TCG: PROJECT REPORT

Mouaz Ali

1. Introduction

The Pokemon Card Game Simulation is a comprehensive Java-based application developed to facilitate in-depth analysis and simulation of gameplay scenarios within the Pokemon trading card game (TCG). The application aims to provide players with valuable insights into the statistical probabilities and dynamics of various game situations, thereby assisting them in making informed decisions and optimizing their gameplay strategies.

2. Purpose

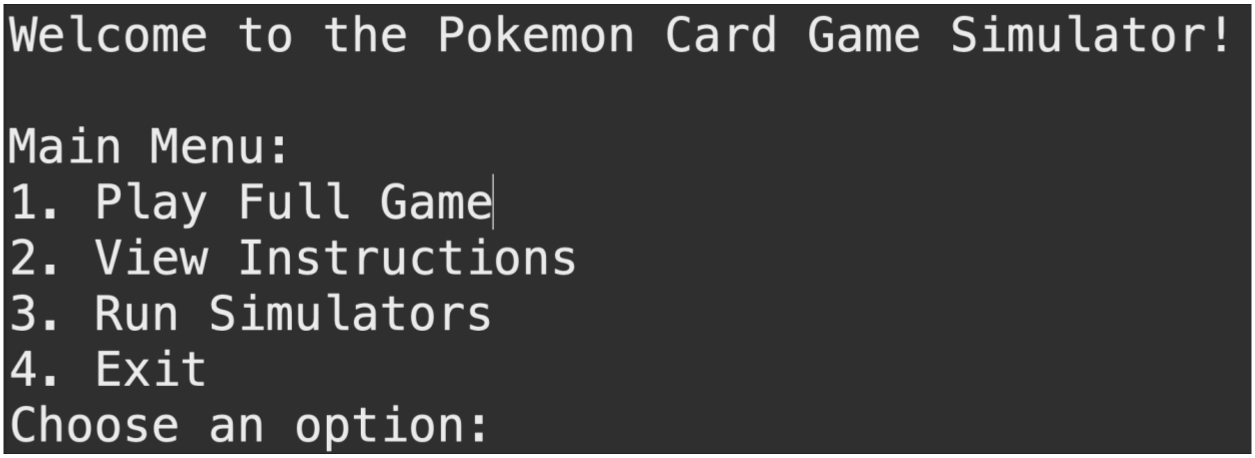
The primary purpose of the Pokemon Simulator is to serve as an educational, analytical, and entertaining tool for players of the Pokemon TCG. By simulating multiple game sessions and collecting statistical data on key gameplay metrics such as Mulligan rates and initial hand openings, the application allows players to:

- Gain a deeper understanding of the probability distributions within the game.
- Evaluate the consistency and effectiveness of different deck compositions and card strategies.
- Refine their deck-building approaches and tactical decision-making processes to improve their overall performance in real gameplay scenarios.

3. Features

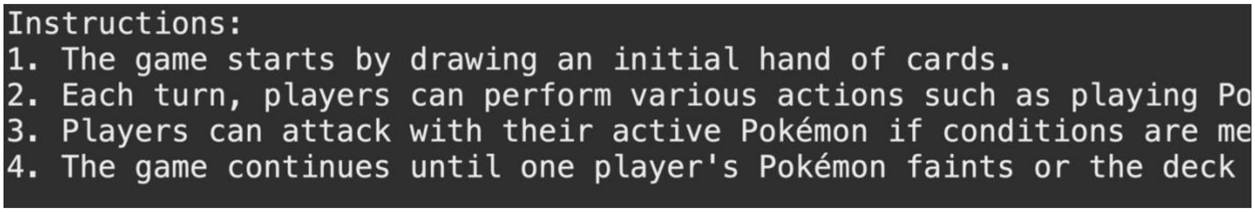
3.1 Menu

When the game starts, the player is presented with a menu in the console output:



```
Welcome to the Pokemon Card Game Simulator!

Main Menu:
1. Play Full Game
2. View Instructions
3. Run Simulators
4. Exit
Choose an option:
```

Here, players have different options to choose from. For example, a beginner, who may not be too familiar with the game, can choose option 3, “view instructions,” which would look like this:

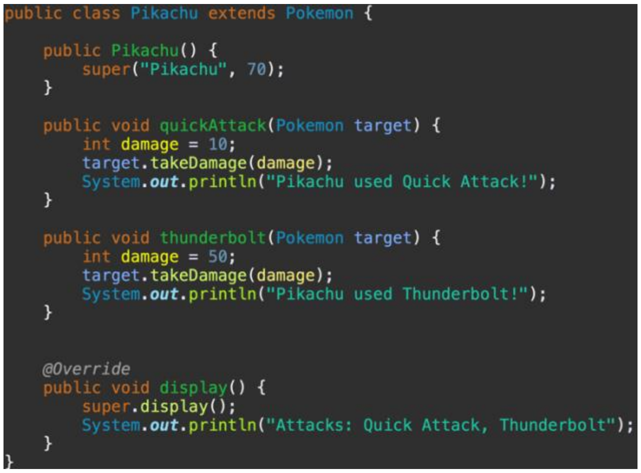


```
Instructions:
1. The game starts by drawing an initial hand of cards.
2. Each turn, players can perform various actions such as playing Po
3. Players can attack with their active Pokémon if conditions are me
4. The game continues until one player's Pokémon faints or the deck
```

3.2 Pokemon classes

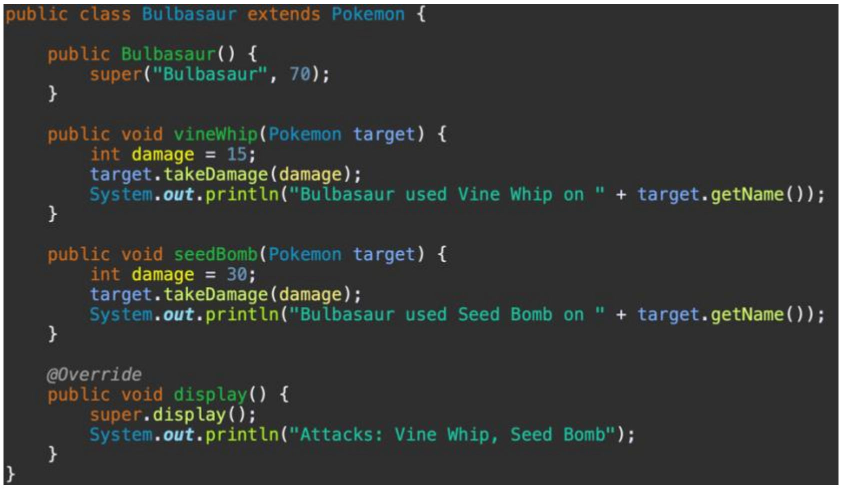
I chose three different specific Pokemons for this program: Pikachu, Bulbasaur, and Charmander. The rest of the cards added were generic Pokemons to serve as placeholders in the deck.

Pokemon class:



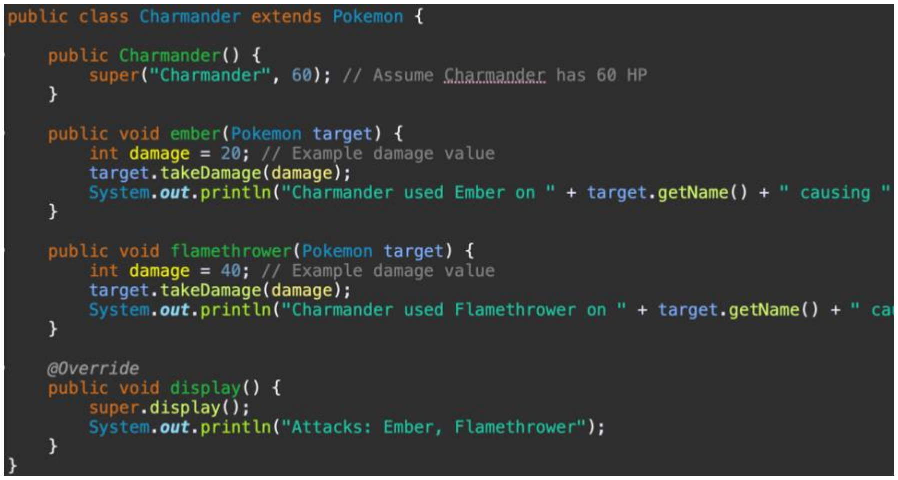
```java
public class Pikachu extends Pokemon {

    public Pikachu() {
        super("Pikachu", 70);
    }

    public void quickAttack(Pokemon target) {
        int damage = 10;
        target.takeDamage(damage);
        System.out.println("Pikachu used Quick Attack!");
    }

    public void thunderbolt(Pokemon target) {
        int damage = 50;
        target.takeDamage(damage);
        System.out.println("Pikachu used Thunderbolt!");
    }


    @Override
    public void display() {
        super.display();
        System.out.println("Attacks: Quick Attack, Thunderbolt");
    }
}
```

Bulbasaur class:



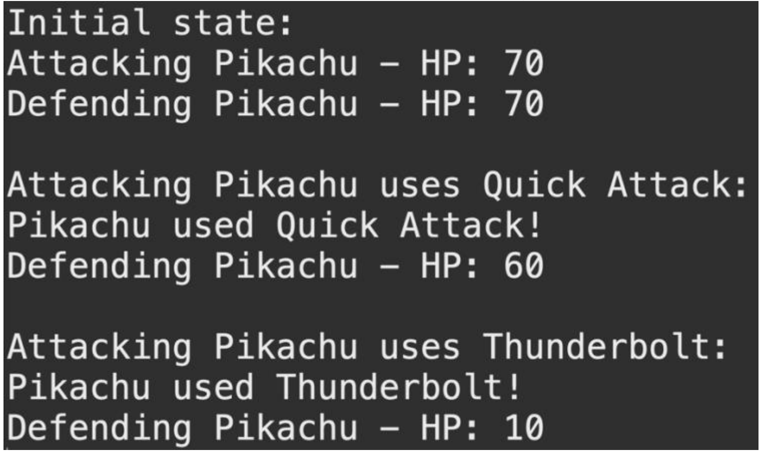
```java
public class Bulbasaur extends Pokemon {

    public Bulbasaur() {
        super("Bulbasaur", 70);
    }

    public void vineWhip(Pokemon target) {
        int damage = 15;
        target.takeDamage(damage);
        System.out.println("Bulbasaur used Vine Whip on " + target.getName());
    }

    public void seedBomb(Pokemon target) {
        int damage = 30;
        target.takeDamage(damage);
        System.out.println("Bulbasaur used Seed Bomb on " + target.getName());
    }

    @Override
    public void display() {
        super.display();
        System.out.println("Attacks: Vine Whip, Seed Bomb");
    }
}
```

Charmander:



```java
public class Charmander extends Pokemon {

    public Charmander() {
        super("Charmander", 60); // Assume Charmander has 60 HP
    }

    public void ember(Pokemon target) {
        int damage = 20; // Example damage value
        target.takeDamage(damage);
        System.out.println("Charmander used Ember on " + target.getName() + " causing "
    }

    public void flamethrower(Pokemon target) {
        int damage = 40; // Example damage value
        target.takeDamage(damage);
        System.out.println("Charmander used Flamethrower on " + target.getName() + " ca
    }

    @Override
    public void display() {
        super.display();
        System.out.println("Attacks: Ember, Flamethrower");
    }
}
```

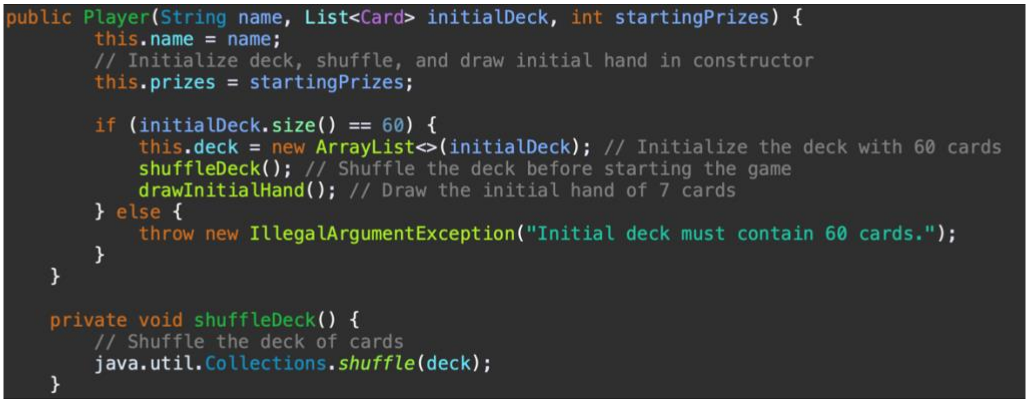
In these classes, these three Pokemons have specific attacks that only they can use. Also, each Pokemon starts off with a number of HP, which decreases based on the opponent’s attack.



```
Initial state:
Attacking Pikachu - HP: 70
Defending Pikachu - HP: 70

Attacking Pikachu uses Quick Attack:
Pikachu used Quick Attack!
Defending Pikachu - HP: 60

Attacking Pikachu uses Thunderbolt:
Pikachu used Thunderbolt!
Defending Pikachu - HP: 10
```

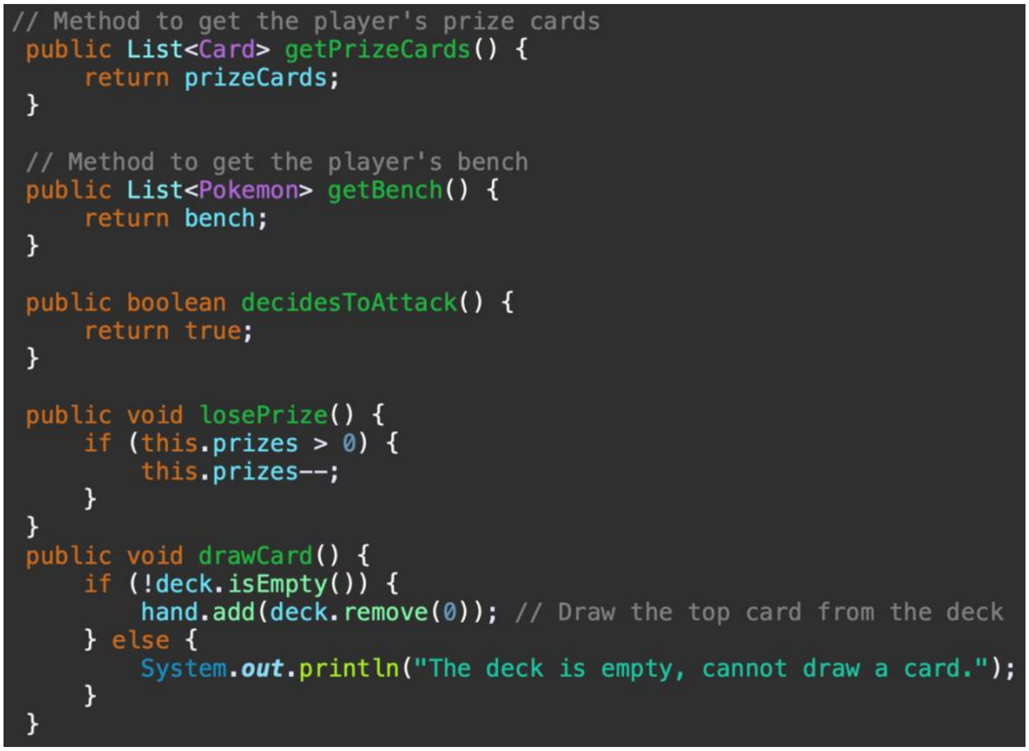
3.3 Deck Initialization

A Pokemon card game deck consists of: 20 Pokemon, 20 Energy, and 20 Trainer cards.



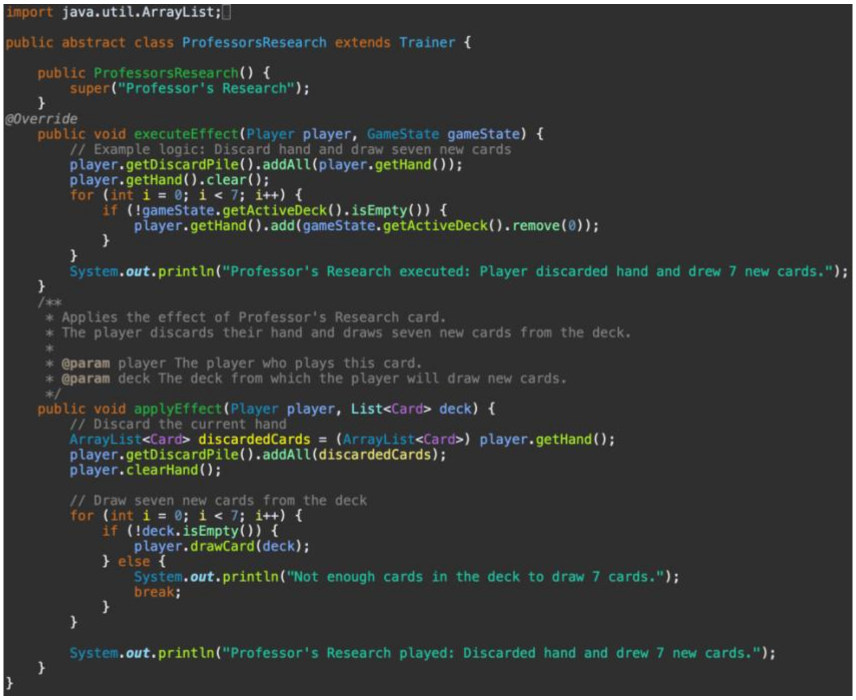
```java
public Player(String name, List<Card> initialDeck, int startingPrizes) {
        this.name = name;
        // Initialize deck, shuffle, and draw initial hand in constructor
        this.prizes = startingPrizes;

        if (initialDeck.size() == 60) {
            this.deck = new ArrayList<>(initialDeck); // Initialize the deck with 60 cards
            shuffleDeck(); // Shuffle the deck before starting the game
            drawInitialHand(); // Draw the initial hand of 7 cards
        } else {
            throw new IllegalArgumentException("Initial deck must contain 60 cards.");
        }
    }

    private void shuffleDeck() {
        // Shuffle the deck of cards
        java.util.Collections.shuffle(deck);
    }
```

If the deck is empty, no card will be drawn. Every player’s deck is shuffled and there’s also a discard pile that is included.

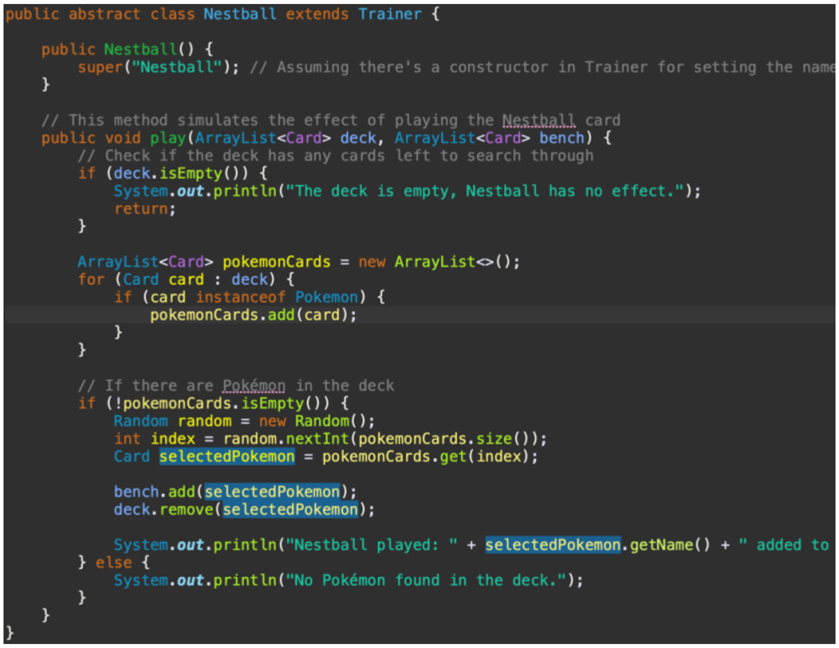


```
// Method to get the player's prize cards
 public List<Card> getPrizeCards() {
     return prizeCards;
 }

 // Method to get the player's bench
 public List<Pokemon> getBench() {
     return bench;
 }

 public boolean decidesToAttack() {
     return true;
 }

 public void losePrize() {
     if (this.prizes > 0) {
         this.prizes--;
     }
 }
 public void drawCard() {
     if (!deck.isEmpty()) {
         hand.add(deck.remove(0)); // Draw the top card from the deck
     } else {
         System.out.println("The deck is empty, cannot draw a card.");
     }
 }
```

There are a few trainer cards that have been implemented into the program, such as Nestball, Professor’s Research, etc., along with generic trainer cards. The trainer cards each have their own specialty. For example, Professor’s Research allows for a player to discard their hand and draw seven new cards from the deck.



```java
import java.util.ArrayList;

public abstract class ProfessorsResearch extends Trainer {

    public ProfessorsResearch() {
        super("Professor's Research");
    }
@Override
    public void executeEffect(Player player, GameState gameState) {
        // Example logic: Discard hand and draw seven new cards
        player.getDiscardPile().addAll(player.getHand());
        player.getHand().clear();
        for (int i = 0; i < 7; i++) {
            if (!gameState.getActiveDeck().isEmpty()) {
                player.getHand().add(gameState.getActiveDeck().remove(0));
            }
        }
        System.out.println("Professor's Research executed: Player discarded hand and drew 7 new cards.");
    }
    /**
     * Applies the effect of Professor's Research card.
     * The player discards their hand and draws seven new cards from the deck.
     *
     * @param player The player who plays this card.
     * @param deck The deck from which the player will draw new cards.
     */
    public void applyEffect(Player player, List<Card> deck) {
        // Discard the current hand
        ArrayList<Card> discardedCards = (ArrayList<Card>) player.getHand();
        player.getDiscardPile().addAll(discardedCards);
        player.clearHand();

        // Draw seven new cards from the deck
        for (int i = 0; i < 7; i++) {
            if (!deck.isEmpty()) {
                player.drawCard(deck);
            } else {
                System.out.println("Not enough cards in the deck to draw 7 cards.");
                break;
            }
        }

        System.out.println("Professor's Research played: Discarded hand and drew 7 new cards.");
    }
}
```



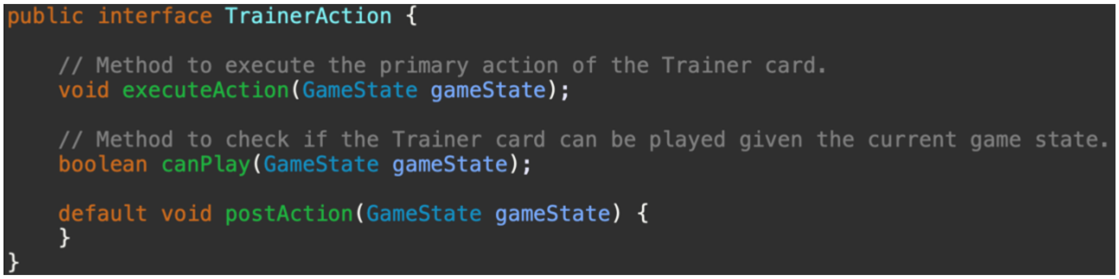
```java
public abstract class Nestball extends Trainer {

    public Nestball() {
        super("Nestball"); // Assuming there's a constructor in Trainer for setting the name
    }

    // This method simulates the effect of playing the Nestball card
    public void play(ArrayList<Card> deck, ArrayList<Card> bench) {
        // Check if the deck has any cards left to search through
        if (deck.isEmpty()) {
            System.out.println("The deck is empty, Nestball has no effect.");
            return;
        }

        ArrayList<Card> pokemonCards = new ArrayList<>();
        for (Card card : deck) {
            if (card instanceof Pokemon) {
                pokemonCards.add(card);
            }
        }

        // If there are Pokémon in the deck
        if (!pokemonCards.isEmpty()) {
            Random random = new Random();
            int index = random.nextInt(pokemonCards.size());
            Card selectedPokemon = pokemonCards.get(index);

            bench.add(selectedPokemon);
            deck.remove(selectedPokemon);

            System.out.println("Nestball played: " + selectedPokemon.getName() + " added to
        } else {
            System.out.println("No Pokémon found in the deck.");
        }
    }
}
```

The prize pile Starts at 6, and one is put in hand each Pokemon knockout. When there are zero remaining prizes, the player wins.

3.4 Initial Hand Drawing

- Upon deck initialization, the application simulates the drawing of an initial hand consisting of 7 cards for each game session.

- Users can visualize and analyze the composition of their starting hands, including the presence of playable Pokemon cards, Energy cards, and Trainer cards.



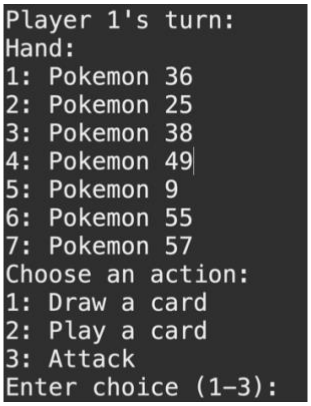
```
public interface TrainerAction {

    // Method to execute the primary action of the Trainer card.
    void executeAction(GameState gameState);

    // Method to check if the Trainer card can be played given the current game state.
    boolean canPlay(GameState gameState);

    default void postAction(GameState gameState) {
    }
}
```

3.5 Turn Simulation

- The Pokemon Simulator simulates a series of turns for each game session, allowing users to observe how gameplay unfolds over multiple rounds.
- Users can play Pokemon cards, attach energy cards to their active Pokemon, and utilize Trainer cards based on simulated gameplay outcomes.

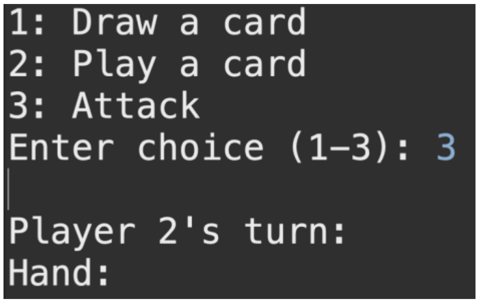
3.6 Gameplay

When a player chooses to start, the game begins with a hand of 7 cards for player one.



```
Player 1's turn:
Hand:
1: Pokemon 36
2: Pokemon 25
3: Pokemon 38
4: Pokemon 49
5: Pokemon 9
6: Pokemon 55
7: Pokemon 57
Choose an action:
1: Draw a card
2: Play a card
3: Attack
Enter choice (1-3):
```

Here, the player can choose to either draw or play a card, which can be done multiple times. But, once player 1 chooses to attack, their turn is over. The player can choose trainer cards as well.

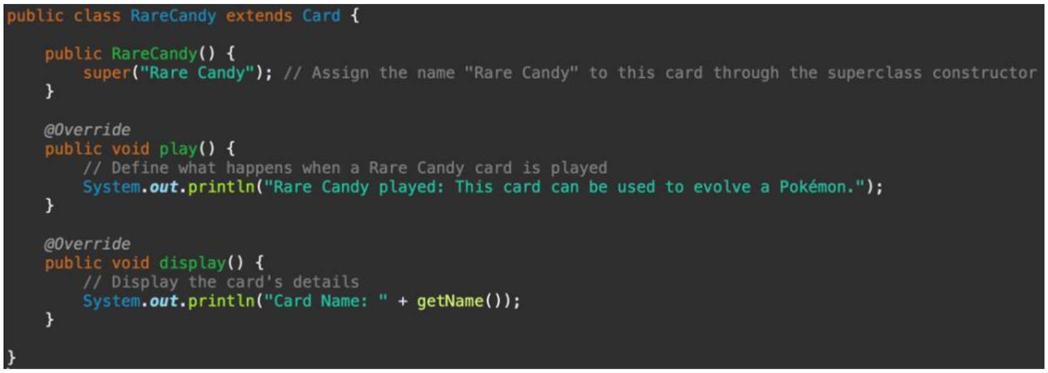


```
1: Draw a card
2: Play a card
3: Attack
Enter choice (1-3): 3

Player 2's turn:
Hand:
```

Once I chose to attack with player 1, the turn was over and now it's the second player's turn. The game continue as long as either player has prize cards remaining.

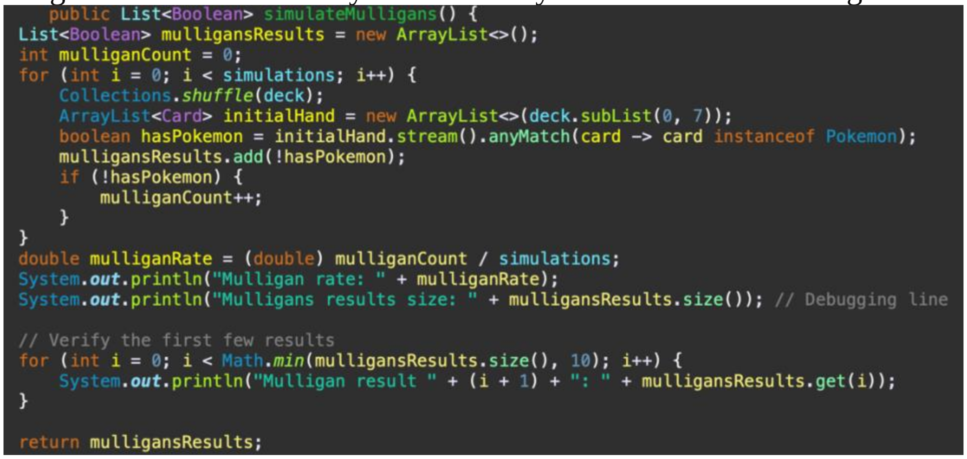
3.7 Mulligan Simulation

One of the key features of the application is the simulation of Mulligan rates, which consists of a a charizard deck finding a rare candy to have a chance at winning.

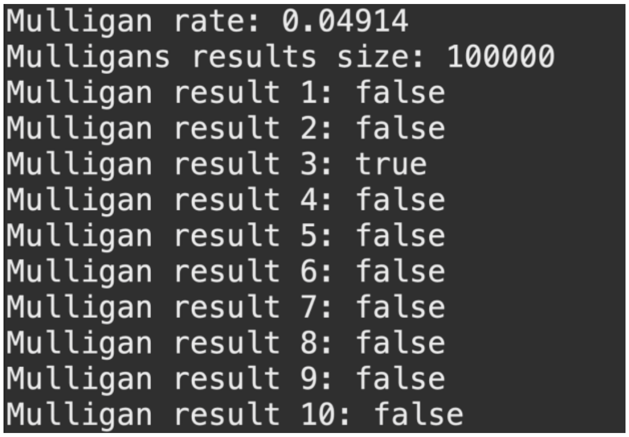


```java
public class RareCandy extends Card {

    public RareCandy() {
        super("Rare Candy"); // Assign the name "Rare Candy" to this card through the superclass constructor
    }

    @Override
    public void play() {
        // Define what happens when a Rare Candy card is played
        System.out.println("Rare Candy played: This card can be used to evolve a Pokémon.");
    }

    @Override
    public void display() {
        // Display the card's details
        System.out.println("Card Name: " + getName());
    }

}
```

By conducting multiple simulations, the application calculates Mulligan rates and provides insights into the consistency and reliability of different deck configurations.



```java
    public List<Boolean> simulateMulligans() {
List<Boolean> mulligansResults = new ArrayList<>();
int mulliganCount = 0;
for (int i = 0; i < simulations; i++) {
    Collections.shuffle(deck);
    ArrayList<Card> initialHand = new ArrayList<>(deck.subList(0, 7));
    boolean hasPokemon = initialHand.stream().anyMatch(card -> card instanceof Pokemon);
    mulligansResults.add(!hasPokemon);
    if (!hasPokemon) {
        mulliganCount++;
    }
}
double mulliganRate = (double) mulliganCount / simulations;
System.out.println("Mulligan rate: " + mulliganRate);
System.out.println("Mulligans results size: " + mulligansResults.size()); // Debugging line

// Verify the first few results
for (int i = 0; i < Math.min(mulligansResults.size(), 10); i++) {
    System.out.println("Mulligan result " + (i + 1) + ": " + mulligansResults.get(i));
}

return mulligansResults;
```

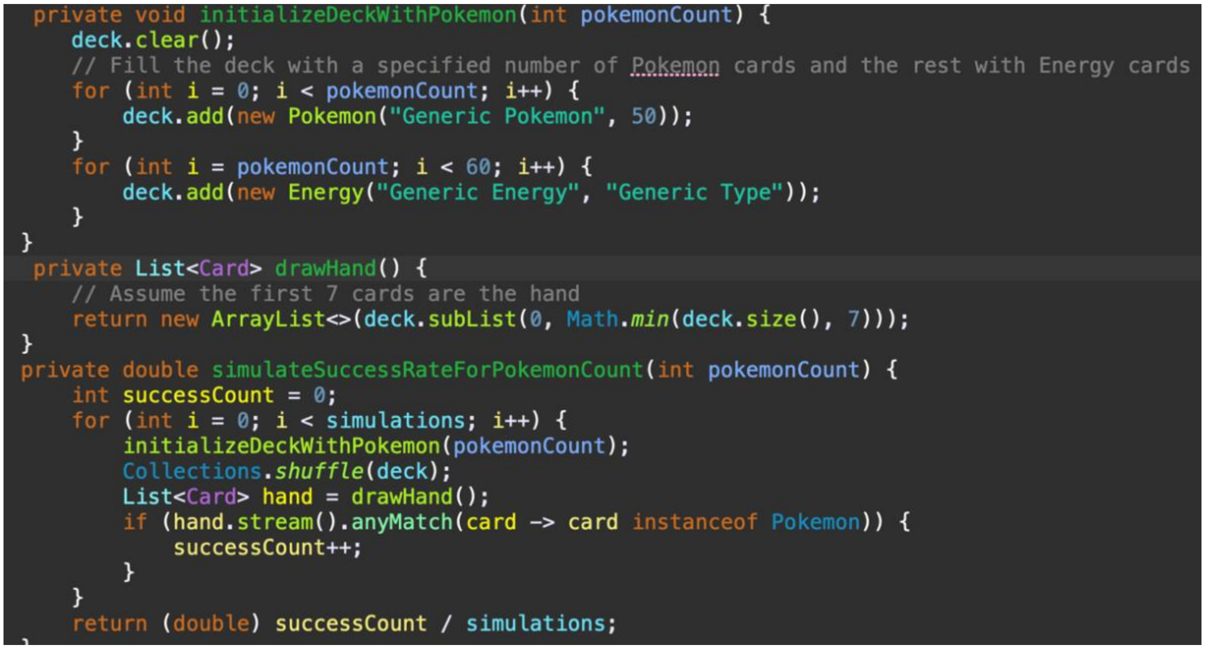


```
Mulligan rate: 0.04914
Mulligans results size: 100000
Mulligan result 1: false
Mulligan result 2: false
Mulligan result 3: true
Mulligan result 4: false
Mulligan result 5: false
Mulligan result 6: false
Mulligan result 7: false
Mulligan result 8: false
Mulligan result 9: false
Mulligan result 10: false
```

The results of the simulation were taken and graphed into an Excel file.

3.8 Opening Hands Simulation

The purpose of this monte carlo simulation is to experiment with drawing hands. The initial deck is going to be populated with 1 Pokemon and 59 energy cards. Then, the number of Pokemon cards will increase to 2, and the energy cards will decrease to 58. The goal is to increase the Pokemon cards by 1 in each run while simultaneously decreasing the number of energy cards.

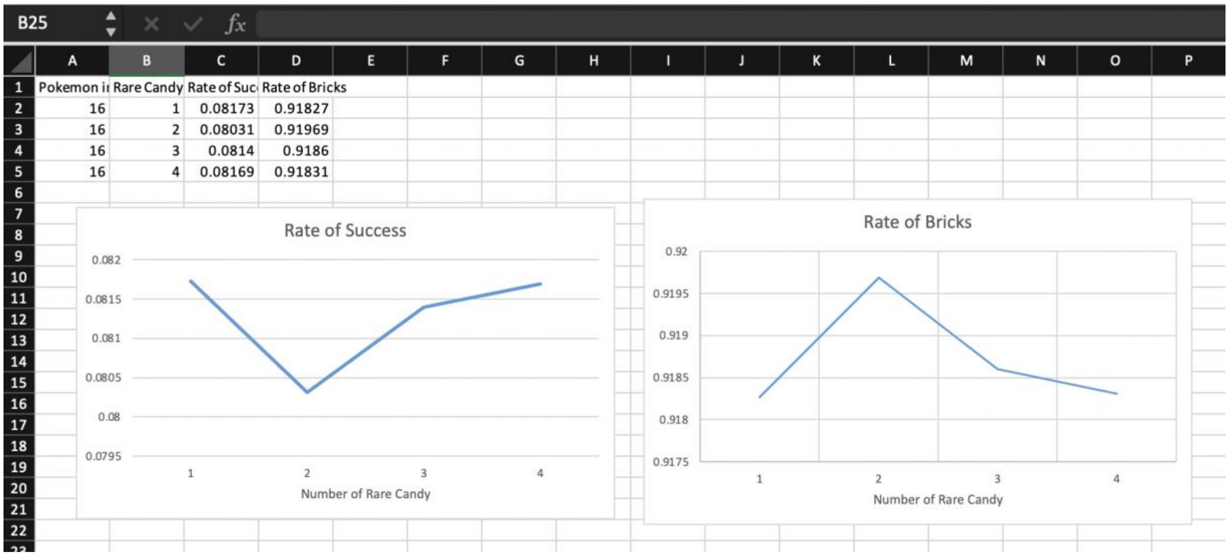


```
 private void initializeDeckWithPokemon(int pokemonCount) {
    deck.clear();
    // Fill the deck with a specified number of Pokemon cards and the rest with Energy cards
    for (int i = 0; i < pokemonCount; i++) {
        deck.add(new Pokemon("Generic Pokemon", 50));
    }
    for (int i = pokemonCount; i < 60; i++) {
        deck.add(new Energy("Generic Energy", "Generic Type"));
    }
}
 private List<Card> drawHand() {
    // Assume the first 7 cards are the hand
    return new ArrayList<>(deck.subList(0, Math.min(deck.size(), 7)));
}
private double simulateSuccessRateForPokemonCount(int pokemonCount) {
    int successCount = 0;
    for (int i = 0; i < simulations; i++) {
        initializeDeckWithPokemon(pokemonCount);
        Collections.shuffle(deck);
        List<Card> hand = drawHand();
        if (hand.stream().anyMatch(card -> card instanceof Pokemon)) {
            successCount++;
        }
    }
    return (double) successCount / simulations;
```

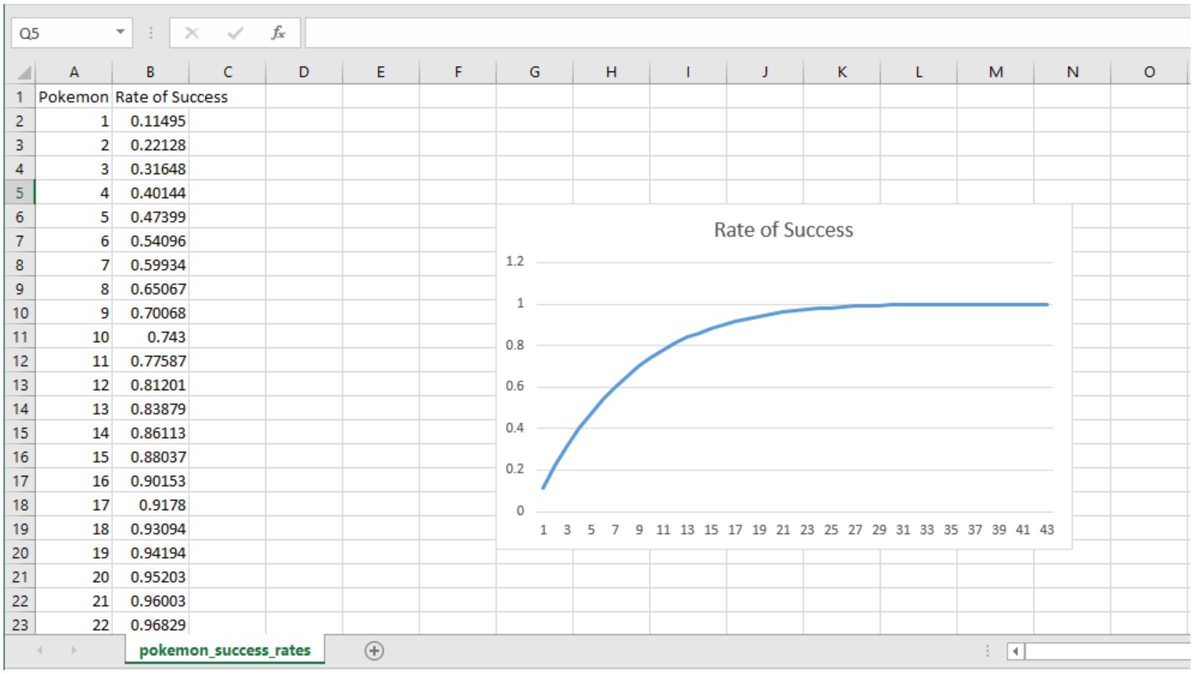
Plotting the results in Excel, I can use the program determine the optimal ratios of pokemon to energy to get the probability of always drawing a pokemon in the opening hand.

3.9 Chart Generation

- To facilitate data visualization and analysis, the Pokemon Simulator includes a chart generation feature that produces graphical representations of simulation results.
- Charts, generated in Excel format, illustrate statistical data such as Mulligan rates, Pokemon presence in initial hands, and other relevant gameplay metrics.



Pokemon i	Rare Candy	Rate of Suc	Rate of Bricks
16	1	0.08173	0.91827
16	2	0.08031	0.91969
16	3	0.0814	0.9186
16	4	0.08169	0.91831



Pokemon	Rate of Success
1	0.11495
2	0.22128
3	0.31648
4	0.40144
5	0.47399
6	0.54096
7	0.59934
8	0.65067
9	0.70068
10	0.743
11	0.77587
12	0.81201
13	0.83879
14	0.86113
15	0.88037
16	0.90153
17	0.9178
18	0.93094
19	0.94194
20	0.95203
21	0.96003
22	0.96829

4. Technologies Used

- **Java:** The core programming language used for application development, providing platform independence and flexibility.
- **NetBeans IDE:** The integrated development environment utilized for coding, debugging, and testing the application's source code.
- **ExcelChartGenerator:** A custom Java class employed for generating charts in Microsoft Excel format, enhancing data visualization capabilities.
- **Git:** A version control system employed for managing project source code, enabling collaboration and codebase maintenance.

5. Future Developments

- **User Interface (UI) Development:** Enhance the application with a graphical user interface (GUI) to improve usability and user interaction.
- **Custom Deck Building:** Introduce functionality for users to create custom decks with specific card compositions and test them through simulations.
- **Advanced Statistical Analysis:** Incorporate more sophisticated statistical analysis features to provide deeper insights into gameplay dynamics and strategy optimization.
- **Multiplayer Support:** Implement multiplayer functionality to simulate matches between groups of players and analyze competitive strategies and interactions.

7. Conclusion

The Pokemon Card Game Simulator serves as a valuable tool for players aiming to boost their knowledge and skills in the Pokemon trading card game. Through the utilization of simulation techniques and statistical analysis, the program enables players to make informed decisions based on data, polish their gameplay strategies, and enhance their performance in actual game situations. As the Pokemon Simulator undergoes ongoing development and improvements, it has the capacity to evolve into a crucial resource for Pokemon TCG enthusiasts across various skill levels.