Roulette Strategy Simulator: Project Overview

By: Jackson Gamel

Project Title: Roulette Strategy Simulator

Objective:

As someone who likes to gamble occasionally and is fascinated with the probability and the various mathematical approaches to try and overcome the house edge, creating a roulette simulator was an idea that popped up in my head when we first discussed the final project in class. The goal of this project is to simulate and compare several well-known roulette betting strategies, including Martingale, Flat Betting, Reverse Martingale, D'Alembert, and Fibonacci, in order to analyze their outcomes over multiple spins and trials. The simulator also allows users to visualize their performance through graphs and save the results to CSV files for further analysis of their bankroll.

Strategies Implemented:

The simulator implements five classic betting strategies that are frequently used by roulette players in the real world. Each strategy has its own approach to managing bets, to maximize profits or minimize losses. Here's an in-depth look at each strategy:

1. Martingale Strategy

- **How it Works**:
  The Martingale strategy is one of the most well-known and risky betting strategies in roulette. It involves doubling the bet after each loss. The idea is that after a loss, the next win will recover all previous losses and yield a profit equal to the original bet. If you have an infinite bankroll, this strategy will theoretically always break even due to the exponential increase in bet size until a win occurs.

- **Implementation in the Simulator**:
  In the simulator, the Martingale strategy starts with a base bet, and each time the player loses, the bet is doubled. If the player wins, they return to the original bet.

- **Key Risk**:
  The major risk of Martingale is that a long streak of losses can lead to extremely large bets, which may exceed the player's bankroll or table limits.

2. Flat Betting

- **How it Works**:
  In flat betting, the player bets the same amount on every spin regardless of wins or losses. This strategy is the simplest and the most conservative approach. Flat betting in my simulator gives the player about a 47% chance. This represents betting on a color like red or black, high or low numbers, or odd/even.

- **Implementation in the Simulator**:
  The simulator uses a fixed starting bet for every spin. There is no change in the bet size, regardless of the results of the previous spins.

- **Key Risk**:
  Flat betting can be less risky but also less profitable in the long run compared to other strategies that adjust bets based on previous results.

3. Reverse Martingale Strategy

- **How it Works**:
  Reverse Martingale is the opposite of the traditional Martingale system. In this strategy, the player doubles their bet after a win rather than a loss. The idea is to maximize winnings during a winning streak while minimizing risk during a losing streak.

- **Implementation in the Simulator**:
  The simulator applies the Reverse Martingale strategy by doubling the bet after each win. After a loss, the bet returns to the starting value.

- **Key Risk**:
  The main risk is that a win streak must occur early enough in the betting sequence to recover any prior losses.

4. D'Alembert Strategy

- **How it Works**:
  The D'Alembert system is a more conservative betting strategy where the player

increases their bet by one unit after a loss and decreases it by one unit after a win. It is designed to provide a balanced approach between risk and reward.

- **Implementation in the Simulator**:
  In the simulator, the player starts with a base bet, and after each loss, the bet is increased by one unit. After each win, the bet is reduced by one unit.

- **Key Risk**:
  While less risky than Martingale, the D'Alembert system does not guarantee recovery from long losing streaks, and it can still lead to significant losses if the player experiences multiple consecutive losses.

5. Fibonacci Strategy

- **How it Works**:
  The Fibonacci strategy is based on the Fibonacci sequence, where each number is the sum of the two preceding ones. In roulette, a player increases their bet according to the Fibonacci sequence after each loss. After a win, they move back two steps in the sequence.

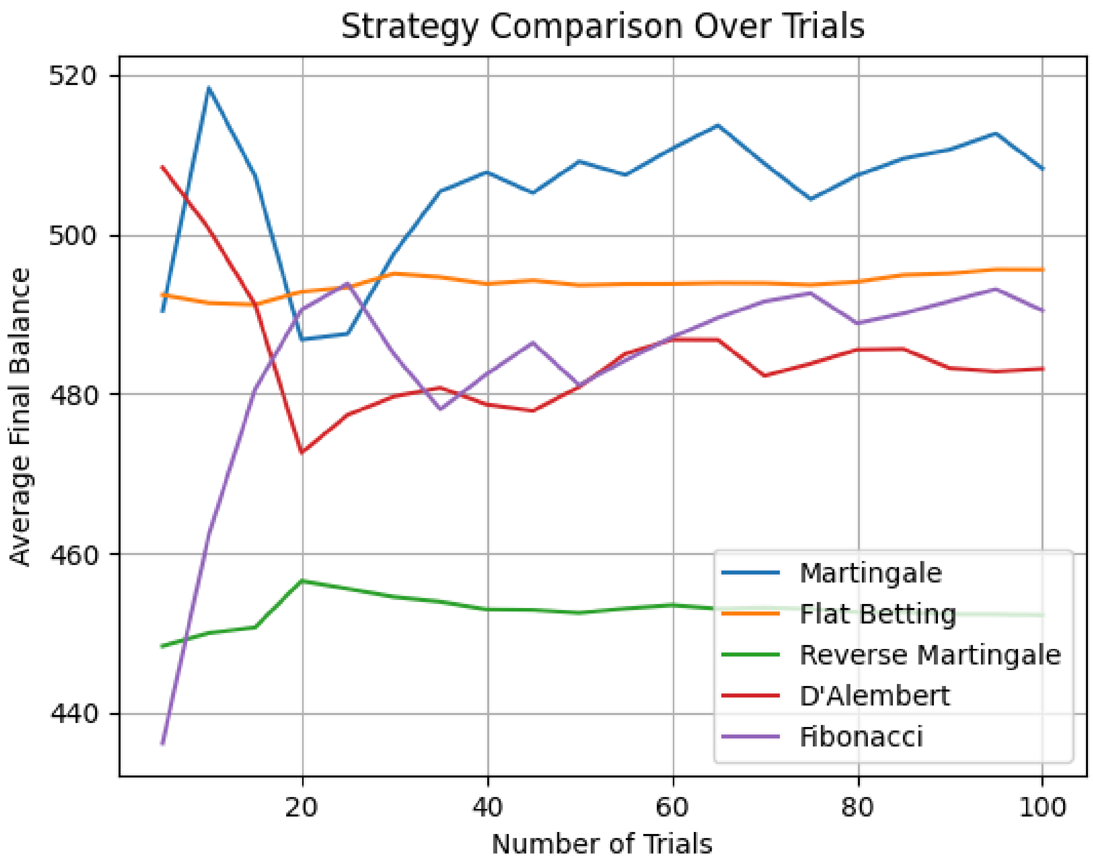
- **Implementation in the Simulator**:
  The simulator applies the Fibonacci sequence for bets after each loss. The player moves forward in the sequence after a loss and back two steps after a win.

- **Key Risk**:
  Like Martingale, the Fibonacci system can lead to large bets during a losing streak, although it is less aggressive than the Martingale strategy.

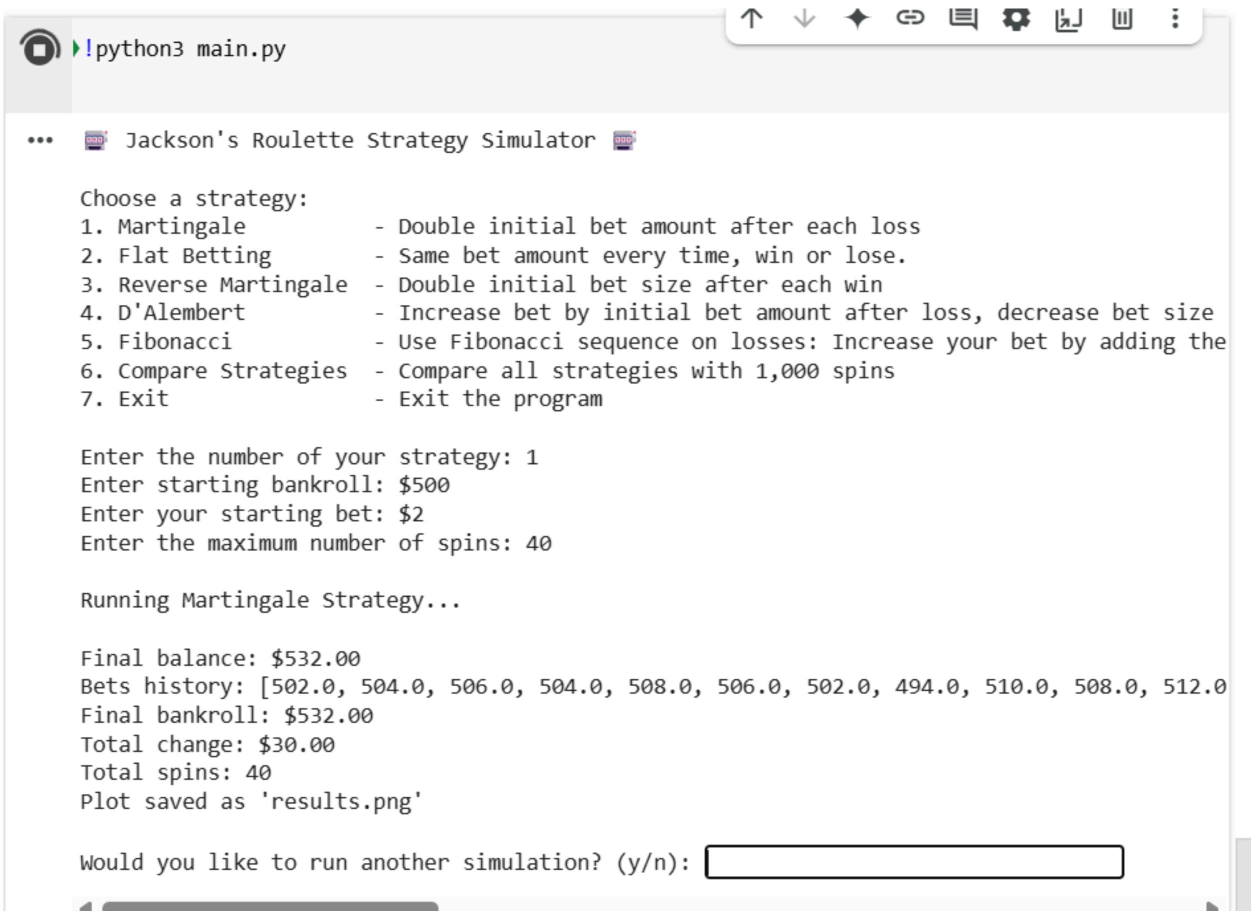
The Compare Strategies function was designed to run multiple betting strategies simultaneously and visually compare their outcomes over a specified number of spins. By using this feature, users can easily observe how each strategy performs in terms of bankroll growth or decline, providing a clear side-by-side analysis of their effectiveness. The function works by running each strategy with the same initial parameters (such as base bet and number of spins), and after all the strategies have completed their simulations, the results are displayed on a graph. This graphical representation makes it easier for users to identify trends, such as which strategies tend to perform better over long runs or which may be riskier in volatile conditions. The comparison feature enhances the simulator by allowing users to make more informed decisions about which betting strategies to use based on their preferences and risk tolerance.

The Auto Feature allows users to automate the process of running the strategies multiple times for a larger dataset, which helps in gathering more meaningful insights. By using a simple command like python3 main.py auto 100, users can set the simulator to run 100 spins for each strategy multiple times (for example, 100 trials). This feature is particularly useful when trying to analyze how each strategy performs over a broader range of trials, as it minimizes human intervention and speeds up the process of collecting large amounts of data. Additionally, the auto feature generates a CSV file with the results of each run, which can be further analyzed or visualized. This makes the simulator more versatile and suitable for users who want to dive deeper into statistical analysis or compare the long-term viability of different strategies.

Here is an example graph that was produced after using the auto feature :



Here is an example of the output for a martingale simulation



```
!python3 main.py

🎰 Jackson's Roulette Strategy Simulator 🎰

Choose a strategy:
1. Martingale          - Double initial bet amount after each loss
2. Flat Betting        - Same bet amount every time, win or lose.
3. Reverse Martingale  - Double initial bet size after each win
4. D'Alembert          - Increase bet by initial bet amount after loss, decrease bet size
5. Fibonacci           - Use Fibonacci sequence on losses: Increase your bet by adding the
6. Compare Strategies  - Compare all strategies with 1,000 spins
7. Exit                - Exit the program

Enter the number of your strategy: 1
Enter starting bankroll: $500
Enter your starting bet: $2
Enter the maximum number of spins: 40

Running Martingale Strategy...

Final balance: $532.00
Bets history: [502.0, 504.0, 506.0, 504.0, 508.0, 506.0, 502.0, 494.0, 510.0, 508.0, 512.0
Final bankroll: $532.00
Total change: $30.00
Total spins: 40
Plot saved as 'results.png'

Would you like to run another simulation? (y/n):
```

A graph can then be viewed if the user would like to visiualize their results better.

After running multiple trials of this, I have come to the conclusion that Martingale gives you the best chance at profiting, but it is also very volatile.

Development Process

The development process for the Roulette Strategy Simulator involved several key steps:

1. Research and Selection of Strategies:

The first step was to research the most commonly used betting strategies in roulette. These strategies were selected based on their popularity and differences in betting approaches. Sources like Casino Life Magazine and other gambling strategy blogs were used to understand the theory behind each system.

2. Implementation of the Game Logic:

The Roulette class was created to simulate a roulette wheel spin, using random number generation to simulate results. This class interacts with the various betting strategies and calculates the player's bankroll after each spin.

3. Simulation and Strategy Comparison:

After implementing the basic strategies, a function was written to compare their effectiveness over a large number of spins and trials. This comparison allows for easy visualization of the cumulative bankroll over time for each strategy.

4. Visualization:

To make the results more understandable, the matplotlib library was used to create graphs that show the bankroll progression over spins. These graphs were saved as PNG images and can be used to visually compare the strategies.

5. User Interaction:

For user interaction, a simple text-based menu system was created. Users can choose between running a specific strategy, running an automated comparison, or exiting the program. Error handling was implemented to ensure that user inputs are valid.

6. Testing and Optimization:

Once the initial code was written, testing was done to ensure that each strategy was working correctly and producing the expected results. Optimization was performed on the CSV output and the graph generation to ensure the program runs efficiently.

Final Thoughts and Future Improvements

The Roulette Strategy Simulator offers a comprehensive way to compare various betting strategies and their effectiveness over time. While the results are primarily based on simulations and theoretical models, they can provide valuable insights for gamblers interested in understanding the potential outcomes of different strategies. It is always important to remember that while all casino games favor the house, games such as roulette, where a real person drops the ball, can lead to a human bias that impacts the wheel, changing the mathematically predetermined odds for certain numbers occurring.

Possible Improvements:

- **Realistic Wheel Simulations**: The current simulation assumes a perfect 50/50 win-loss ratio, while actual roulette wheels have a slight house edge. Future versions could incorporate a more realistic simulation.

- **Graphical User Interface (GUI)**: A GUI could be added to make the simulation more user-friendly and visually appealing.

- **More Strategies**: Additional strategies could be implemented, such as Labouchère or Oscar's Grind.

Conclusion

This project serves as both a practical exercise in Python programming and a deeper dive into the mechanics of roulette betting systems. Through careful planning and research, the project effectively simulates and compares five of the most popular roulette strategies, providing users with valuable insights into their effectiveness. The ability to analyze and visualize the results further enhances the learning experience for anyone interested in gambling strategies and Python development.

Works Cited

"Roulette Strategies: Let's Discover the Most Successful Ones and How They Work." *Casino Life Magazine*, https://www.casinolifemagazine.com/blog/roulette-strategies-let%E2%80%99s-di scover-most-successful-ones-and-how-they-work. Accessed 13 May 2025.

"Roulette Strategy Guide." *888 Casino*, https://www.888casino.com/blog/roulette-strategy-guide/roulette-odds#:~:text=With%2037%20numbers%20(1%2D36,European%20roulette%20for%20better%20odds! Accessed 13 May 2025.

"Roulette Strategy." *Gambling Zone*, https://www.gamblingzone.com/uk/roulette/strategy/. Accessed 13 May 2025.