PROJECT: BLACKJACK SIMULATION

By Shikan Chen 23202508

01 RESEARCH

Investigation into applying reinforcement learning to casino games, focusing on Blackjack to simulate decision-making processes and strategy development in AI models.

BLACKJACK RULES

1. Dealing Cards:
   - Each player, including the dealer, starts with two cards.
   - One of the dealer's cards is dealt face down (the "hole card") and one face up.
   - The players' cards are usually dealt face up.
2. Player Decisions:
   - Hit: Take another card from the dealer.
   - Stand: Keep your current hand and end your turn.
   - Surrender: Forfeit half your bet and end your participation for the round (if allowed).
3. Dealer Play:
   - The dealer reveals the hole card and must hit if the total is 16 or less.
   - The dealer must stand on all totals of 17 or higher.
4. Winning the Game:
   - If your total is closer to 21 than the dealer's total without going over, you win.
   - If the dealer goes over 21 ("busts"), all remaining players win.

02 ANALYSIS

Examination of standard Blackjack strategies and potential AI enhancements, employing statistical and machine learning techniques to analyze performance and optimize decision-making.

03 OBJECTIVES

Develop a robust Blackjack simulation that: {1} employs an AI trained via Q-learning, {2} adheres to traditional Blackjack rules, {3} improves AI decision-making through continuous gameplay and feedback.

Why I Chose Q-Table for Blackjack Simulation

The Q-table approach in our Blackjack AI utilizes Q-learning, a model-free reinforcement learning algorithm, which is highly adaptable to the unpredictable nature of Blackjack. This method allows the AI to learn optimal strategies directly from gameplay without needing a predefined model of the environment. It provides a straightforward, efficient way to estimate and update the utility of each action in every game state, facilitating dynamic learning and continuous strategy improvement as more games are played. This choice ensures that the AI progressively refines its decision-making skills, making our simulation both robust and scalable.

```
Blackjack-Simulation/
│
├── ai_models/              # Trained AI models and data
│   ├── ...
├── images/                 # Images and figures used or produced by the project
│   ├── ...
├── results/                # Output results and datasets
│   ├── ...
├── eval/                   # Scripts for evaluation of simulation results
│   ├── analysis_plot.py    # Script for plotting analysis results
│   └── simulate.py         # Script for running simulations
│
├── src/                    # Source files containing the main application logic
│   ├── __init__.py         # Python Package
│   ├── agent.py            # Defines the AI agent
│   ├── card.py             # Card class for handling playing cards
│   ├── deck.py             # Deck class for handling decks of cards
│   ├── game.py             # Core game logic
│   ├── gameGUI.py          # Graphical user interface for the game
│   ├── hand.py             # Hand class for handling cards in a hand
│   ├── main.py             # Entry point of the program
│   └── train.py            # Training routines for the AI
│
├── .gitignore              # Specifies intentionally untracked files to ignore
├── environment.yml         # Conda environment file
├── LICENSE                 # License details
├── literature_review.pdf   # Background literature review document
└── README.md               # Project overview and instructions
```

```python
def calculate_reward(self):
        # Basic reward system: +1 for win, 0 for draw, -1 for loss
        if self.game_over:
            if self.player_hand.value > 21:
                return - 1.0 / (self.player_hand.value - 21)  # Player busts
            elif self.dealer_hand.value > 21:
                return 1.0 / (22 - self.player_hand.value) + 1.0 / (self.dealer_hand.value - 21)  # Dealer busts
            elif self.player_hand.value > self.dealer_hand.value:
                return 1.0 / (self.player_hand.value - self.dealer_hand.value)  # Player wins
            elif self.player_hand.value < self.dealer_hand.value:
                return - 1.0 / (self.dealer_hand.value - self.player_hand.value)  # Player loses
            else:
                return 0.5 + 1.0 / (22 - self.player_hand.value)  # Draw
        else:
            return 1.0 / (22 - self.player_hand.value)
```

1. Player or Dealer Busts:

If the player's hand value exceeds 21 (busts), a negative reward is given:
This reward becomes less negative as the player's hand value approaches 21, penalizing larger oversteps more harshly.

If the dealer's hand value exceeds 21 and the player's hand value is 21 or less, a positive reward is given:
This reward becomes more positive the closer the player's hand is to 21 and the more the dealer busts by.

2. Player Wins or Loses:

If the player's hand value is greater than the dealer's and both are 21 or less, a positive reward is given:
This reward increases as the margin between the player's and dealer's hands increases.

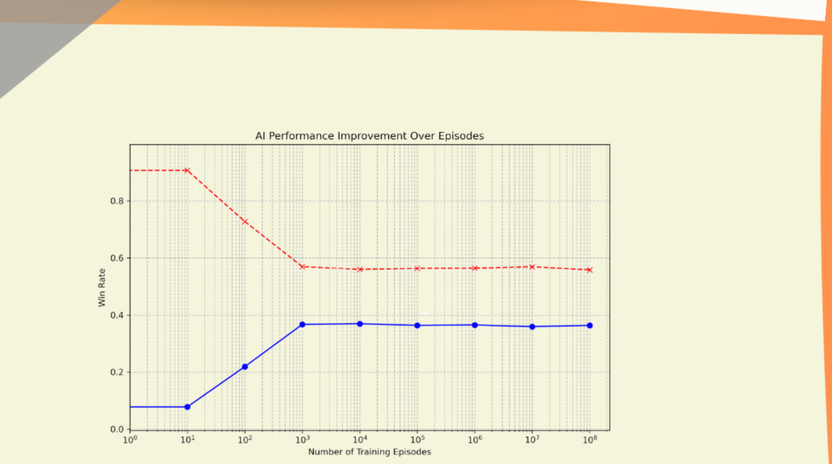
If the player's hand value is less than the dealer's, a negative reward is assigned:
The negative reward is more severe the greater the difference between the dealer's and player's hand values.

3. Draw:

If the player and dealer have the same hand value, a neutral reward with a slight positive bias is given to encourage staying in the game rather than busting:
This slight positive reward increases as the player's hand value approaches 21, promoting higher hand values up to 21.

04 METHODOLOGY

Utilized Q-learning, a model-free reinforcement learning algorithm, to train an AI on thousands of simulated Blackjack games. The AI adjusts its strategy by updating a Q-table based on game outcomes, learning optimal moves in various game scenarios.

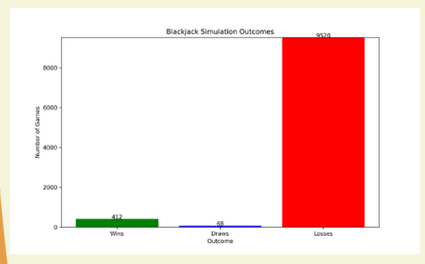


05 DATA

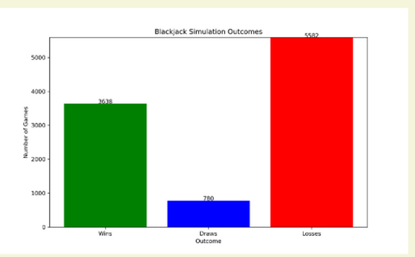
Generated data from over 100,000 simulated games, capturing detailed records of game decisions, outcomes, and AI adjustments. This dataset provides a foundation for analyzing trends, AI learning progress, and strategy efficacy.

06 RESULTS

AI performance analysis shows significant improvement in strategy and increased win rates after extended training periods, as well as detailed outcome metrics validate the AI's capability to adapt and optimize gameplay strategies over time.



Randomized Performance



Trained Q-table Performance