Matching Pennies Blockchain Game

Game Overview

Matching Pennies is a simple two-player zero-sum game implemented as a smart contract on the Sepolia Testnet. In this game, each player selects one of two options—commonly represented as 0 or 1. The rules are as follows:

- **If both players choose the same value:** Player A (the initiator) wins.
- **If the players choose different values:** Player B (the joiner) wins.

Each participant contributes a wager (0.05 ETH) when joining the game. The winner receives the entire pot (0.1 ETH), less any applicable gas fees.

How to Play

1. **Game Initiation:**
    - **Player A** starts the game by calling the `startGame()` function.
    - When starting the game, Player A submits a cryptographic hash of their choice (using a commit-reveal scheme). This ensures that they cannot change their selection later once Player B has joined.
    - At this stage, Player A deposits 0.05 ETH as their wager.
2. **Joining the Game:**
    - **Player B** then calls the `enterGame()` function.
    - Player B submits their chosen value (0 or 1) in plaintext along with their 0.05 ETH wager.
    - The smart contract records the opponent's selection and the wager.
3. **Reveal Phase:**
    - Once Player B has joined, Player A must reveal their original choice using the `revealSelection()` function.
    - The contract verifies that the revealed value matches the original hash (thus proving Player A's commitment) and then determines the winner:
        - If the revealed choice matches Player B's, Player A wins.
        - Otherwise, Player B wins.
4. **Claiming Winnings:**
    - After the game is finalized, the winning player calls `claimWinnings()` to transfer the accumulated pot (0.1 ETH) to their account.
5. **Refund Option:**
    - If a player does not follow through (for example, if Player A fails to reveal within a set time limit), the `requestRefund()` function can be used so that both players can recover their wager, preventing funds from being locked indefinitely.

Matching Pennies Blockchain Game

Key Design and Security Decisions

- **Commit-Reveal Scheme:**
  The use of hashed choices (commit phase) prevents Player A from cheating by changing their selection after seeing Player B's move.
- **Wager Distribution:**
  Both players contribute an equal amount (0.05 ETH). The funds are held securely in the contract and only transferred after a valid game completion, ensuring that the reward is paid solely from the players' wagers.
- **Refund Mechanism:**
  A timeout-based refund function (`requestRefund()`) is implemented to prevent funds from being locked in cases where one player fails to complete their required action (e.g., not revealing the choice).
- **Gas Efficiency:**
  The contract functions are optimized to reduce gas consumption (e.g., `startGame()` ~200,000 gas, `enterGame()` ~150,000 gas), though Player A tends to incur slightly higher costs because they perform both the commit and reveal phases.
- **Security Against Vulnerabilities:**
  Several potential hazards (such as frontrunning during the reveal phase and intentional game stalling) have been considered. For example, requiring both players to reveal within a designated timeframe reduces the window for an attacker to exploit on-chain data, and best practices (e.g., checks-effects-interactions pattern) are followed to mitigate reentrancy attacks.

Playing the Game: A Step-by-Step Walkthrough

1. **Starting a Game (Player A):**
   - Generate a random salt.
   - Choose a value (0 or 1) and compute its hash combined with the salt.
   - Call `startGame(hashedSelection)` while sending 0.05 ETH.
2. **Joining a Game (Player B):**
   - Identify an available game by its ID.
   - Decide on your selection (0 or 1) and call `enterGame(gameId, selection)` with 0.05 ETH.
3. **Revealing the Choice (Player A):**
   - After Player B has joined, call `revealSelection(gameId, originalChoice, salt)` to prove your commitment.
   - The contract automatically compares the submitted value against Player B's selection to determine the winner.
4. **Claiming Winnings:**

- The victorious player uses `claimWinnings()` to withdraw the total ETH prize from the contract.

5. **In Case of Stalls:**
   - If the game does not progress (for instance, if Player A fails to reveal in time), either party can use the refund mechanism (`requestRefund()`) to recover their funds.