Mia Watts

**Monte Carlo Simulations Documentation**

**Pokémon Monte Carlo**

The classes associated with this assignment were designed based on the first Monte Carlo assignment in which we were to calculate the probability of drawing a Pokémon in our hand from 1-60 total Pokémon being added to the deck. The probability calculations show that the more Pokémon there are in the deck, the likelier it is to draw one in your hand. The classes in this simulation include the following:

* PokemonMonteCarlo
  + This class houses all of the methods associated with the Monte Carlo simulation. The methods are as follows:
    - evaluateOpeningHand() – Determines whether there is a Pokémon in the player’s hand. If there is, i.e., there is a card of type Pokémon in the hand, the method returns true and returns false otherwise.
    - drawHand() – Draws 7 cards from the deck to populate the player’s hand.
    - drawCard() – Randomly selects a card from the deck using Java’s Random class and returns the card.
    - newDeckMultiplePokemon() – This method creates a new deck, initializing the deck with the amount of Pokémon cards specified in the parameter list.
    - run() – Simulates the Monte Carlo Pokémon card assignment by taking a parameter that determines how many times the inner loop is run. The outer loop loops through the Pokémon card amount and the inner loop loops through the run amount specified in the parameter. At the end of each iteration, the loop prints the amount of Pokémon there are in the deck and prints the probability of drawing a hand with Pokémon after it.
* TestPokemonMonteCarlo
  + This test class initializes a PokemonMonteCarlo object and runs the simulation, passing the number of trials through the run method in the form of a parameter. The results are printed on the console and display the amount of Pokémon in the deck along with the percentage of hands with Pokémon.

**Rare Candy Monte Carlo**

These classes were designed based on the second Monte Carlo assignment in which we were to add 1, 2, 3, then 4 rare candies into a deck, then calculate the probabilities of getting a bricked deck in which all the candies were in the prize pile and there were no Pokémon in the player’s hand. There were two classes associated with this, those being described below.

* RareCandyMonteCarlo
  + This class houses all the methods associated with the run method which actually performs the simulation. A constructor is used to initialize the deck, hand, and prize pile array lists, and the methods in the class include the following:
    - newDeckMultipleCandies() – This method creates a brand new deck that contains more than one candy (or just one) depending on the number passed in the parameter. In the case of the run method, the variable i is passed as the parameter, making it so 1, 2, 3, then 4 candies are added with each pass of the outer loop. The deck is filled with the proper amount of candies, 15 Pokémon, 15 Energy cards, and the rest are Trainer cards.
    - evaluateOpeningHand() – This particular method checks to see whether the opening hand drawn has Pokémon in it. In the actual card game, if a drawn hand has no Pokémon, the hand must be redrawn. So, this method, for the case of the simulation, makes sure there are Pokémon (at least 1) in the deck, otherwise, it will return false.
    - evaluatePrizePile() – The evaluation of the prize pile is essential for the simulation because the prize pile cannot contain all candies or else the player’s hand is bricked. This method checks to see whether the prize pile has rare candies in it, and if it does, it returns true for having rare candies and false otherwise.
    - drawHand() – This method draws 7 cards from the deck and places them into the player’s hand.
    - drawPrizePile() – This method draws 6 cards from the deck and places them into the player’s prize pile.
    - drawCard() – This method draws a card from the deck randomly and returns it using the Random class in Java.
    - run() – The run method is the one that actually does the Monte Carlo Simulation. Having an outer loop that loops through the number of candies and an inner loop that loops through the amount of trials the user wants to do, the method calculates the probability that the deck will be bricked and returns the percentage along with how many hands or prize piles actually had rare candies. Methods are used within the loop to check whether the player’s hand is bricked, checking to see if all the candies are inside the prize pile or if there are no Pokémon in the player’s hand.
    - openingHand() – This method checks to see if the hand has a Pokémon in it, and if it does, it continues, but if it doesn’t, it redraws the hand until there is a Pokémon in it. ***This method was implemented with the help of Jake Cubernot.***
    - restartOpeningHand() – The restart method places all cards in the hand back in the deck and shuffles the deck. ***This method was implemented with the help of Jake Cubernot and was written originally by him.***
    - isCandyBricked() – This method checks to see if the hand the player has is bricked. If there are rare candies in the prize pile, it returns false, and if there aren’t, it returns true. ***This method was implemented with the help of Jake Cubernot.***
* TestRareCandyMonteCarlo
  + This was the tester class of the RareCandyMonteCarlo class. Here, an object is created that can be used to run the simulation, and the run method is called with the parameter being filled based on how many times the user would like to run the simulation. The results are printed on the console when the program is run.