**Developer:** Rayan Minhaj

**TABLE OF CONTENTS**

**1. INTRODUCTION----------------------------------------------- 2**

**2. BACKGROUND------------------------------------------------- 2**

**3. DESIGN & IMPLEMENTATION---------------------------- 3-4**

**4. RESULTS / SAMPLE OUTPUTS----------------------------- 5-7**

**5. CONCLUSION------------------------------------------------- 7-8**

**6. REFERENCES----------------------------------------------------- 8**

1. **INTRODUCTION:-**

This project implements a simplified version of the classic casino game **Blackjack (21)**, with a focus on applying core data structure concepts, specifically **stacks** and **queues**. The game allows 1 to 7 players to play against the dealer using 1 to 8 decks.

The simulation handles dealing, shuffling, scoring, and statistical analysis of outcomes. Rather than allowing interactive choices like “hit” or “stand,” the game deals two cards to each player and the dealer, then determines the winner automatically based on Blackjack rules.

The program is written in **Python** and demonstrates:

* Object-oriented programming
* Stack and queue manipulation
* Randomized shuffling algorithms
* Statistical tracking and analysis

**2. BACKGROUND:-**

* **2.1 Blackjack Game Overview**

Blackjack is a card game where players aim to get a hand value as close to 21 as possible without exceeding it. Number cards are worth their face value, face cards (J, Q, K) are worth 10, and aces are worth either 1 or 11. In this simplified version, only two cards are dealt per player and to the dealer; the hand with the highest value wins.

* **2.2 Data Structure Principles**
* **Stack (LIFO):** Used to model the deck of cards. Cards are pushed into the stack during setup and popped during transfer to the shoe.
* **Queue (FIFO):** Used to simulate the dealer's shoe. Cards are enqueued after shuffling and dequeued during the dealing phase.

Python lists are used internally to implement both structures with custom methods such as push(), pop(), enqueue(), and dequeue().

**3. DESIGN & IMPLEMENTATION:-**

* **3.1 Class Structure**

The program is modular and built using the following custom classes:

* **Card:** Represents a playing card.
* **Stack:** Implements the deck as a LIFO structure.
* **Queue:** Models the dealer's shoe as a FIFO structure.
* **BlackjackGame:** Manages game logic, simulations, and statistics.
* **3.2 Deck Initialization**

The initialize\_deck() method creates a stack of cards based on the chosen number of decks. Each card is represented by a Card object, and the complete deck is pushed onto the stack.

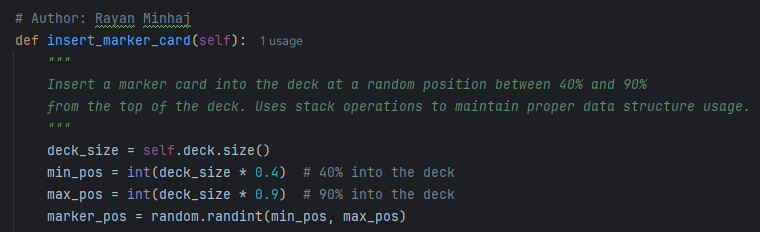
* **3.3 Shuffling and Marker Card**

Cards are shuffled using the Fisher-Yates algorithm. A marker card is then inserted between 40% and 90% of the deck to signal when a reshuffle is needed. This mimics real-world casino shoe behavior.

* **3.4 Game Flow**

The game proceeds as follows:

1. Cards are moved from the deck (stack) to the shoe (queue).
2. Each player and the dealer receive two cards.
3. Hand values are calculated with proper ace handling.
4. Results (win/loss/push) are determined.
5. Statistics are updated.
6. Marker card triggers reshuffle when hit.

**Code Snippet for Marker Insertion**

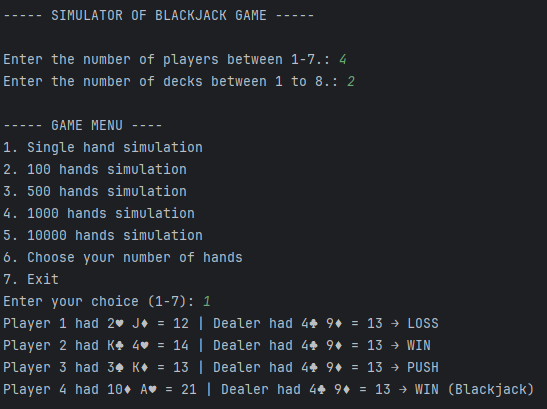
* **3.5 Statistics**

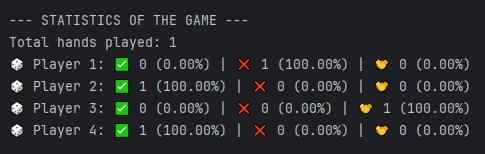
Statistics are calculated and displayed after each hand or simulation:

* Wins, losses, pushes per player
* Win/loss/push percentages

**4. RESULTS/SAMPLE OUTPUTS:-**

* **4.1 Single Hand Simulation:**





Upon selecting **"1. Single hand simulation"** from the menu, the following round was simulated:

**Breakdown of the result:**

* **Player 1**: Total value = 12 → **LOSS** (Dealer had 13)
* **Player 2**: Total value = 14 → **WIN** (Dealer had 13)
* **Player 3**: Total value = 13 → **PUSH** (tie with Dealer)
* **Player 4**: Total value = 21 → **WIN (Blackjack)**

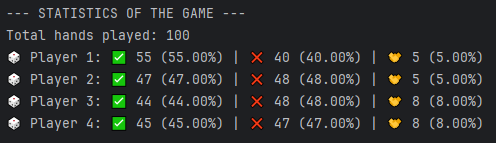
The program evaluates the hands, applies simplified Blackjack rules, and determines outcomes accordingly.

* **4.2 Continuing Further:**

--- Starting simulation of 100 hands ---

(Prints the simulation. Output is too long to print)

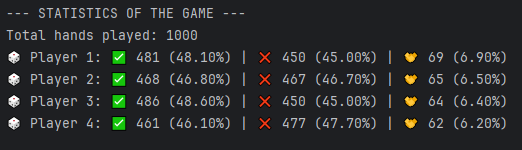
----- Simulation of 100 hands completed -----



--- Starting simulation of 1000 hands ---

(Prints the simulation. Output is too long to print)

----- Simulation of 1000 hands completed -----



After running a larger simulation with **100 hands and** **1,000 hands respectively**, the results provided a clearer statistical picture:

* Win percentages for players ranged from **46.1% to 48.6%**
* Loss rates ranged from **45.0% to 47.7%**
* Pushes (ties) occurred in about **6.2% to 6.9%** of hands

As expected, with a higher number of simulations, the data began to stabilize and reflect consistent Blackjack dynamics. The win/loss/push ratios were fairly balanced across all four players, indicating fair and randomized game mechanics. The push rate remained steady at around **6–7%**, which is normal in Blackjack.

The results validated the functionality of the simulation and the correctness of the underlying algorithms and data structures.

* 1. **CONCLUSION:-**

This project gave me the opportunity to implement a simplified Blackjack game by applying the concepts of stack and queue data structures in a real-world simulation. I used a stack to manage the deck of cards (LIFO), and a queue to model the dealer’s shoe (FIFO), allowing cards to be dealt in the proper order. By structuring the game in a modular and class-based approach, I was able to maintain clarity and organization in the code while fulfilling the project requirements.

Throughout this project, I learned:

1. How to build custom Stack and Queue classes using lists and enforce their intended usage
2. How to shuffle cards properly using the Fisher-Yates algorithm
3. How to insert and detect a marker card to signal when a reshuffle should occur
4. How to calculate hand values, especially with edge cases involving Aces
5. How to simulate and analyze large amounts of game data using statistical output

What makes me proud is that I was able to run full-scale simulations (100, 1,000 hands), which revealed a steady pattern in win/loss/push percentages — a strong indicator that the logic and randomness are working properly. The Blackjack game consistently produced results that matched expected behavior, even across thousands of hands.

\*Note: I would have improved the program even further by using features like graphical interface if I had more time because personally it’s really hard for me to scroll through the simulation.

Overall, this project helped reinforce the importance of data structures in game logic and gave me hands-on experience with how abstract concepts can be translated into interactive programs.

* 1. **REFERENCES:-**

I used the following sources to help me in making this project:

* [How to play Blackjack](https://www.blackjackapprenticeship.com/how-to-play-blackjack/)
* [Stack and Queue](https://www.geeksforgeeks.org/stack-and-queues-in-python/)
* [Yates Shuffle](https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle)