School of Informatics
Blockchains and Distributed Ledgers

Assignment 2

Erodotos Demetriou (s2187344)

Smart Contract High-Level Description

- Simple description of the game
- Who pays for the reward of the winner?
- How is the reward paid to the winner?
- Anticheat measures taken
- What data type/structure did you use for the pick options and why?

Gas Costs Evaluation

Gas Costs

Contract Owner Fees	
Contract Deployment	1,798,945

Players Fees	***Player A***	***Player B***
giveHiddenBet()	135,643	84,620
giveRealBet()	81,007	83,807
evaluate()	40,423	
withdraw()	43,717	

- Cost of deploying and interacting with contract
- Comment on gas fairness
- Techniques to make smart contract more fair and cost efficient.

Potential Hazards and Vulnerablities

- List of security mechanisms used to mitigate such hazards

Security vs Performance

- security vs performance trade-offs

Fellow Student Contract Analysis

- Vulnerabilities
- How the player can exploit these vulnerabilities and win the game ?
- Include code snippets

Smart Contract Execution History

Implementation Code

```
pragma solidity 0.8.0;

/// @title Matching pennies game
/// @author Erodotos Demetriou
contract Game {
    uint256 public _playDeadline;
    address public _playedLast;
    address public _adr_playerA;
    address public _adr_playerB;

    mapping(address => Bet) public _bets;

    struct Bet {
        string _realBet;
        bytes32 _hiddenBet;
        bool _isValid;
    }

    uint8 public _locked = 0;
    uint8 public _playersJoined = 0;
    address public _winner;

    event Play(address indexed _playerAddress, uint8 _playerNumber);
    event WinnerAnnounced(address indexed _winner);
    event NewGame(string _newGame);

    /// @notice Takes 1 ETH as bet stake and set contract state accordingly
    /// @param _bet This is an obscured 32-byte string produced after
    /// hashing (real_bet || salt)
    function giveHiddenBet(bytes32 _bet) public payable {
        // Perform checks
        require(
            _locked == 0,
            "There are already 2 players. Wait for the next game to start!"
        );
        require(msg.value == 1 ether, "You must bet 1 ETH");
        require(
            _bets[msg.sender]._hiddenBet == bytes32(0),
            "You have already put your bet"
        );

        // Change the smart contract state
        _playersJoined += 1;
        _bets[msg.sender]._hiddenBet = _bet;
        _playDeadline = block.timestamp + 10 minutes;
        _playedLast = msg.sender;
```

```
47
48          // Lock the contract if both players beted
49          // and emmit events to announce their participation
50          if (_playersJoined == 2) {
51              _locked = 1;
52              _adr_playerB = msg.sender;
53              emit Play(msg.sender, 2);
54          } else {
55              _adr_playerA = msg.sender;
56              emit Play(msg.sender, 1);
57          }
58      }
59
60      /// @notice Receives the players real bets
61      /// and their salt and check the initial bet validity
62      /// @param _realBet A string representing the real bet
63      /// @param _salt The salt that the message sender used
64      /// to create his initial obscured bet
65      function giveRealBet(string memory _realBet, string memory _salt)
            external {
66          require(_playersJoined == 2, "Wait for player #2 to join the game");
67          require(
68              keccak256(abi.encodePacked(_realBet, _salt)) ==
69                  _bets[msg.sender]._hiddenBet,
70              "Error: Provided invalid input: Abort"
71          );
72
73          _bets[msg.sender]._realBet = _realBet;
74          _bets[msg.sender]._isValid = true;
75
76          _playedLast = msg.sender;
77          _playDeadline = block.timestamp + 10 minutes;
78      }
79
80      /// @notice Calculates the game winner
81      function evaluateWinner() external {
82          require(
83              _bets[_adr_playerA]._isValid && _bets[_adr_playerB]._isValid,
84              "Error: Players did not provide their real bet"
85          );
86
87          if (
88              keccak256(abi.encode(_bets[_adr_playerA]._realBet)) ==
89              keccak256(abi.encode(_bets[_adr_playerA]._realBet))
90          ) {
91              _winner = _adr_playerA;
92          } else if (
93              keccak256(abi.encode(_bets[_adr_playerA]._realBet)) !=
94              keccak256(abi.encode(_bets[_adr_playerA]._realBet))
95          ) {
96              _winner = _adr_playerB;
97          }
98
99          // Emit event
100         emit WinnerAnnounced(_winner);
101     }
102
103     /// @notice Let a player to stop the game and get
104     /// refund in case his opponent griefs
105     function requestRefund() external {
```

```
106         // Checks
107         require(
108             block.timestamp > _playDeadline &&
109                 msg.sender == _playedLast &&
110                 _winner == address(0),
111             "You are not allowed  to request a refund yet!"
112         );
113 
114         gameReset();
115     }
116 
117     /// @notice Allows the winner to withdraw his reward
118     function withdraw() external {
119         // Checks
120         require(msg.sender == _winner, "You are not the winner!");
121 
122         gameReset();
123     }
124 
125     /// @notice Send money to the winner or the
126     /// refund requestor and reset game variables for a new round
127     function gameReset() internal {
128         _locked = 0;
129         _winner = address(0);
130         _playersJoined = 0;
131         _bets[_adr_playerA] = Bet("", bytes32(0), false);
132         _bets[_adr_playerB] = Bet("", bytes32(0), false);
133         _adr_playerA = address(0);
134         _adr_playerB = address(0);
135         _playDeadline = 0;
136 
137         // Reward/Refund transfer
138         (bool success, ) = msg.sender.call{value: 2 ether}("");
139         require(success, "Error: Withdraw unsuccessful");
140 
141         // Emmit event
142         emit NewGame("New game spots available");
143     }
144 }
```