
## FT5004 FINAL PROJECT

# RPS BETTING GAME

ZHAO WEIBO
CAO YAHAN
FU RONGRONG

# Project Overview

Code available at: https://github.com/LexieZB/FT5004-Project

## *Purpose & Objective*

We are going to design a Rock-Paper-Scissors Betting Game, a thrilling and entertaining game that combines the timeless classic of Rock-Paper-Scissors with the excitement of betting. This game is designed to bring joy and excitement to players of all ages while also providing an opportunity for players to test their luck and strategy to potentially earn some extra cryptocurrency. Our RPS betting game consists of three contracts, which will be described in the following.

The objective of the Hand.sol contract is to provide a smart contract that allows players to control their hands and register their moves.

The objective of the HandBattle.sol contract is to provide a smart contract implementation for a battle between two players' hands that have been registered in the Hand contract. Players can place bets on their hands and choose their opponent. If they win, they get the bet from both sides, otherwise they lose their bet.

The objective of the HandMachineBattle.sol contract is to provide a smart contract implementation for a battle between a player's hand and a computer's hand. In this human-robot mode, players do not need to select their opponents and can directly play against a computer-generated hand. The player receives a double-bet if she wins the game; otherwise, she gets nothing.

The contracts provide a transparent and decentralized way for players to participate in a game of rock-paper-scissors and bet on the outcome using cryptocurrency.

## *Scope*

The scope of the contracts is to provide a decentralized platform for playing a game of rock-paper-scissors.

1) The first contract, "Hand," manages the creation and management of the

"hands," which are the game pieces used in the game.

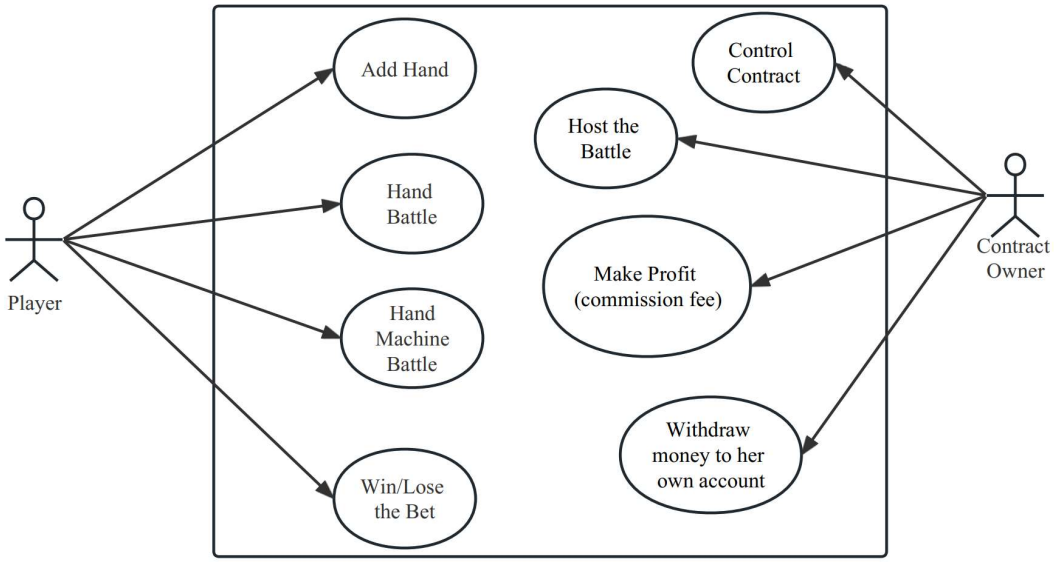
2) The second contract, "HandBattle," manages the human-to-human gameplay by allowing players to select an opponent and bet on the outcome of a game using their hands.
3) The third contract, “HandMachineBattle,” offers a streamlined experience for users who want to engage in the game without the need for finding and selecting an opponent manually.

The contracts aim to ensure that the game is fair and transparent, with all transactions being recorded on the blockchain.

# Technical Architecture and Design

## *Technial Diagram and Design*

To delve into the technical architecture, we can examine various figures that describe the system.




User Case Diagram

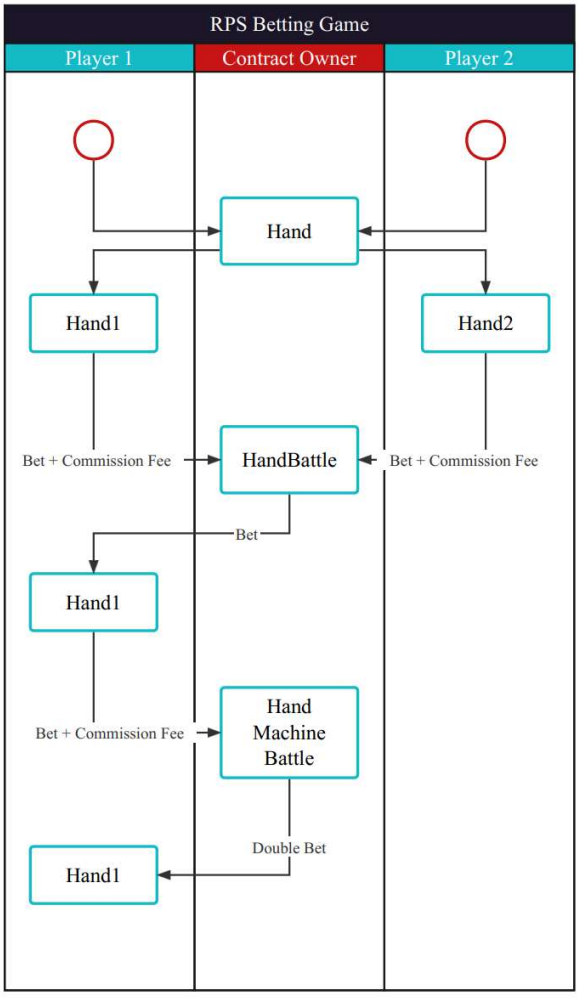
The use case diagram illustrates two roles in the RPS game: the Player and the Contract Owner.

Players can add Hand (you must have at least one hand to get chances to play), put their hand into battle (against another player or machine), and either win or lose their bet. When players put their hand into battle, they need to transfer

money to the game contract, which retains 1 ETH as a commission fee and uses the rest as the bet. Once the battle is over, the bet is automatically assigned to the winner.

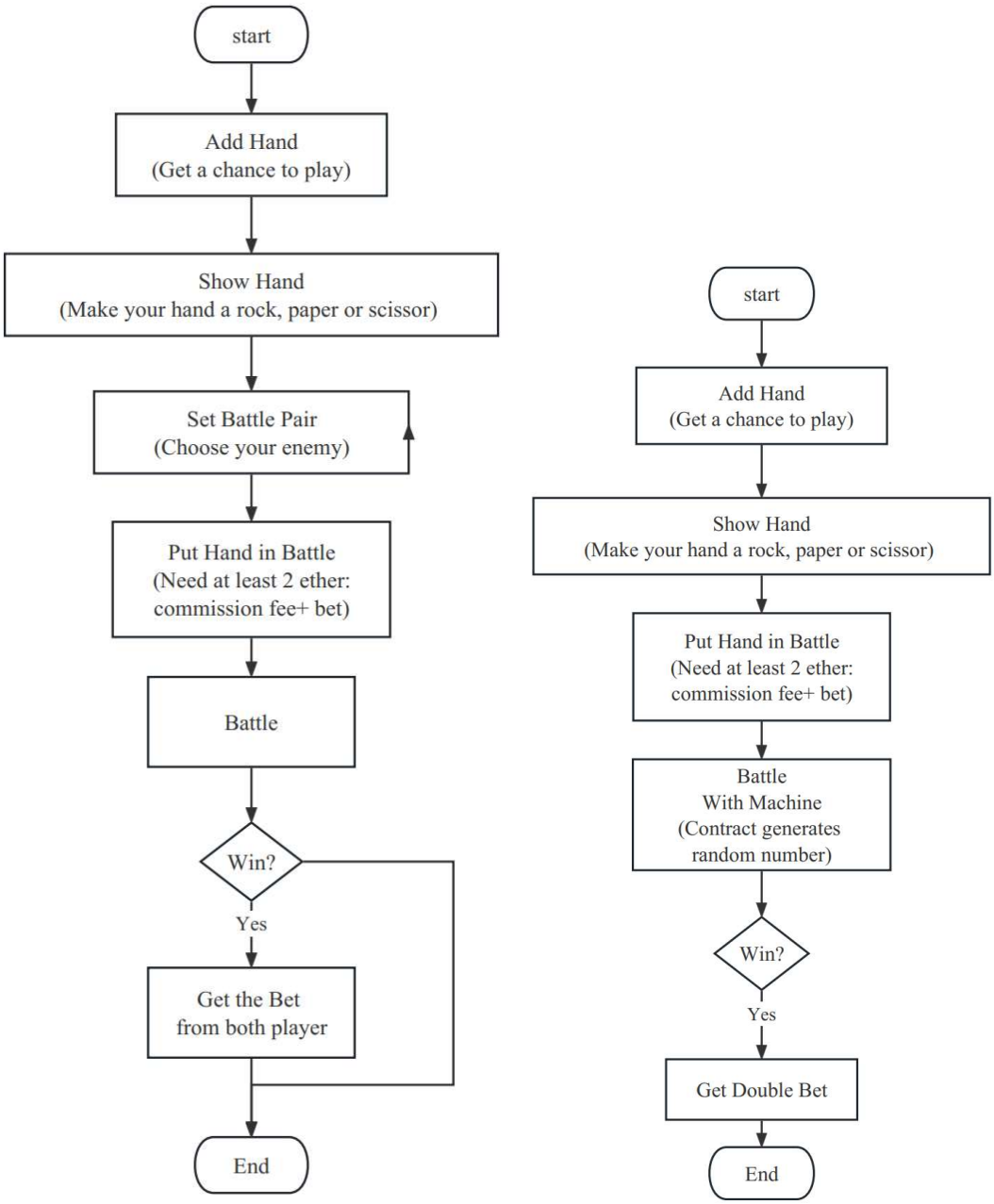
The Contract Owner owns the Hand.sol, HandBattle.sol, and HandMachineBattle.sol contracts. The Contract Owner is responsible for hosting the battles and receives commission fees from them. The commission fee is retained in the contract, but the Contract Owner can withdraw ETH to her own account whenever she wants.

As depicted in the user case diagram, the RPS betting game can benefit both the contract owner and the player. Players can participate in the game to win more ETH by betting on their luck, while the contract owner can earn a profit from the commission fee charged on each game.




Swimlane Diagram

The swimlane diagram indicates that HandBattle requires the participation of two players, while HandMachineBattle can be played with just one player.




Flowchart for HandBattle & HandMachineBattle

From the flowchart, we can see that the RPS game consists of two main parts: RPS with human & RPS with machine.

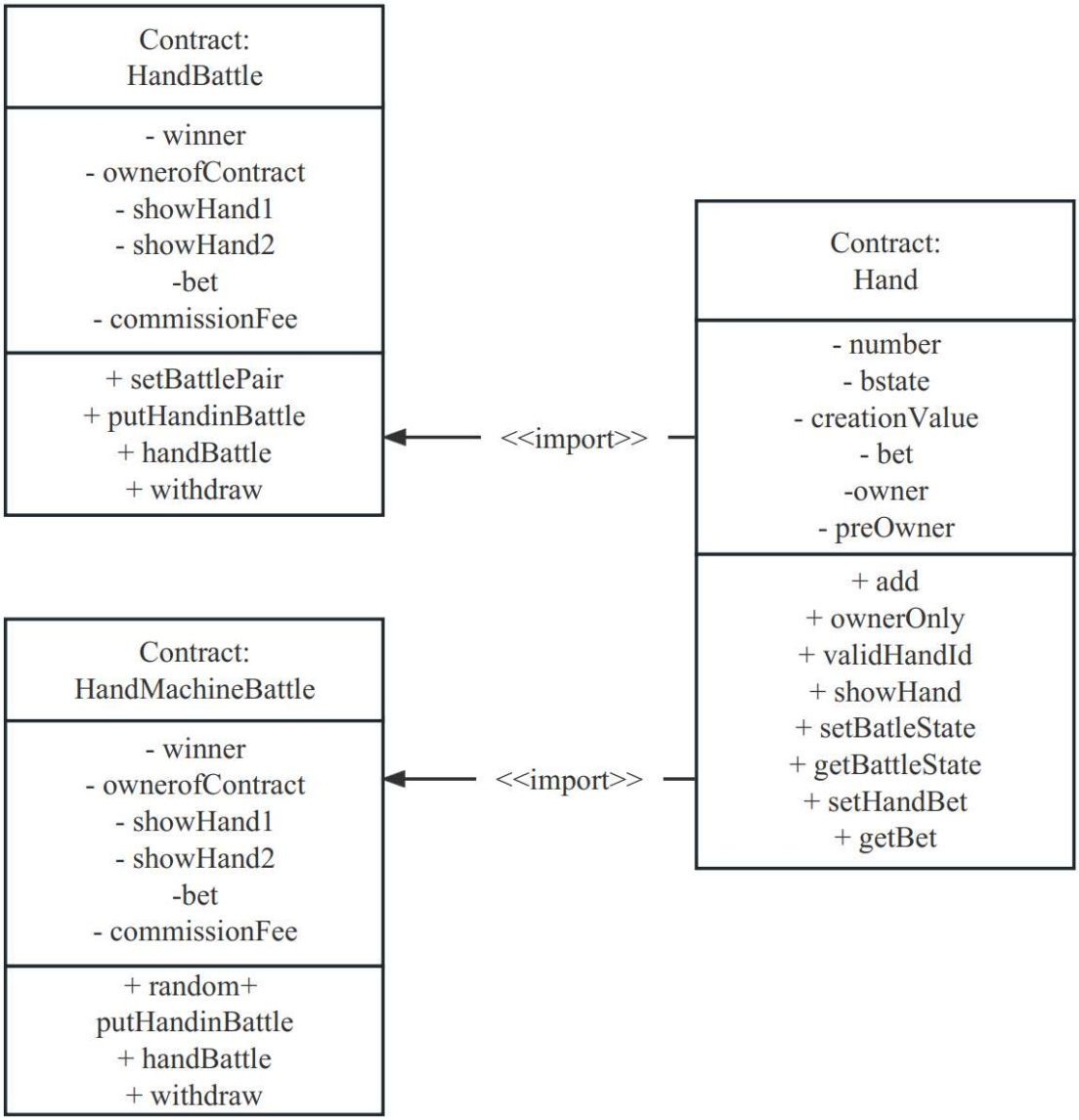
In RPS with human, players can engage in the following steps to play:

1) create a hand (needs at least 0.01 ETH)
2) select an opponent by specifying opponent's addresss
3) put their hand into battle by

   a) specify the hand they want to use
   b) send ETH to the HandBattle contract (Bet + commissionFee)
4) once two players have put their hands into battle, contract owner can start the battle, and the winner of the game is determined based on the rules of rock-paper-scissors. Then
   a) the winner takes the bet from both sides, and the ETH is automatically assigned to the winner's account.
   b) the loser receives nothing.
   c) the HandBattle contract updates the state of the hands to indicate that they are no longer in battle.
5) play again

In RPS with machine, players can engage in the following steps to play:

1) create a hand (needs at least 0.01 ETH)
2) put their hand into battle by
   a) specify the hand they want to use
   b) send ETH to the HandMachineBattle contract (Bet + commissionFee)
3) HandMachineContract generate a random number (1 or 2 or 3) as the opponent of user, and the winner of the game is determined based on the rules of rock-paper-scissors. Then
   a) if player wins, she gets double of her bet
   b) if player loses, she gets nothing
   c) the HandMachineBattle contract updates the state of the hand to indicate that it is no longer in battle.
4) play again




UML diagram

The UML diagram illustrates the relationship between the three contracts in the RPS system. Both the HandBattle and HandMachineBattle contracts import the Hand contract. The HandBattle and HandMachineBattle contracts are independent of each other.

## *Contract & Functions*

In Hand.sol, the main functions are:

1) **setHandNumber(uint256 hand, uint8 number)**: This function is called by the user to set the hand number for a specific hand. The user needs to provide the ID of the hand and the number (1 for rock, 2 for paper, 3 for scissors) to set the number for that hand.

2) **getHandNumber(uint256 hand) view returns (uint8)**: This function is used to get the hand number for a specific hand. The user needs to provide the ID of the hand, and the function returns the number (1 or 2 or 3) of that hand.
3) **setHandBet(uint256 hand, uint256 bet)**: This function is called to set the bet amount for a specific hand. The caller needs to provide the ID of the hand and the bet amount in wei to set the bet for that hand.
4) **getBet(uint256 hand) view returns (uint256)**: This function is used to get the bet amount for a specific hand. The user needs to provide the ID of the hand, and the function returns the bet amount of that hand.
5) **setBattleStatetoIn/Out(uint256 hand)**: This function will set the battle state for a specific hand to "in" / “out”. The caller needs to provide the ID of the hand, and the function sets the battle state of that hand to "in" / ”out”.
6) **getBattleState(uint256 hand) view returns (uint256)**: This function is used to get the battle state for a specific hand. The caller needs to provide the ID of the hand, and the function returns the battle state associated with that hand.

In HandBattle.sol, the main functions are:

1) **constructor(Hand handAddress)**: Constructor function that initializes the "Hand" contract address when deploying the "HandBattle" contract.
2) **setBattlePair(address enemy)**: Function that sets the opponent for a player to battle against. It takes in the address of the opponent and sets them as the opponent for the calling player.
3) **putHandinBattle(uint256 hand,uint256 commissionFee)**: The function allows a player to place their hand in battle. When called by the player, it verifies that the player takes sufficient ETH (1 ETH for commission fee and at least 1 ETH for the bet), that the hand ID corresponds to a hand owned by the player and that the hand has a valid number (1, 2 or 3). If these conditions are satisfied, the function sets the bet for the hand, updates the state of the hand to "in battle", and returns the result.
4) **handBattle(uint256 hand1, uint256 hand2)**: This function is called by

the contract owner when two hands are ready to battle. It accepts as input the IDs of the two hands and verifies that both hands are in the "in battle" state and have been set as opponents for each other. If these conditions are satisfied, the function uses the game logic of rock-paper-scissors to determine the winner of the battle and distributes the bets to the corresponding parties.

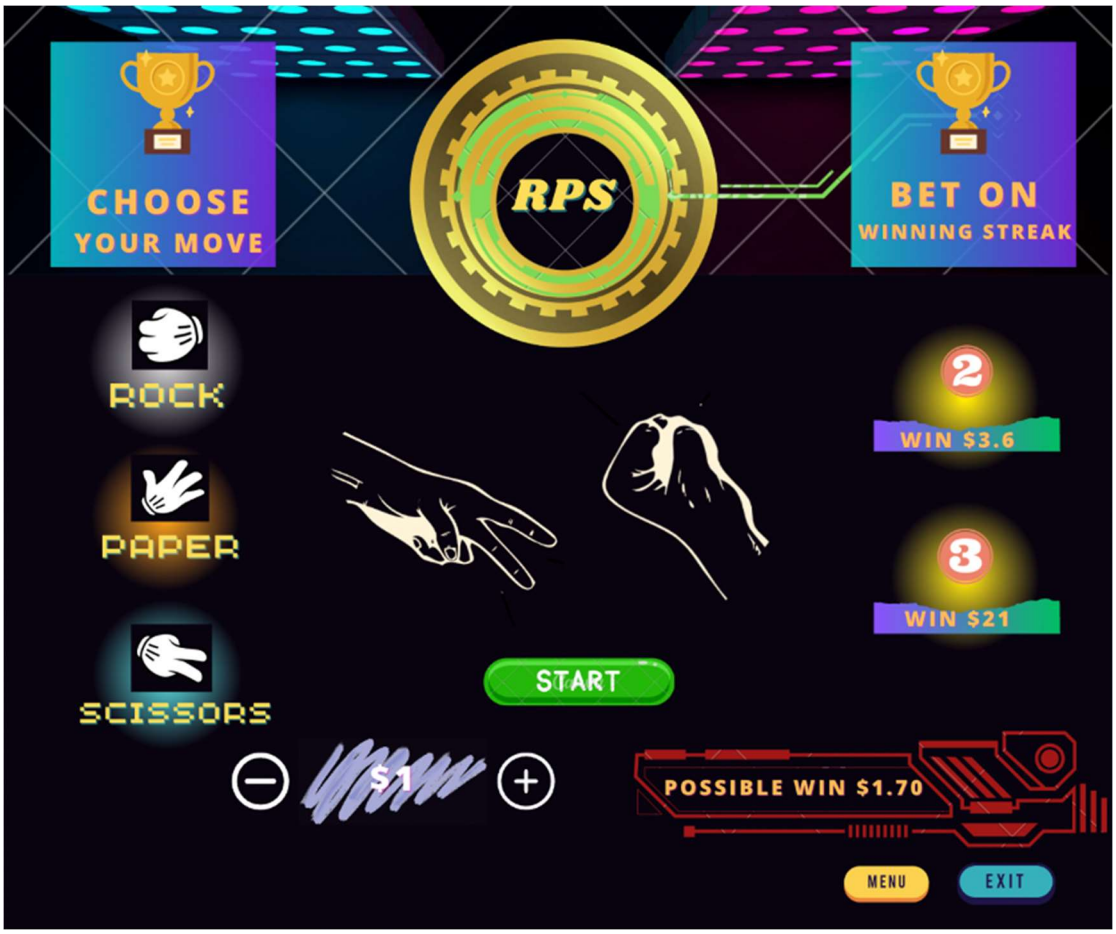
In HandMachineBattle.sol, the main functions are:

1) **random(uint number) public view returns(uint)**: This is the most important function in “HandMachineBattle.sol”. It generates a random number between 1 and 3 based on the current timestamp. This function is used by the handBattle function to generate the number (i.e. rock, paper, or scissors) of the second Hand.
2) **putHandinBattle(uint256 hand,uint256 commissionFee)**: This function allows a player to place their hand in battle. It has the same logic as the “putHandinBattle” function in HandBattle.sol.
3) **handBattle(uint256 hand1, uint256 hand2)**: This function is called by the contract owner when two hands are ready to battle. It has the same logic as the “handBattle” function in HandBattle.sol.

## User Interface


Start Screen of RPS Game

Upon entering the RPS Game, players are greeted with a start screen where they can choose between playing against the computer (AI) or another human player. After making a selection, they are taken to the game interface.




Game Interface

On the left side, gestures represent the player, while on the right side, gestures represent the other player or the computer. First, users select their bet amount for the round in the bottom-left corner. Then, they can see the potential winnings for the round in the bottom-right corner if they emerge victorious.

Next, players choose their gesture for the round on the left side. Once both parties have made their choices, they can press the START button. The screen's center will display each player's choice, determining the winner and loser. The right side will show the amount of money won in each round. After a round concludes, players have the option to continue playing or exit the game.

# Goals, Challenges, and Business Cases

## *Goals and Challenges*

Our primary goal is to develop an engaging betting game that functions independently without the need for any central authority. The game will ensure a smooth experience for players while upholding the highest levels of transparency and security since the rules are enforced through smart contracts. Another objective is to implement a human-robot mode, which will enable players to place bets against an automated opponent. This feature caters to players seeking a single-player experience, as well as those wishing to hone their skills before participating in the human-human mode.

Recently, we have made significant progress in the development of the human-robot mode. The game now includes a function where players can click on "putHandinBattle" to set their bet, identical to the process in the human-human mode. Afterward, the contract owner initiates the "handbattle," during which the automated opponent randomly generates a rock-paper-scissors hand to compete against the player. If the player wins, they receive double their bet, but if the automated opponent prevails, the player loses their bet. This innovation enhances the game and provide a unique and engaging experience for our users.

One of the main challenges we face is the development of a user-friendly interface that allows players to easily access the game, place bets, and withdraw winnings. Achieving this requires seamless integration between the blockchain back-end and the front-end user interface. Our ambition is to create an intuitive platform that appeals to users with varying degrees of blockchain expertise, thereby expanding the game's reach.

## *Business Cases*

Integration with existing online casino platforms: Incorporating our decentralized RPS Betting Game into existing online casinos can provide an additional form of entertainment for users. Our game's decentralized and transparent nature addresses some of the most prevalent concerns in the

online gambling industry, such as fairness and trustworthiness. The game can be added to a casino's website or mobile application, offering users a new way to enjoy their leisure time. Furthermore, it can be easily customized to align with the casino's branding and visual identity, ensuring a cohesive user experience.

White-label solutions are products or services created by one company and rebranded by another for resale under their brand. Our decentralized betting game can be offered as a white-label solution for businesses looking to create their own branded betting games. This approach allows companies to customize the game rules and appearance while leveraging the smart contract infrastructure to guarantee a fair and decentralized experience. For instance, a sports bar chain could develop a branded version of the game to engage patrons during live sports events. Users could place bets on various aspects of the game, such as the final score or individual player performances. This interactive experience can be effortlessly integrated into the sports bar's existing website, enhancing customer engagement and promoting brand loyalty simultaneously.

## Conclusion

The development of a decentralized and user-friendly betting game presents unique goals and challenges. By focusing on creating an intuitive platform, we aim to expand the reach of our game to a diverse user base. Furthermore, the potential business cases, such as integration with existing online casinos and white-label solutions, showcase the versatility of our decentralized betting game in various industries.