**Monte Carlo - Door Game**

The Door Game consists of three doors in which the player selects one door to acquire the money prize behind it, the host reveals one of the trap doors and asks if the player would like to switch after selecting their intial choice. This program finds the success probability of both possibilities of switching doors every time asked by the host or not switching at all. The DoorChoice java folder contains Doors.java and Gametest.java, Doors java file containing both game methods and Gametest running the measurement of 10,000 trials of each game type. Doors java file begins with the intialization of the win percentage variable and Random rand object. The first game method, no switching doors, has int trials as a parameter and has int wins variable that will count each win in the game loop. The game for loop runs the game for number of trials and sets both the player’s door choice and the prize money door to random ints in 1 to 3 range, simulating the 3 doors. If the door choice equaled the prize money door, the wins would increase, then calculating the win percentage by casting int wins divided by int trials as the double win percentage.

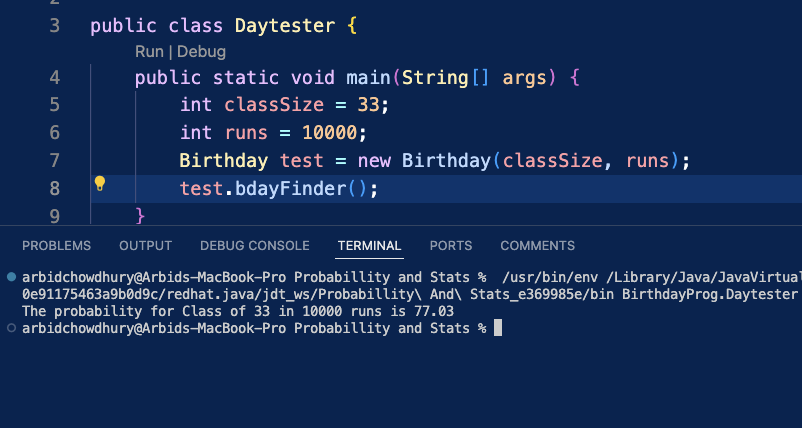
The switching door game where the player switches their initial door every time the host reveals the trap door has similar structure with the addition of the exposed door which is random int in 1 to 3 range. A while loop is used to reassign the exposed door if it was the same door as the door choice or money door, then the program initializes the switched new door choice. The second while loop assures that if the switched new door is the exposed or original door choice, the door will be reassigned. If that switched new door is now equal to the prize money door, the win count increases, and the win percentage is calculated the same way.



**Monte Carlo – Birthday Program**

The Birthday Program finds the probability of two people sharing the same birthday in certain class size, which in this case is 33. The program folder consists of 3 files being Person.java, Birthday.java, and Daytester.java which help generate People with birthdays, check/calculate probability of shared birthdays in the class, and to test the output which can adjusted outside the loop in Daytester. The Person class outlines the Person by initializing private int bday, constructor for Person with bday, and a getBirthday method. This is referenced in the Birthday file which first has private int Size/Runs, Random day object for birthday generation, and a constructor for birthday with those ints so that it can be modified in test file. Afterwards, the first of 3 methods is the classDays method which creates the class\_bdays ArrayList of Person objects. This method uses a for loop to generate Person objects with random birthdays based on 1 to 365 days of the year, the random int of 1 to 365 being the int parameter in each Person object.

The class\_bdays ArrayList is used in the next method being the bdayMatcher, which is a boolean method that uses a nested for loop to check each birthday for match. The nested loop compares each person in class\_bdays to every other person in the Arraylist, returning true if match is found during check. The third method bdayFinder which finds the probability of each match over the number of runs specified, initializing count first to count the matching birthdays, for loop creating a new class\_bdays every iteration and uses bdayMatcher to see if it has matches, in which count increases. Lastly, the probability of multiple random class iterations is calculated by casting int count divided by runs as a double.



**Monte Carlo – Pokémon Battler**

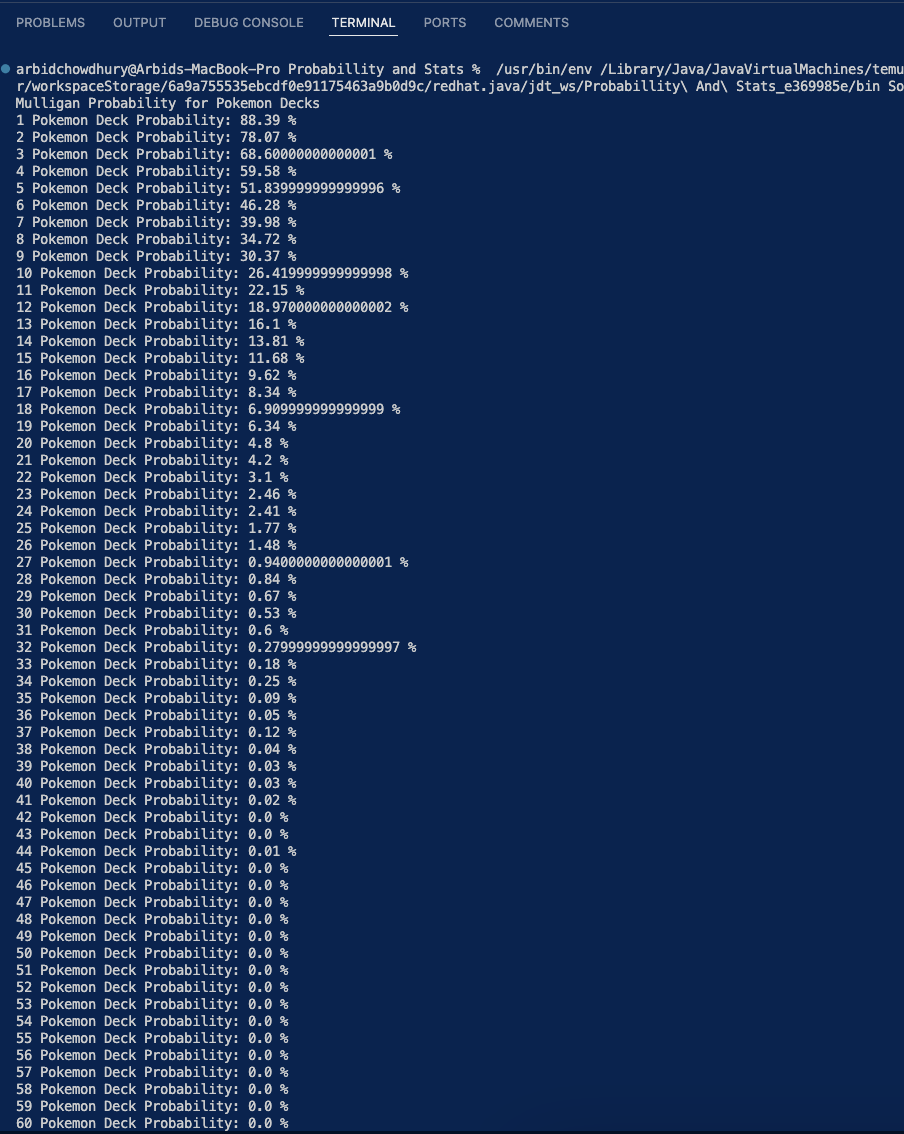
The Pokemon Battler simulates a simple battle between a Pikachu and Charmander and is groundwork for the Pokemon Project. The java folder contains Pokemon.java, Pikachu.java, Charmander.java, Stadium.java, and TestPokemon.java files. The Pokemon java file is a super class which contains all the information like hp, attack damage, defense, speed, and special attack damage which is not used but there to give Pokemon full stats. The file contains getters and setters for each attribute and this super class is inherited by Pikachu and Charmander files. The Pikachu and Charmander files both extend to Pokemon and use the setters to set their stats in their respective constructors. Stadium java file is where all the battle logic is, and the file contains two methods to simulate the pokemon battle attack turns. The battle method has a while loop ensuring the turns continue while both Pokemon p1 and p2 are above 0 hp. Then it checks the speed of the pokemon to see which one is faster, which determines which Pokemon attacks first. If the p1 is first then the helper method damageTurns lets p1 attack first and p2 second, then and checks if either Pokemon is below or at 0 hp. If p1 or p2 is below or at 0 hp the winner is printed out, this is all repeated if p2 is speed is faster but p2 would attack first. Essentially, the helper method damageTurns checks if the Pokemon p1 or p2 defense is greater than attack, in which either pokemon being attacked will remain at full hp. Otherwise, p2’s health with defense is lessened by attack damage from p1 and vice versa. The TestPokemon file creates a new Charmander BigChar, Pikachu BigPika, and a new Stadium tester object and calls the battle method to simulate the battle with BigPika and BigChar as the p1 and p2. The tester accurately displays the winner depending on the two Pokemons’ stats.



**Monte Carlo – Mulligan**

The Mulligan Probability program finds the probability of getting a mulligan, a hand without pokemon, depending on varying decks with 1 through 60 pokemon. The goal is to find the percentage chance of varying decks getting a mulligan to determine the optimal amount of pokemon for deck. This program is one half of the SoftwareCheckups folder which also contains Brick.java, Cards.java, and ProbTester. java files. The Cards java file is used by both Mulligan and Brick java files for generating decks. The Cards file enumerates the constant card types (Energy, Pokemon, Rare Candy), initializing the card type in its constructor and having a get type method. The Mulligan program begins with the deckMaker method which creates an ArrayList called deck with in pokeQuantity parameter. For loop adds the Card class’s Pokemon cards to deck depending on the pokeQuantity parameter. The second loop adds the remainder between pokeQuantity parameter and 60 to fill deck with Energy cards from the Card class. The deckShuffler method is called at the end to shuffle the deck using the Collections library shuffle command. drawCard method adds first 7 cards from deck to hand, removing it from deck.

Following is the drawHand method, which creates an ArrayList called hand and uses for loop to first 7 cards from deck to hand, removing the cards from the deck. Then there is the checkMulliganHand boolean method which uses a for each loop to check every Card in hand, returning false if there is at least one Pokemon card type in hand. The mulliganProbability method calculates the double percentage of the specific deck’s mulligans over 10,000 runs. The method initializes the mulligan count to 0 and then loops for 10,000 runs, which generates a new deck and checks if checkMulliganHand for drawHand is true to increase mulligan count. It then calculates the mulligan probability by casting the mulligan count over 10,000 runs as double which is returned by the method. The last method is the probabilityTest method which essentially loops mulliganProbability for 60 iterations, the int i of loop is the int pokeQuantity parameter effectively generating the 1-60 pokemon card decks and running each of those decks in mulliganProbability for 10,000 runs each.



**Monte Carlo – Brick/Rare Candy**

The Brick program finds the probability of getting a brick, all rare candies from deck in prize pile, depending on decks with 1-4 of rare candy cards. This file uses the same Cards file for deck generations and uses same methods from Mulligan java file like deckShuffler, drawCard, drawHand, and checkMulliganHand. The deckMaker method for Brick is slightly different from Mulligan, the difference being that the rareCandyQuantity parameter is how many rare candy cards are added to deck, and the rest of the 60 card deck being filled with Energy and Pokemon Cards. The deck is also shuffled, then there is the prizePileDrawer method which draws prize pile, if checkMulligan is false, using for loop to add 6 cards from deck to PrizePile ArrayList. Next is the checkPrizePile method which is exactly like the checkMulligan method but checks for each card in prize pile matching the rare candy card type, returning true if one is found. Following that, the candyPrizeProbabilty method calculates the probability of brick by first initializing the rare candy count to 0. Using for loop just like in mulliganProbability, the deck is generated and the boolean checkPrizePile of the generated prize pile is true, then the rare candy count increases and the probability is calculated/casted as a double just like in mulliganProbability. The last method is candyProbabilityTest which uses the same logic as probabilityTest from Mulligan, which is to increment the candyPrizeProbabilty method using the for loop’s int i, simulating decks with 1-4 rare candy cards, to find the brick probability of 10,000 runs for each deck.

