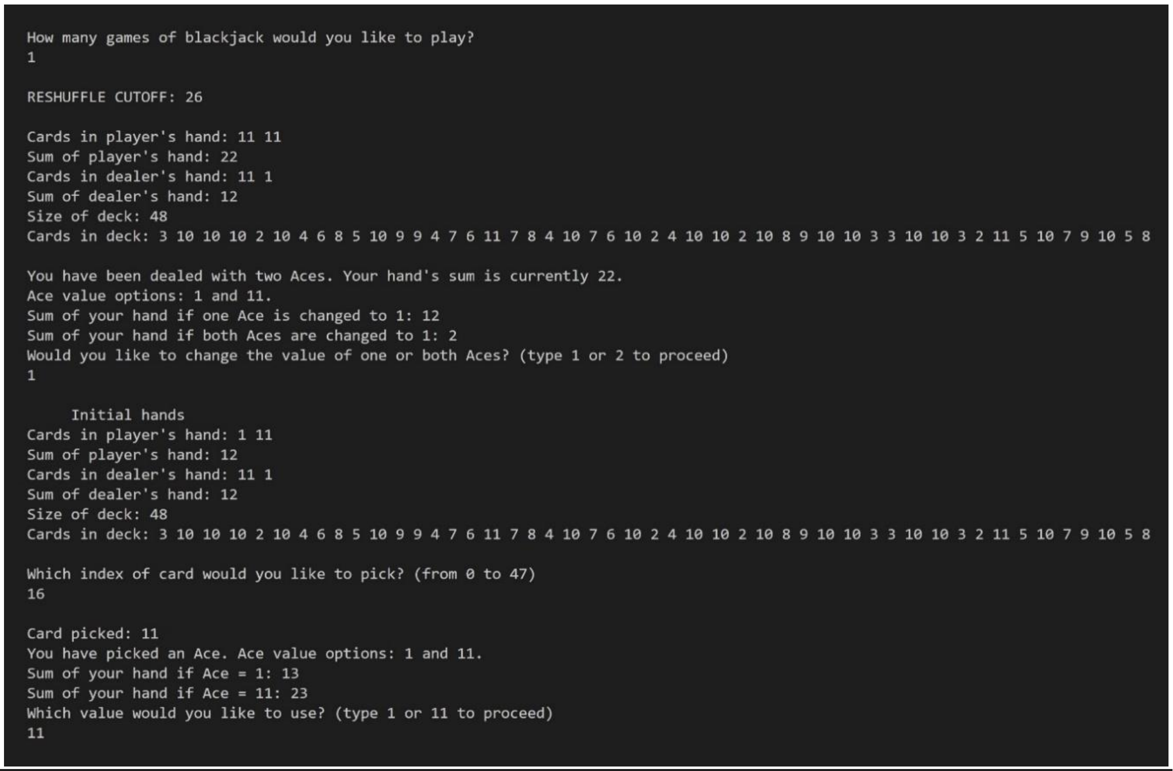
```
     Results
Number of games the player wins: 434
Percentage of the player winning: 43.4%
Number of games that are pushes: 97
Percentage of pushes: 9.7%
Number of games the dealer wins: 469
Percentage of the dealer winning: 46.9%
```

10 results of 1000 simulated games

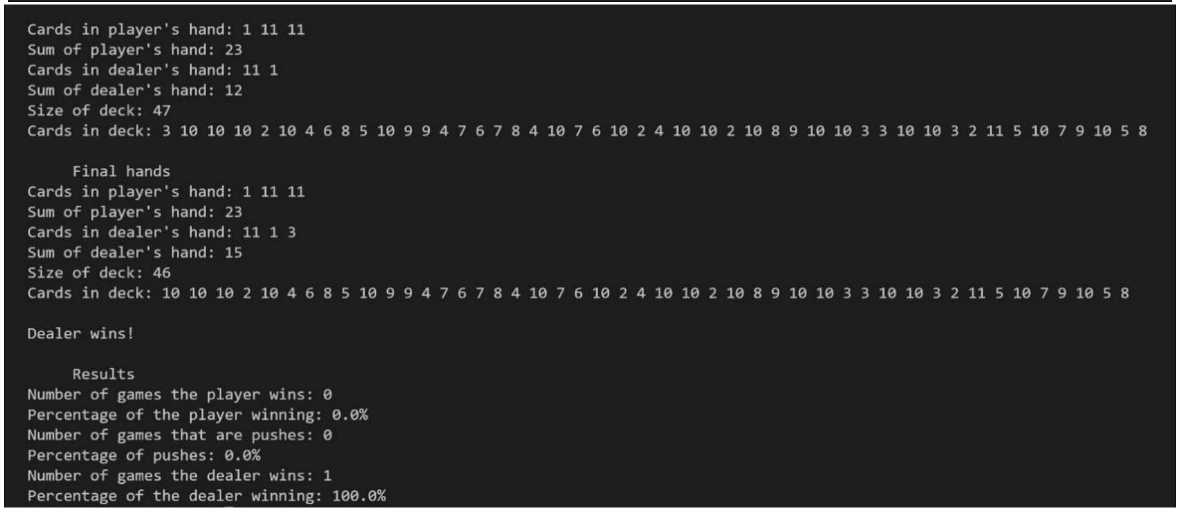
Looking at the final results of the games, we can see that the dealer wins notably more games than the player. This may be because the player has to finish with their hand first. If they bust, the dealer will automatically win the game. The percentage of push games is the lowest because there is a low chance of the two hands getting the same sum of cards or both busting.

Extensions

My first extension is my aceOptionAfterDealed, playerTurnInteractive and gameInteractive methods in the Blackjack class, together with my Interactive class. An interactive game will look like this (except from the Cards in deck line which is for me to easily pick the card I want to demonstrate my methods):



```
How many games of blackjack would you like to play?
1

RESHUFFLE CUTOFF: 26

Cards in player's hand: 11 11
Sum of player's hand: 22
Cards in dealer's hand: 11 1
Sum of dealer's hand: 12
Size of deck: 48
Cards in deck: 3 10 10 10 2 10 4 6 8 5 10 9 9 4 7 6 11 7 8 4 10 7 6 10 2 4 10 10 2 10 8 9 10 10 3 3 10 10 3 2 11 5 10 7 9 10 5 8

You have been dealed with two Aces. Your hand's sum is currently 22.
Ace value options: 1 and 11.
Sum of your hand if one Ace is changed to 1: 12
Sum of your hand if both Aces are changed to 1: 2
Would you like to change the value of one or both Aces? (type 1 or 2 to proceed)
1

     Initial hands
Cards in player's hand: 1 11
Sum of player's hand: 12
Cards in dealer's hand: 11 1
Sum of dealer's hand: 12
Size of deck: 48
Cards in deck: 3 10 10 10 2 10 4 6 8 5 10 9 9 4 7 6 11 7 8 4 10 7 6 10 2 4 10 10 2 10 8 9 10 10 3 3 10 10 3 2 11 5 10 7 9 10 5 8

Which index of card would you like to pick? (from 0 to 47)
16

Card picked: 11
You have picked an Ace. Ace value options: 1 and 11.
Sum of your hand if Ace = 1: 13
Sum of your hand if Ace = 11: 23
Which value would you like to use? (type 1 or 11 to proceed)
11
```



```
Cards in player's hand: 1 11 11
Sum of player's hand: 23
Cards in dealer's hand: 11 1
Sum of dealer's hand: 12
Size of deck: 47
Cards in deck: 3 10 10 10 2 10 4 6 8 5 10 9 9 4 7 6 7 8 4 10 7 6 10 2 4 10 10 2 10 8 9 10 10 3 3 10 10 3 2 11 5 10 7 9 10 5 8

     Final hands
Cards in player's hand: 1 11 11
Sum of player's hand: 23
Cards in dealer's hand: 11 1 3
Sum of dealer's hand: 15
Size of deck: 46
Cards in deck: 10 10 10 2 10 4 6 8 5 10 9 9 4 7 6 7 8 4 10 7 6 10 2 4 10 10 2 10 8 9 10 10 3 3 10 10 3 2 11 5 10 7 9 10 5 8

Dealer wins!

     Results
Number of games the player wins: 0
Percentage of the player winning: 0.0%
Number of games that are pushes: 0
Percentage of pushes: 0.0%
Number of games the dealer wins: 1
Percentage of the dealer winning: 100.0%
```

Extension 1: an interactive game of Blackjack

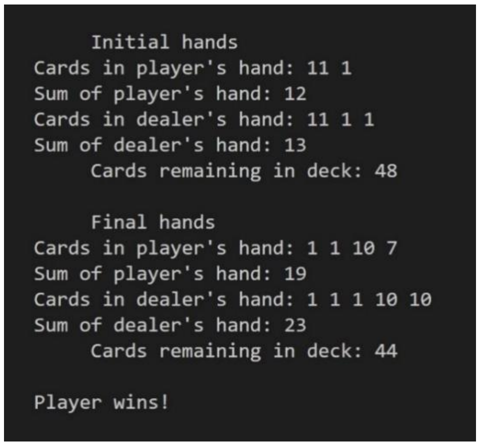
To use user input, I import the Scanner class in both the Blackjack and Interactive classes. In the Blackjack class, the method aceOptionAfterDealed allows the player to choose the value of Ace (1 or 11) if they are dealt with Ace(s). In the method's body, I first initialize a Scanner object and an int called aceOptionAfterDealt. I used an if statement to go through the two cards in the player's hand and check whether one of them has the value of 11. If the condition is satisfied, it continues to the inner if statement that checks whether both of the cards are 11. If this is true, the player will have the choice to change one or both Aces to the value of 1. I use the nextInt method of the Scanner class to take the user input and create a new card of value 1. If the user input is 1, the card of index 0 in the hand will be replaced with the card of value 1. (Note: the replace method is added to the Hand class by using the set method of the ArrayList class). Instead, if the user input is 2, both cards in the player's hand will be changed to the value of 1. Back to the outer if statement, if there is only one Ace in the hand, the player will be asked if they want to change the value of that Ace to 1 or keep it at 11. Afterwards, I create a new card of the value that the player has just typed into the terminal. Then, I check whether the 0 index card or the 1 index card has the value of 11 and set it to the value of the user input.

My second interactive method in the Blackjack class is playerTurnInteractive method that allows player to choose the index of card in the deck they would want to pick (after being dealt with 2 cards at the start of the game) and choose the value of Ace (1 or 11) after picking. I first initialize 2 Scanner objects for the index of Card and the value of the Ace, then two ints called pickOption and aceOption. While the sum of the player's hand is less than 16 and both hands' sums are less than 21, the player will be asked for the index of card they would like to pick, from 0 to (the deck's size-1). Then, I create a card that takes the value of the picked card and add it to the player's hand. I create a new card because the pick method removes the card from the deck, and if I keep calling this.deck.pick(*user input for index picked*), I will have different cards for each call. Then, if the value of the picked card is 11, the player is asked for the value of Ace they would want to use. Inside this if statement, I create an int that takes the value of the last index of the picked card in the player's hand to ensure that it is the card that has just been added. (Note: the lastIntdexOf method is added to the Hand class by using the lastIndexOf method of the ArrayList class). Afterwards, I create a new card with the value of the user input and set the picked Ace in the player's hand to the new card.

The gameInteractive method is similar to the game method. A difference between the two methods is that the gameInteractive method calls the aceOptionAfterDealt after the cards have been dealt and the dealer's hand has been checked by the autoAce method. Another difference is that inside the while loop, instead of adding cards to the player's hand, I called the playerTurnInteractive method. Thus, the gameInteractive method, as it plays a single game of Blackjack, also allows player to choose the index of card in the deck they would want to pick and to choose the value of Ace.

My Interactive class is similar to the Simulation class, except for that the Interactive constructor takes a parameter that is the number of games that will be executed. In the class's main method, I create a new Scanner and ask the player for the number of games they want to play. The Interactive constructor will then take in the user input and run the number of games the player wants.

My second extension is the autoAce(boolean interactive) method in the Blackjack class.



```
     Initial hands
Cards in player's hand: 11 1
Sum of player's hand: 12
Cards in dealer's hand: 11 1 1
Sum of dealer's hand: 13
     Cards remaining in deck: 48

     Final hands
Cards in player's hand: 1 1 10 7
Sum of player's hand: 19
Cards in dealer's hand: 1 1 1 10 10
Sum of dealer's hand: 23
     Cards remaining in deck: 44

Player wins!
```

Extension 2: autoAce method

This method automatically replaces the Ace(s) of value 11 for 1 if the hand's sum is greater than 21. In the normal, automatic mode (interactive == false), the method corrects both hands. Meanwhile, in the interactive mode (interactive == true), it only corrects the dealer's hand. This is because in the interactive mode, the player can choose the value of Ace that they want. Therefore, I think it would be more interesting if they have to think carefully about what value they should choose because they will not be able to change it later even if the value gets over 21. In the case of the photo given above (in normal, automatic mode), I intentionally replace all cards in both's hands with 11 and add one more card with the value of 11 to the dealer's hand after the cards are dealed. The method replaces one card in player's hand and two in dealer's hand so that the sum of both hands become less than 21. This method is also called after the hands have been added with other cards from the deck inside the while loop to make sure that if the hands' sum is reduced to lower than 16 for the player and 17 for the dealer, more cards will be added to the hands and the autoAce method shall do the checking again until the conditions of the while loop do not hold true anymore. To enumerate, the method checks if the sum(s) of the hand(s) are over 21. Should the if statement be satisfied, the method will go through the cards in the hand(s) to find the card that has the value of 11 and replace it with a new card with the value of 1 (that is created at the beginning of the method's body).

Reflection
Since this is our first project in Java, I spent a lot of time figuring out the Java documentation and how to use library classes. However, the project got easier as I moved on with the requirements, due to increasing familiarity with the documentation and the increasing knowledge we have from new lectures: from simple class building in the Card class and initializing objects such as ArrayList object to importing Java's library classes, using the classes' methods, and building our own methods. I also really enjoyed doing the extensions because I got to be more creative aside from following the report's requirements and explore the library classes and try to debug and fix logical errors!

References
I received help from Professor Al Madi who helped me clarify the project's instructions, Professor Skrien who provided me with extremely informative lectures, and Anna Chen who helped me get started with the project during the lab period. I also based the outline of my report on Olivia Schirle.