Coursework 2

**Description**

In terms of the design of this smart contract, there are four major things that need to be considered. Firstly, in one single round of the game, one of the players will be in charge of rolling the dice and the other one is in charge of paying the reward to the winner. Before the game starts, these two players will need to deposit 5 ETH into the account as the reward. If player A rolls the dice at the beginning of the game, then no matter whether A wins or loses, player B will be in charge of paying the reward to the winner. Then the player in charge of paying the reward to the winner uses the **withdraw ()** function, which returns the original deposit of the winner including the reward to the winner and returns the original deposit of the loser subtracting the debit to the loser. Another issue that needs to be considered is that the players can cheat. In this smart contract, when the players join the game, they are required to input a string of seeds, which will be used to generate a random number of dice. However, one thing that may happen is that the player can use their seed to cheat by using their seed. To avoid these issues, a random number of dice will be generated by using the hash function keecak-256, which will take the seed of the player and the address of the player who does not roll the dice combined with the timestamp, and since EVM executes the sequential transactions, which means that the time when different player join the game will never be the same. So, as long as the player doesn’t know the address of the opposite player, he is not able to predict the dice number. In terms of the data type and data structure used in this smart contract, considering there is more than one player in one single round of the game and each player has members of different data types (bool, uint256, string, etc), struct is used in this smart contract to store different types of members for one player as shown in Figure 1, which has the benefit of easy maintaining when having small collections of data type. Also, considering the fact that using uint256 can cost less gas than uint, so most of the integres are stored using the data type uint256.

Text

Description automatically generated

Figure 1: Data type of struct used in contract

**Gas evaluation**

. Usually, contract deployment is the most expensive, since it is a combination and according to the transaction, the result is around 2.1 million Wei as shown in Figure 2.

Graphical user interface, text, application

Description automatically generated

Figure 2: Gas Evaluation of Contract Deployment

In terms of the gas evaluation of the interaction of this contract, there are three displaying functions: show\_A\_balance(),show\_B\_balance(),and show\_player\_number(), which are used to show changes of important parameters in this smart contract. The cost of using these functions is quite low since when calling these functions, it just returns parameters, so according to the estimated gas from Remix, the show\_A\_balance and show\_B\_balance() functions, which are used to show the balance of player A and player B, respectively ,cost around 4680 Wei while for that of show\_player\_number(),which displays the current number of players is around 2460 Wei as shown in Figure 3. The reason why the cost of the first two functions is about 1.5 times bigger than the third is that, in the first two functions, it returns 0 in **if-else** statement.

Text

Description automatically generated

Figure 3: Gas Evaluation of three displaying functions

Theoretically, if there are more variables modified (complexity of one transaction), the cost of a function will be higher, since there are more variables, which need to be allocated memory. The **deposit()** function is used to transfer money from the player’s account to the contract account. According to the result of the cost shown in Figure 4, it is around 44 thousand Wei, which is much bigger than that of the previous three displaying functions. The reason for that is that player A and player B are modified in the deposit function, while in the previous displaying functions, variables are not allowed to be modified.

Graphical user interface, text, application

Description automatically generated

Figure 4: Gas Evaluation of **Deposit()**

As shown in Figure 5, the gas cost of function **withdraw()** is around 42 thousand Wei, which is quite close to that of **deposit()** function**.**

According to the result of the cost of the function **join ()** as shown in Figure 5, it costs around 130 thousand Wei, which is higher than that in **deposit()** function, since it has more variables modified.

Graphical user interface, text, application

Description automatically generated

Figure 5: Gas Evaluation of **Join ()**

In function **EndGame(),** the result of the gas evaluation is around 27 thousand Wei as shown in Figure 6, which is smaller than the previous functions, and the reason for this result is that it doesn’t have many variables modified.

Graphical user interface, text, application

Description automatically generated

Figure 6: Gas Evaluation of **EndGame()**

In terms of the function **Rolling()**, its gas cost is around 95 thousand Wei as shown in Figure 7. The reason why it is bigger than the previous functions is that it has a lot of variables modified in the **if-else** statement.

**Graphical user interface, text, application

Description automatically generated**

Figure 7: Gas Evaluation of **Rolling()**

In this contract, to achieve the gas fairness, this contract is designed to make sure that the number of steps each player needs to take to interact with the contract is the same. So, as mentioned in the **Description** section, apart from the other steps that these two players both need to take: Join(), EndGame(), Deposit(), one of the players will need to roll the dice and then the other will need to withdraw the money (including the reward for the winner, and the rest of the money for the loser) from the contract to pay the reward to the winner . So, the gas difference will be reduced to the gas difference between these two functions: **withdraw(),** **Rolling().** According to the result in one single game, the player rolling the dice pays for around 96 thousand Wei, while for that of the player who withdraws the money pays for around 50 thousand Wei. Compared with the number of variables modified in **withdraw()** function, in the **Rolling ()** function, it has more variables modified, which is the reason why it will cost more gas as mentioned before.

As mentioned before, using **Rolling()** functioncan cost more gas than using **withdraw(),** and to mitigate this issue, reducing the number of modified variables could be the direction optimization. As shown in Figure 8, the variable Dice has been modified twice under different conditions, so, it can be optimized by putting the modification outside the **if-else** statement.

Text

Description automatically generated

Figure 8: Code snippet of Rolling()

**Potential Hazards**

1. **Issues of random number generation**

To generate a random number when rolling the dice, this contract uses the combination of the timestamp and the player’s seed, combined with the other player’s address. However, the **Rolling()** function is set as public in this contract, which means it can be called by any user, without taking on any tasks, which may affect the randomness of rolling.

1. **Timestamp manipulate**

Also, the timestamp can be manipulated by the miners, which means that miners can manipulate the random number as expected. To mitigate these issues, the other player’s address is added to increase the complexity of decoding the hash value of the **keccak256(abi.encodePacked())** function.

1. **Uint256 underflow**

Also, most of the integers are stored in the uint256, which can have issues of the underflow. For example, in this contract, the player’s balance is stored as uint256 and if the player’s balance is 0, if at this time, the player’s balance is subtracted from a positive integer, underflow can happen. This issue is usually not detected when the version of solidity is smaller than 0.8 when the contract complies. To detect this issue, the solidity version in Remix is set to be bigger than 0.8 and every time there is a subtraction happening to a uint256 variable, it will be checked first, if the result is smaller than zero, it will set the result to zero directly.

**Tradeoffs and choices**

In terms of the tradeoffs in this contract, gas fairness cannot be achieved perfectly, since one of the players need to roll the dice while the other need to withdraw money, which will lead to different gas cost, and this contract cannot totally avoid this issue. So, instead of trying to make the two players have the exact same steps, this contract tries to mitigate it by reducing the gas cost of the **Rolling()** function by reducing the number of modified variables.

**Analysis of fellow student’s contract**

In Xiaozhou’s contract, the major issue is that his contract uses two players’ seed and the player B’s timestamp to generate the random number as shown in Figure 8. However, the timestamp can be manipulated by the miners and if the player B wants to cheat with miners and share the reward, it would be very easy.

Text

Description automatically generated

Figure 8: Xiaozhou’s code snippet of generation of random number

Also, as shown in Figure 8 ,in Xiaozhou’s contract, apart from those common steps, which both A and B need to take, B always has one more step to take, which is the dice rolling, and hence his total gas cost will be bigger than that of player A.

**Transaction History**

{

  "accounts": {

    "account{0}": "0x1C924717143de1ba4A4AB976Fefd2d1150dDBD16"

  },

  "linkReferences": {},

  "transactions": [

    {

      "timestamp": 1667143358434,

      "record": {

        "value": "4000000000000000000",

        "inputs": "()",

        "parameters": [],

        "name": "deposit",

        "type": "function",

        "to": "created{undefined}",

        "from": "account{0}"

      }

    },

    {

      "timestamp": 1667143372769,

      "record": {

        "value": "0",

        "inputs": "(uint256)",

        "parameters": [

          "125443"

        ],

        "name": "joinGame",

        "type": "function",

        "to": "created{undefined}",

        "from": "account{0}"

      }

    },

    {

      "timestamp": 1667155705600,

      "record": {

        "value": "0",

        "inputs": "()",

        "parameters": [],

        "name": "",

        "type": "constructor",

        "abi": "0x68422588f8defd750baa1a778970a5caf2426fe30d05162a61bfa1c3343a7c51",

        "contractName": "RollDice",

        "bytecode": "",

        "linkReferences": {},

        "from": "account{0}"

      }

    },

    {

      "timestamp": 1667155736395,

      "record": {

        "value": "0",

        "inputs": "(string)",

        "parameters": [

          "sadashjsd"

        ],

        "name": "join",

        "type": "function",

        "to": "created{1667155705600}",

        "abi": "0x68422588f8defd750baa1a778970a5caf2426fe30d05162a61bfa1c3343a7c51",

        "from": "account{0}"

      }

    },

    {

      "timestamp": 1667155755811,

      "record": {

        "value": "5000000000000000000",

        "inputs": "()",

        "parameters": [],

        "name": "deposit",

        "type": "function",

        "to": "created{1667155705600}",

        "abi": "0x68422588f8defd750baa1a778970a5caf2426fe30d05162a61bfa1c3343a7c51",

        "from": "account{0}"

      }

    },

    {

      "timestamp": 1667155777234,

      "record": {

        "value": "0",

        "inputs": "()",

        "parameters": [],

        "name": "Rolling",

        "type": "function",

        "to": "created{1667155705600}",

        "abi": "0x68422588f8defd750baa1a778970a5caf2426fe30d05162a61bfa1c3343a7c51",

        "from": "account{0}"

      }

    },

    {

      "timestamp": 1667155940292,

      "record": {

        "value": "0",

        "inputs": "(string)",

        "parameters": [

          "jkalsdasd"

        ],

        "name": "New",

        "type": "function",

        "to": "created{1667155705600}",

        "abi": "0x68422588f8defd750baa1a778970a5caf2426fe30d05162a61bfa1c3343a7c51",

        "from": "account{0}"

      }

    },

    {

      "timestamp": 1667155960364,

      "record": {

        "value": "5000000000000000000",

        "inputs": "()",

        "parameters": [],

        "name": "deposit",

        "type": "function",

        "to": "created{1667155705600}",

        "abi": "0x68422588f8defd750baa1a778970a5caf2426fe30d05162a61bfa1c3343a7c51",

        "from": "account{0}"

      }

    },

    {

      "timestamp": 1667156110536,

      "record": {

        "value": "0",

        "inputs": "()",

        "parameters": [],

        "name": "withdraw",

        "type": "function",

        "to": "created{1667155705600}",

        "abi": "0x68422588f8defd750baa1a778970a5caf2426fe30d05162a61bfa1c3343a7c51",

        "from": "account{0}"

      }

    }

  ],

  "abis": {

    "0x68422588f8defd750baa1a778970a5caf2426fe30d05162a61bfa1c3343a7c51": [

      {

        "inputs": [],

        "name": "deposit",

        "outputs": [],

        "stateMutability": "payable",

        "type": "function"

      },

      {

        "inputs": [],

        "name": "EndGame",

        "outputs": [],

        "stateMutability": "nonpayable",

        "type": "function"

      },

      {

        "inputs": [

          {

            "internalType": "string",

            "name": "seed",

            "type": "string"

          }

        ],

        "name": "join",

        "outputs": [],

        "stateMutability": "nonpayable",

        "type": "function"

      },

      {

        "inputs": [

          {

            "internalType": "string",

            "name": "seed",

            "type": "string"

          }

        ],

        "name": "New",

        "outputs": [],

        "stateMutability": "nonpayable",

        "type": "function"

      },

      {

        "inputs": [],

        "name": "Rolling",

        "outputs": [],

        "stateMutability": "nonpayable",

        "type": "function"

      },

      {

        "inputs": [],

        "name": "withdraw",

        "outputs": [],

        "stateMutability": "nonpayable",

        "type": "function"

      },

      {

        "inputs": [],

        "name": "Dice",

        "outputs": [

          {

            "internalType": "uint256",

            "name": "",

            "type": "uint256"

          }

        ],

        "stateMutability": "view",

        "type": "function"

      },

      {

        "inputs": [],

        "name": "show\_A\_balance",

        "outputs": [

          {

            "internalType": "uint256",

            "name": "",

            "type": "uint256"

          }

        ],

        "stateMutability": "view",

        "type": "function"

      },

      {

        "inputs": [],

        "name": "show\_B\_balance",

        "outputs": [

          {

            "internalType": "uint256",

            "name": "",

            "type": "uint256"

          }

        ],

        "stateMutability": "view",

        "type": "function"

      },

      {

        "inputs": [],

        "name": "show\_player\_number",

        "outputs": [

          {

            "internalType": "uint256",

            "name": "",

            "type": "uint256"

          }

        ],

        "stateMutability": "view",

        "type": "function"

      }

    ]

  }

}

**Contract Code**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated