TEXAS HOLD' EM POKER

Application using AI Algorithm

UCS2504 – Foundations of Artificial Intelligence

PROJECT REPORT

Submitted By

Samyuktaa Sivakumar 3122 22 5001 121

Shaun Allan H 3122 22 5001 127

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

Kalavakkam – 603110

November 2024

INDEX

INTRODUCTION

Problem Statement:

The goal of this project is to design and develop a comprehensive and interactive poker game simulation in Unity3D that features two AI agents alongside a user-controlled player. Each AI agent will employ strategic decision-making techniques, primarily utilizing a simplified Monte Carlo simulation and hand strength evaluation to simulate realistic poker play. The agents will evaluate the strength of their hand, predict possible outcomes, and make decisions such as folding, calling, or raising, to compete against the player. The ultimate aim of this project is to provide a dynamic, engaging, and competitive gaming experience, where the user can challenge the AI and improve their poker skills. Through this simulation, players will experience strategic decision-making based on probabilities, simulated card outcomes, and realistic AI behaviors, enhancing the game's authenticity and replayability.

Objectives:

1. Create a 3D poker game environment in Unity with responsive, visually appealing user interfaces.
2. Implement AI agents that evaluate hand strength and simulate possible outcomes to make strategic game decisions.
3. Enable player interaction for actions such as call, raise, and fold.
4. Integrate a simplified Monte Carlo simulation and hand strength evaluation to guide AI decisions.

RULES OF POKER

1. The game is played with 2-10 players per table.
2. Each player is dealt 2 private hole cards face down.
3. There are 5 community cards dealt face up on the table.
4. Players must make the best 5-card hand using their 2 hole cards and the 5 community cards.
5. The game has 4 betting rounds:
    a. Pre-flop: Players bet after receiving their hole cards.
    b. Flop: 3 community cards are dealt. Players bet again.
    c. Turn: A 4th community card is dealt. Players bet again.
    d. River: A 5th community card is dealt. Final round of betting.
6. Players have the following betting options:
    a. Fold: Discard your hand and leave the round.
    b. Bet: Place chips into the pot.
    c. Raise: Increase the current bet.
    d. Call: Match the current bet.
7. At the end of the final betting round (after the river), if multiple players remain, the best 5-card hand wins the pot.
8. Hand rankings (from highest to lowest):
    a. Royal Flush: A, K, Q, J, 10, all the same suit.
    b. Straight Flush: 5 consecutive cards, all the same suit.
    c. Four of a Kind: 4 cards of the same rank.
    d. Full House: 3 of a kind + a pair.
    e. Flush: 5 cards of the same suit, not in sequence.
    f. Straight: 5 consecutive cards, not all the same suit.

g. Three of a Kind: 3 cards of the same rank.

h. Two Pair: Two pairs of cards with the same rank.

i. One Pair: Two cards of the same rank.

j. High Card: The highest card if no other hand is made.

9. After the final round of betting, players reveal their hole cards in a showdown.
10. The player with the best hand wins the pot. If hands are tied, the pot is split.

SOFTWARE REQUIREMENTS AND DESIGN CHOICES

Software Requirements:

1. Development Platform: Unity 3D Engine, C# Programming Language.
2. AI Techniques: Simplified Monte Carlo simulation, hand strength evaluation heuristic.
3. User Interface: 2D/3D graphical interface for player interaction and feedback.
4. Game Logic: Poker rules, betting rounds, AI and user turns.

Design Alternatives:

1. Monte Carlo Simulation Methods: Full Monte Carlo simulations vs. Simplified simulations for efficiency.
2. AI Decision Heuristics: Choosing between hand strength evaluation, rule-based decisions, or neural network-based approaches.
3. Game Type: Traditional poker (Texas Hold'em) vs. simplified poker variations to suit project scope.

Chosen Design:

To achieve our objectives efficiently, we implemented a simplified **Monte Carlo Tree Search (MCTS)** with hand strength heuristics tailored to Texas Hold'em poker rules. This approach allows the AI to make quick, informed decisions, enhancing gameplay without imposing heavy computational demands. MCTS is particularly well-suited here, as it **accounts for probabilities in scenarios where information is partially observable**—a key aspect of poker games. By leveraging probabilistic outcomes in a partially observable environment, the AI can better handle the inherent uncertainty in opponent hands, improving both its strategy and adaptability in the game.

Design Overview

Objective:

- Simulate a poker game with AI decision-making based on hand strength evaluation and predefined thresholds for calls, raises, and folds.
- Use a Monte Carlo simulation-inspired approach to estimate hand strength heuristics and dynamically adjust game states.

Components:

- GameManager: Manages the game's flow, including ante placement, card dealing, round transitions, and determining the winner.
- Player: Represents each player (AI or main player) with attributes like handStrength and methods to adjust funds.
- Decision: Handles AI decision-making based on hand strength thresholds and round progression.
- Dealer: Handles dealing of cards in each round such as flop, river and turn
- Card: The card rankings based on suit and numbers are stored here
- Deck: The deck is the object from which cards are dealt in every round

METHODOLOGY

The algorithm in the Player class evaluates the strength of a poker hand, allowing AI players to make informed decisions during the game. The approach used in the Player class includes two key methods for hand strength evaluation:

1. **EvaluateStrength:** A general strength evaluation method utilising simplified Monte Carlo Simulation, used at the start of the game.
2. **EvaluateHandStrength:** A more detailed evaluation method, used in later rounds, that assesses hand strength based on the player's cards and community cards.

EvaluateStrength

Purpose: This method provides an initial assessment of the player's two-card hand at the beginning of the game (before community cards are dealt).

Algorithm:

- Card Rank Calculation: Evaluate each card in the player's hand based on its rank (e.g., 2 through Ace).
- Pair Evaluation: Check if both cards have the same rank, indicating a pair. Pairs are valued highly in poker, as they offer a strong starting hand.
- High Card: If no pair exists, the hand is evaluated by the highest card's rank (e.g., an Ace is stronger than a King).
- Simulation:| 5 cards are required in poker to form a hand. In the first round (i.e pre-flop) only 2 cards are dealt to players. We randomly choose 20 cards and perform simulations of these cards with the existing 2 cards to identify possible hand combinations and their respective strengths following which we compute an average hand strength.

The result is a numeric strength value that provides a baseline for decision-making, indicating the relative power of a player's starting hand.

EvaluateHandStrength

Purpose: This method provides a more comprehensive evaluation of the player's hand as the game progresses and community cards are revealed. It calculates the strength of the full hand (player's two cards + community cards), aiming to identify the best five-card combination.

Algorithm:

- Combine Cards: Create a pool of cards using the player's hand and the revealed community cards (Flop, Turn, River).
- Identify Hand Rankings:
    - Pairs, Three of a Kind, Four of a Kind: Count the occurrences of each rank to detect pairs or multiples. This is essential for identifying pairs, three-of-a-kind, or four-of-a-kind combinations.
    - Straight: Sort the ranks and check if there are consecutive cards to form a straight.
    - Flush: Count the suits in the combined cards to see if five or more share the same suit.
    - Full House: Check for both a pair and three of a kind.
    - Straight Flush / Royal Flush: If a flush and a straight are found, check if they overlap. For a Royal Flush, confirm that the straight ends with an Ace.
- Rank Assignment: Assign a numeric value to each possible hand type:
    - Royal Flush (highest), Straight Flush, Four of a Kind, Full House, Flush, Straight, Three of a Kind, Two Pair, One Pair, and High Card (lowest).

Choose Best Hand: Out of all possible five-card combinations, select the highest-ranking hand according to poker hand rankings.

Return Hand Strength: This numeric strength value is used to guide the AI's decision-making, with higher values encouraging a more aggressive action (e.g., raising) and lower values leaning towards cautious actions (e.g., calling or folding).

DESIGN AND IMPLEMENTATION DETAILS

1. Game Manager class

The GameManager class orchestrates the entire poker game, managing phases, player actions, and game progression. It uses a singleton pattern to ensure only one instance exists, and its state transitions between phases like Ante, DealCards, Player Turns, DealerTurn, and DeclareWinner.

Key Phases:

- Ante: Players place an initial ante bet.
- DealCards: Each player is dealt two cards, with the main player's cards displayed in the UI.
- Player Turns: Players take turns to call, raise, or fold based on hand strength and the current bet, guided by the Decision class.
- Dealer Turn: The dealer reveals community cards over three rounds (Flop, Turn, River).
- Declare Winner: At the end of the River round, the player with the highest hand strength wins the pot.

Main Methods:

- UpdateGameState: Manages state transitions and initiates phase-specific actions.
- HandleAnte: Sets up ante bets for the main player and AI players.
- PlaceBet: Deducts ante from each player's funds, updating the total pot.
- DealCards: Shuffles and deals cards to players.
- HandlePlayerTurn: Uses Decision to determine each player's action based on hand strength.
- NextTurn: Advances the turn to the next player or dealer phase.
- DeclareWinner: Finds and announces the winner, awarding the pot to the highest hand.

The GameManager provides a seamless flow across phases, utilizing state management and Decision logic to drive AI actions and ensure a competitive game experience

2. Player Class

The Player class represents both AI agents and the main player, with attributes for hand management and funds.

Attributes:

- funds: The player's available funds.
- hand: Stores the player's hand cards.
- handStrength: Numeric evaluation of the player's hand strength, calculated based on card combinations.

Methods:

- AddCardToHand(): Adds a card to the player's hand.
- DeductFunds(): Deducts the ante or bet amount from the player's funds.
- EvaluateStrength() and EvaluateHandStrength(): Evaluate the player's hand to determine its strength, simulating hand strength evaluation similar to a Monte Carlo simulation.
- ClearHand(): Resets the player's hand for the next game.

3. Decision Class

The Decision class uses AI-driven logic to make strategic moves based on hand strength and the current round. It utilizes three thresholds:

- callThreshold: Minimum strength to call the current bet.
- raiseThreshold: Minimum strength to raise.
- maxRaiseAmount and minRaiseAmount: Define the range for random raises.

Attributes:

- gm: Reference to GameManager for accessing the current round, current bet, and pot.
- callThreshold, raiseThreshold: Predefined thresholds determining when to call or raise based on hand strength.

maxRaiseAmount, minRaiseAmount: Bounds for the random raise amount.

Methods:

- MakeDecision(): Central decision-making method where AI agents evaluate their hand strength and round number to decide on an action (call, raise, or fold).
- CalculateCallAmount(): Calculates a call amount that scales based on the current round.

4. Dealer Class

The Dealer class is responsible for managing the dealing of community cards in the poker game. It interacts with the shared Deck to draw cards for each phase of the game (Flop, Turn, and River), ensuring that each card drawn is unique and that player cards are excluded when necessary. The Dealer class provides controlled, sequential access to community cards that form the basis for hand evaluation across all players.

Attributes:

- Deck: A reference to the shared Deck component containing all cards, managed in GameManager and accessed by the Dealer to draw cards.
- Flop: A list of three community cards dealt during the Flop phase and accessible to players for hand evaluation.
- Turn: The fourth community card dealt in the Turn phase.
- River: The fifth and final community card dealt in the River phase.

Methods:

- GetRandomCommunityCardsExcluding:
    - Draws a specified number of community cards while excluding any cards already dealt to the player. This ensures that no duplicate cards appear in the game.
- Process:
    - Filters the deck to exclude any cards passed in the excludeCards list.
    - Randomly shuffles the available cards and selects the specified number.
    - Returns a list of unique community cards that do not overlap with any player cards.
- DealFlop: Draws three cards from the deck to form the initial community card set (Flop) and stores these three cards in the Flop list, which is then used by players to evaluate their hands.
- DealTurn: Draws one additional community card, representing the Turn phase.
- DealRiver: Draws the final community card for the River phase.

5. Card class

The Card class represents a single playing card with a Suit (Hearts, Diamonds, Clubs, Spades) and a Rank (Two through Ace). It includes methods to retrieve the card's rank and suit values, which are useful for sorting, comparisons, and game logic.

Attributes:

- Suit: Enum representing the card’s suit.
- Rank: Enum representing the card’s rank.

Methods:

- GetRankValue: Returns the numeric value of the card's rank for easy comparison.
- GetSuitValue: Returns the numeric value of the card's suit, useful in flush evaluations.
- ToString: Provides a readable string format (e.g., "Ace of Spades") for debugging and display.

6. Deck class

The Deck class represents a deck of playing cards and includes various methods for managing the deck, including initialization, shuffling, drawing, and retrieving random cards while excluding specific ones.

Attributes:

- deck: A list that holds the collection of Card objects representing the playing cards in the deck.

Methods:

- InitializeDeck: Initializes a standard 52-card deck by iterating through all possible combinations of card suits (Hearts, Diamonds, Clubs, Spades) and ranks (Two through Ace).
- ShuffleDeck: Shuffles the deck using the Fisher-Yates algorithm to ensure that cards are randomly ordered.
- DrawCard: Draws a card from the top of the deck and removes it, ensuring that the deck is not empty before drawing.
- GetRandomCardsExcluding: Retrieves a specified number of random cards from the deck, excluding a list of specified cards. The method first filters out the excluded cards, shuffles the remaining ones, and then selects the specified number.

TESTING AND VALIDATION

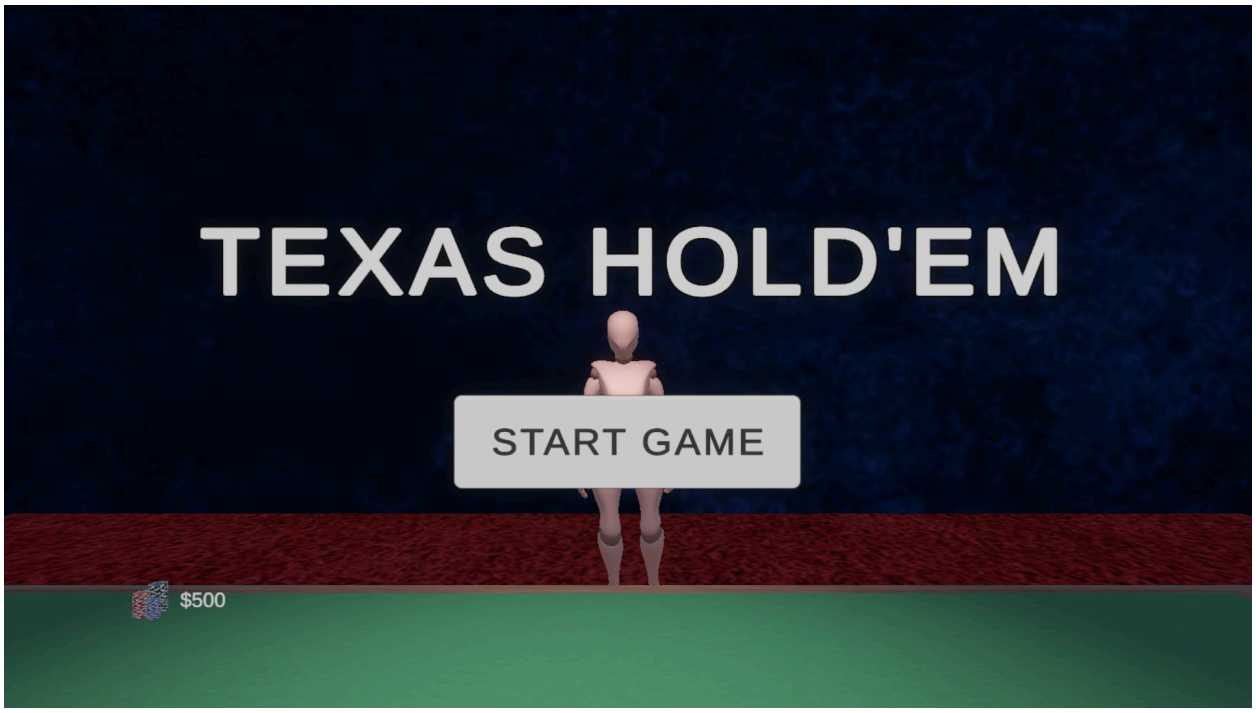
Unit Testing: Each function, such as hand strength calculation, simulation, and AI decision-making, is tested individually to ensure reliability.

Integrated Testing: All components are tested together in a live game environment to validate that AI decision-making correctly interacts with user actions and game state.

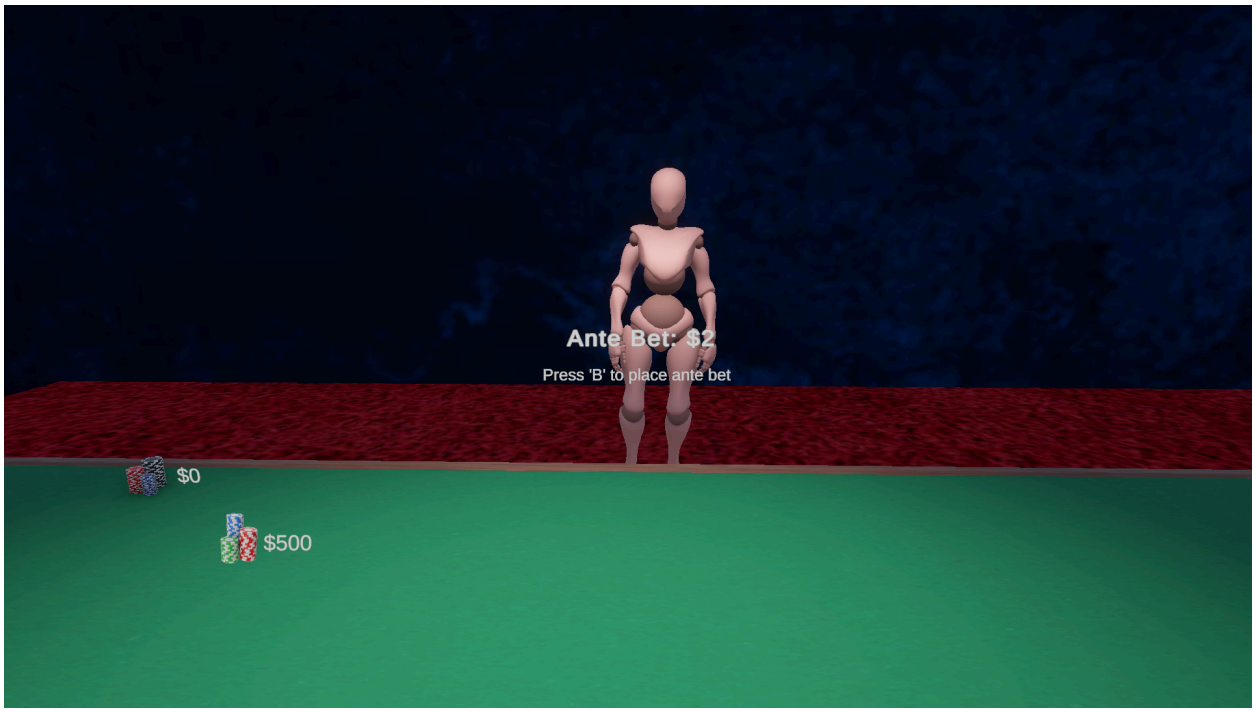
User Testing: Conduct user testing to assess AI difficulty and adjust thresholds for a challenging yet fair experience.

OUTPUTS

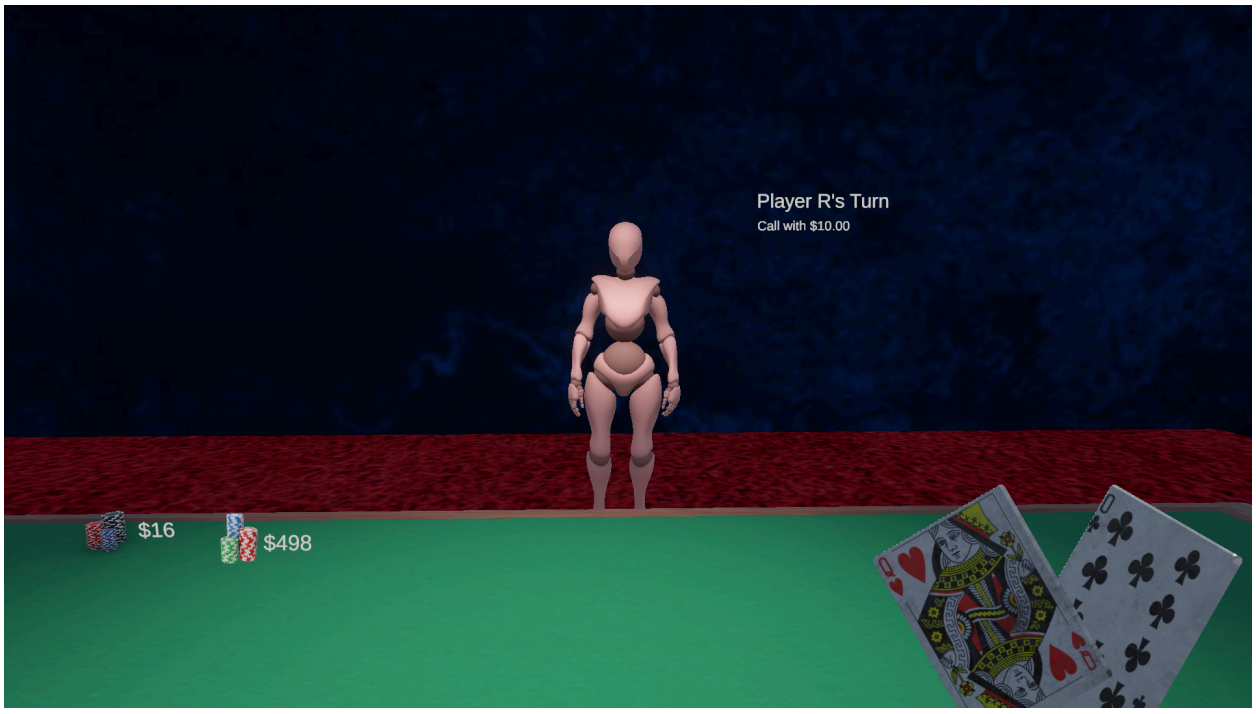
Main Menu:



Ante Bet:



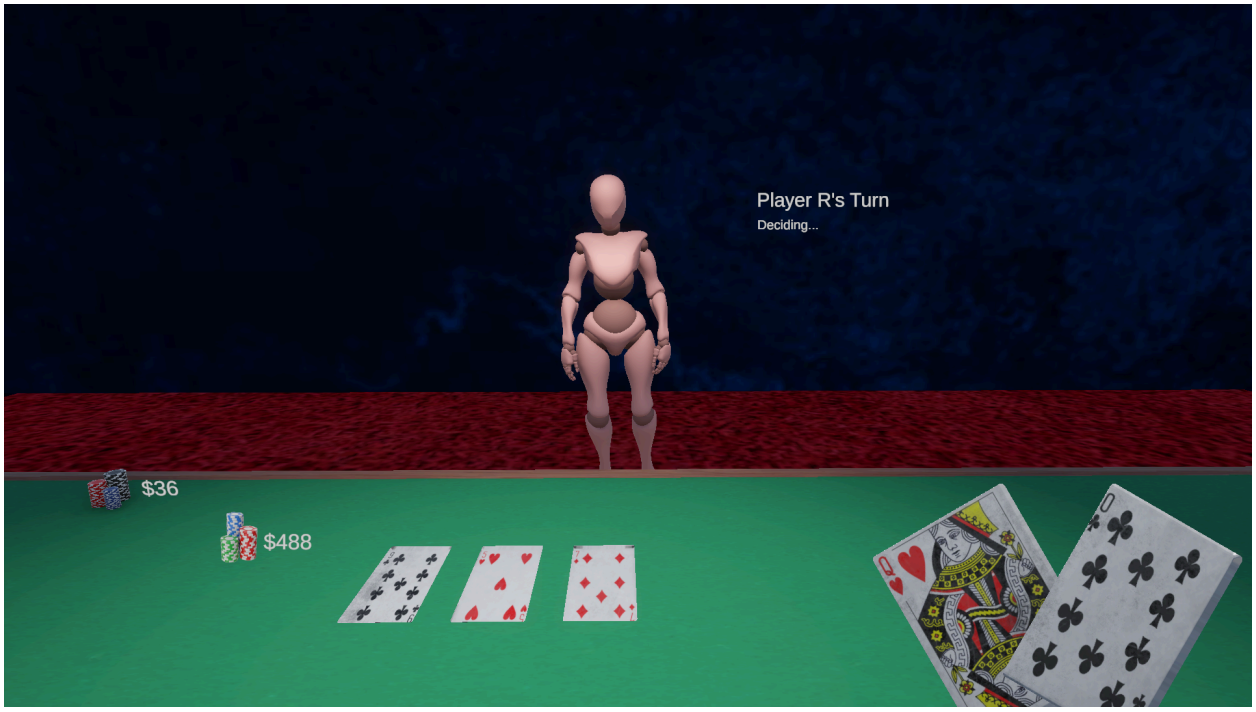
Player R's (AI) Turn:



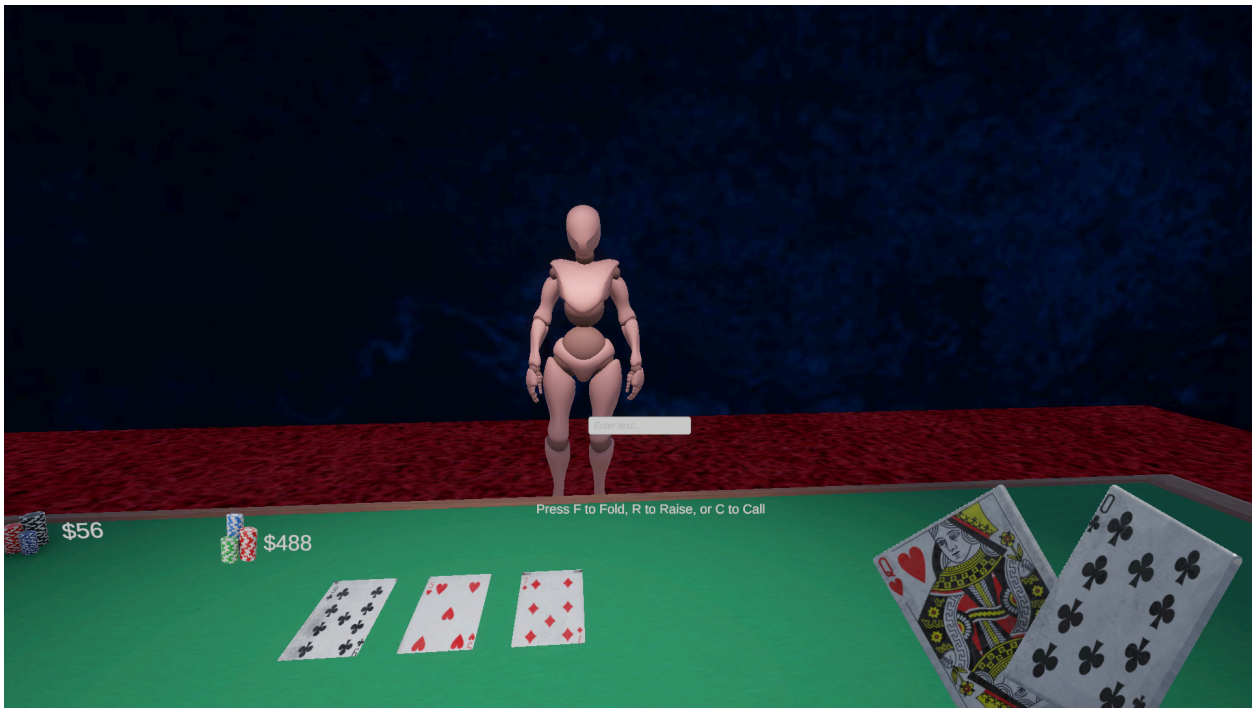
Main Player (Human's Turn):

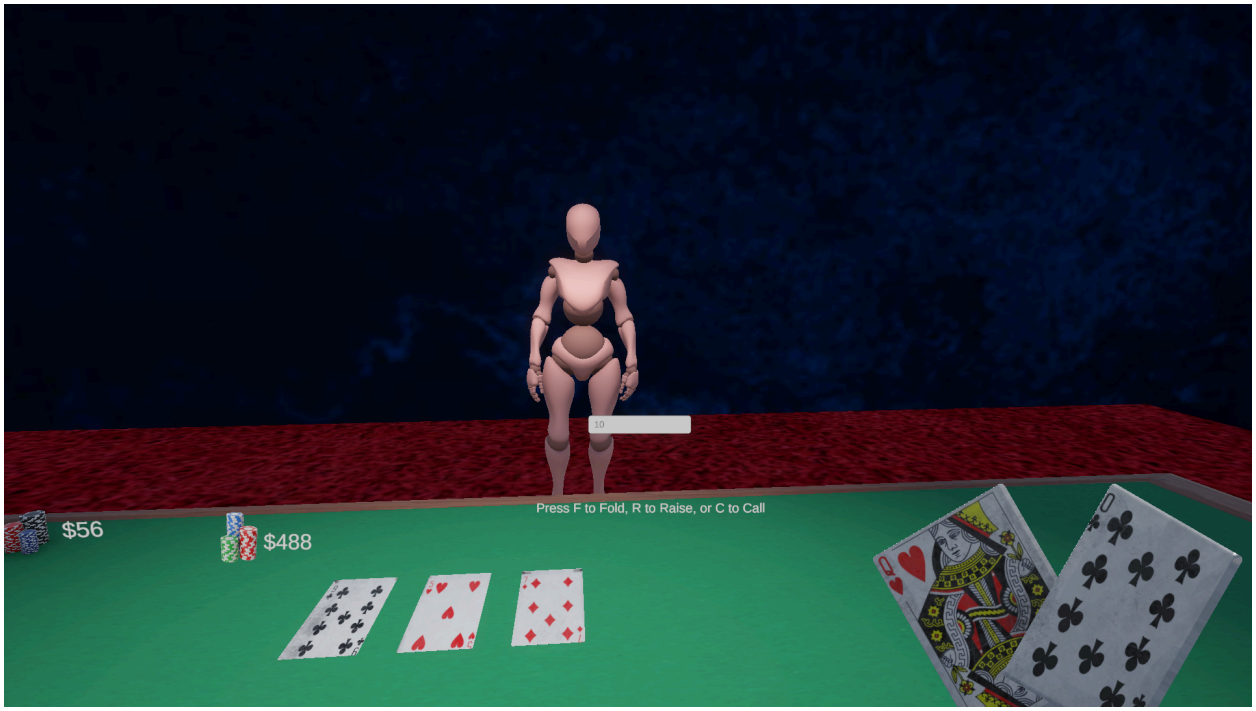
Player L's (AI) Turn:

End of Round 1 – Revealing of Turn (First 3 community cards):

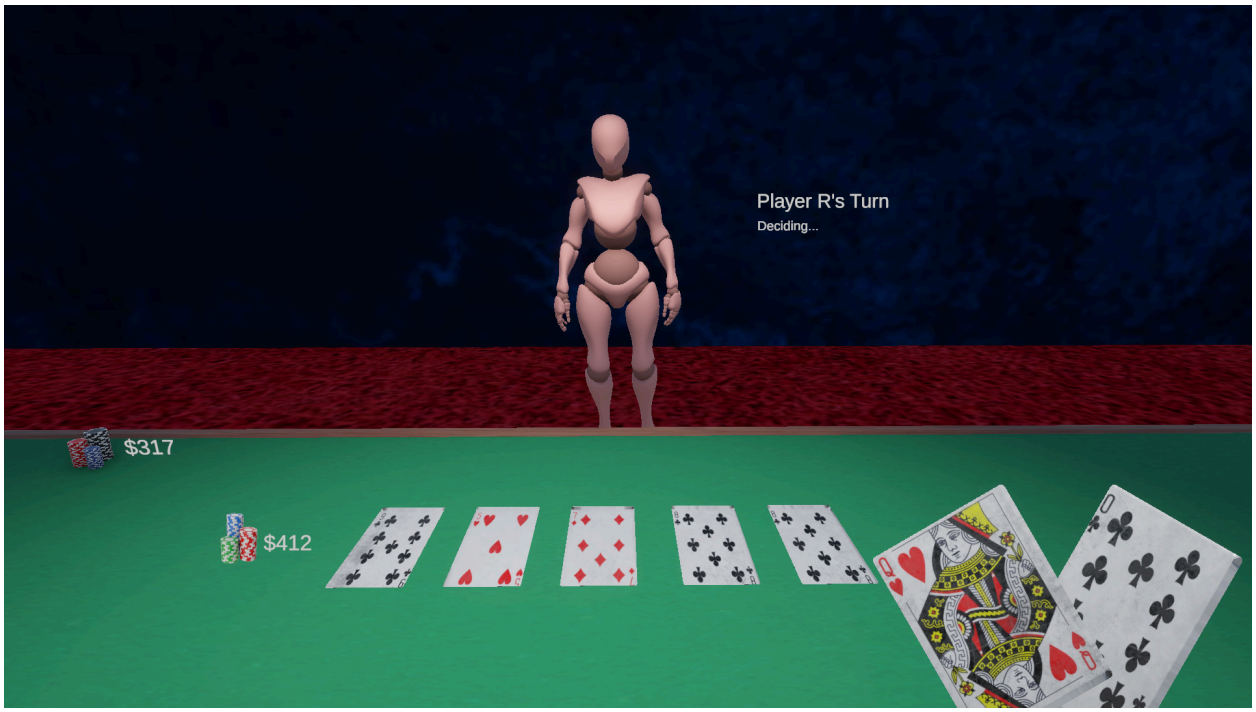


Raising the bet:

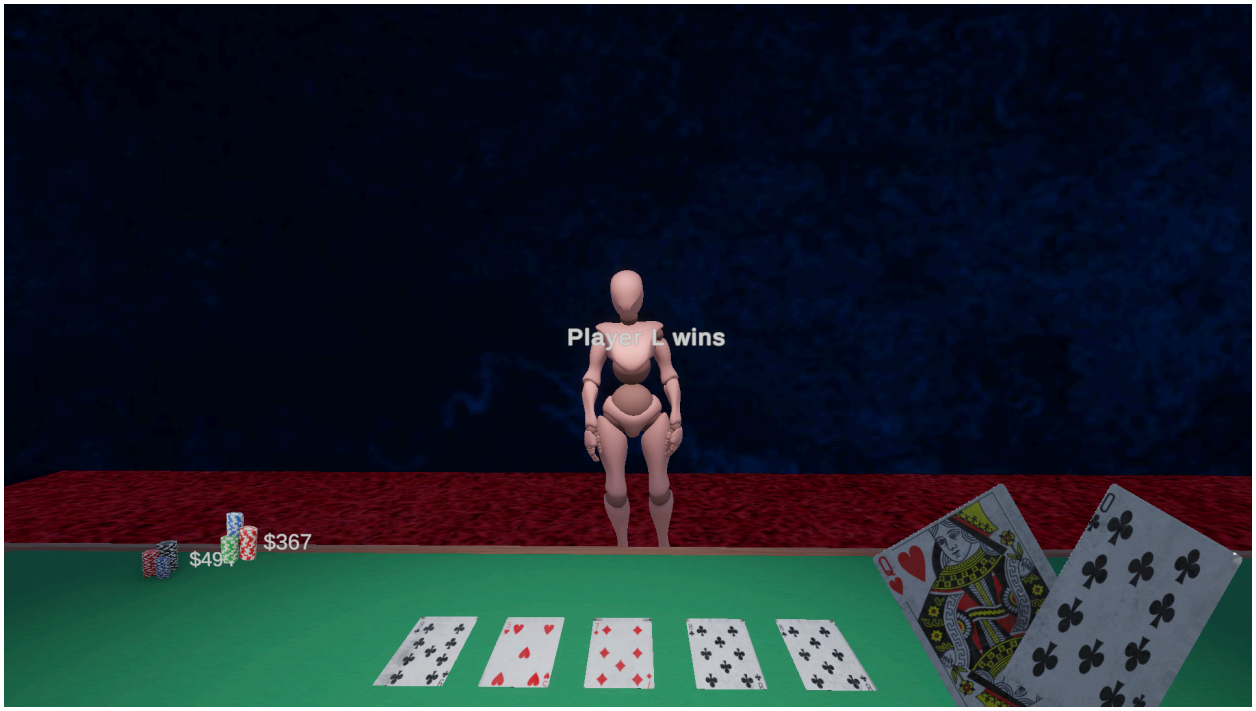




Revealing all community cards:



Winner:



TECHNOLOGICAL IMPROVEMENT

In this project, we focus on enhancing the AI decision-making process by integrating a simplified Monte Carlo simulation and hand strength evaluation heuristics. This allows for computational efficiency while maintaining a realistic and challenging gameplay experience.

Moving forward, technological improvements could include the incorporation of more advanced AI strategies such as neural network-based decision-making to simulate more human-like behavior.

Additionally, we could explore real-time dynamic difficulty adjustment to ensure that AI difficulty scales according to the player's skill level. Another improvement could involve utilizing machine learning techniques to allow the AI agents to adapt and learn from player behavior over time, providing a more personalized challenge.

These advancements would significantly increase the game's realism, replayability, and accessibility, providing a richer experience for the player.