

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 keccak-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.

```

uint256 public Dice;
uint256 Rollcount;
uint256 player_number;
bool game_end=false;

// record the address which already rolled the dice'
address lastrolled;
struct Player{
    //player's address
    address player_address;

    // player's balance
    uint256 player_balance;

    // check if the player join the game
    bool if_join;

    // player's seed
    string player_seed;

    // the time when the player join
    uint256 join_time;
}

```

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.

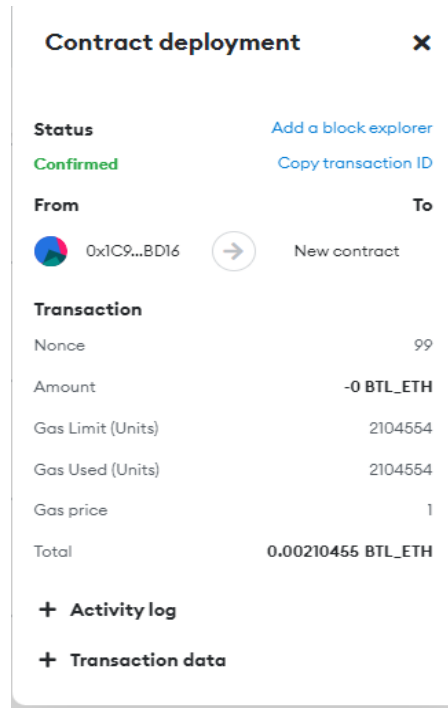


Figure 2: Gas Evaluation of Contract Deployment

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.

```

function show_A_balance() public view returns (uint256) 4625 gas
{
    if (playerA.if_join == false)
    {
        return 0;
    }
    else
    {
        return playerA.player_balance;
    }
}

// Display the player B's balance
function show_B_balance() public view returns (uint256) 4624 gas
{
    if (playerB.if_join == false)
    {
        return 0;
    }
    else
    {
        return playerA.player_balance;
    }
}

function show_player_number() public view returns(uint256) 2460 gas
{
    return player_number;
}

```

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.

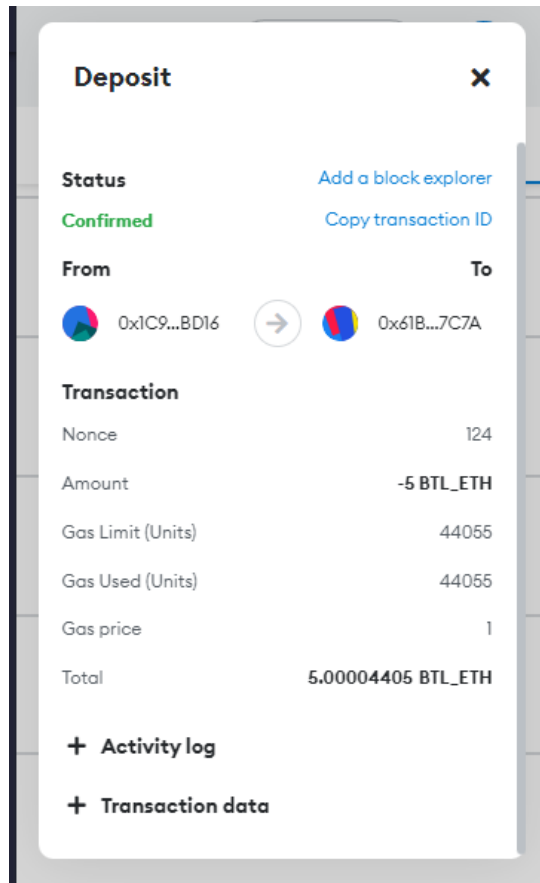


Figure 4: Gas Evaluation of **Deposit()**

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.

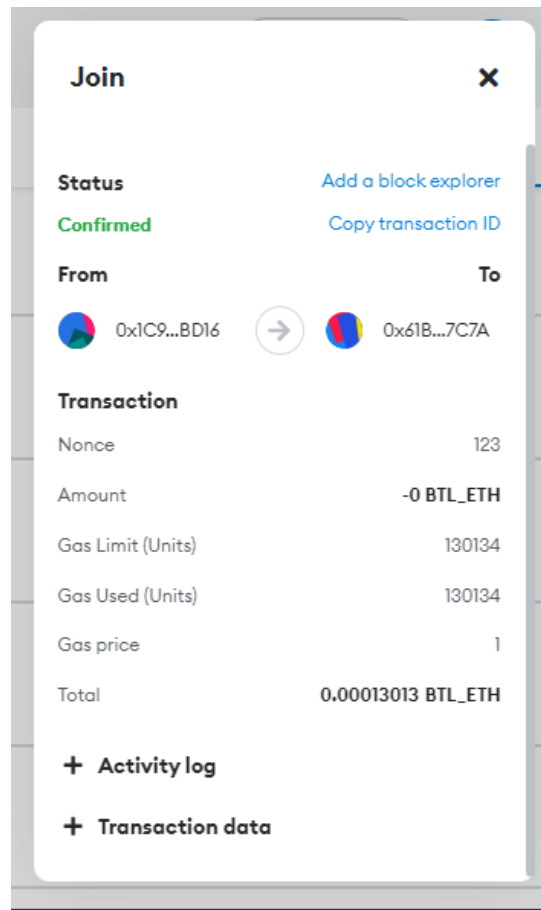


Figure 5: Gas Evaluation of **Join ()**

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.

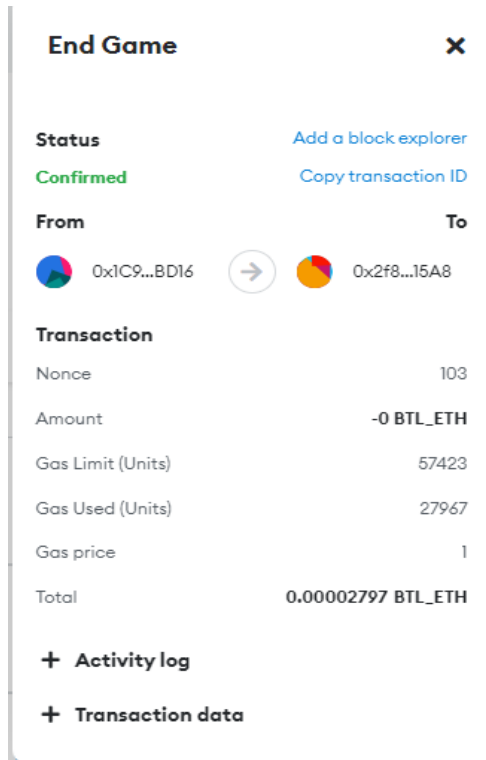


Figure 6: Gas Evaluation of **EndGame()**

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.

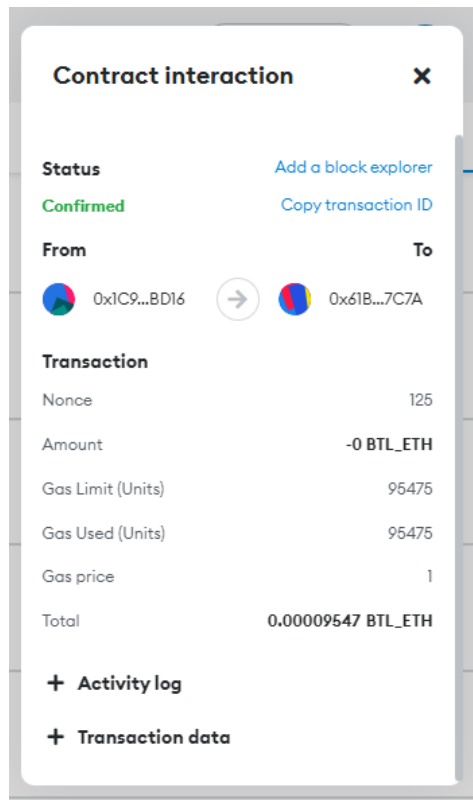


Figure 7: Gas Evaluation of **Rolling()**

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()** function can 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.


```

function Rolling () public {
    // check if the player is in the game
    require(playerA.player_address == msg.sender || playerB.player_address == msg.sender, "You are not in the game!!!");

    // check if player A and player B are both in the game
    require(playerA.if_join == true && playerB.if_join == true, "There should be two players in the game!!!");

    //check if both of the players' accounts have enough money.
    require(playerA.player_balance >= 5 && playerB.player_balance >= 5, "The account money is not enough!!!");

    // if the current game doesn't end, then it should not be rolled
    require(Rollcount == 0, "One rolling in one single game !!!");

    if (msg.sender == playerA.player_address)
    {
        //generate the random number in the range from(1,6)
        Dice = uint(keccak256(abi.encodePacked(playerA.join_time, playerA.player_seed, playerB.player_address))) % 6;
        Dice = Dice + 1;
        Rollcount = Rollcount + 1;
        Lastrolled = playerA.player_address;
    }
    else
    {
        Dice = uint(keccak256(abi.encodePacked(playerB.join_time, playerB.player_seed, playerA.player_address))) % 6;
        Dice = Dice + 1;
        Rollcount = Rollcount + 1;
        Lastrolled = playerB.player_address;
    }
}

```

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.

2. 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.

3. 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.


```
function genDiceNumber() private view returns (uint256) {  infinite gas
    require(seedA!=0 && seedB!=0);
    return (uint256(keccak256(abi.encodePacked(seedA,seedB,block.timestamp))) % 6)+1;
}
```

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": "400000000000000000",
      "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":
"60806040526000600360006101000a81548160ff02191690831515021790555034801561002b57600080fd5b506123ca8061003b6000396000f3fe6080604052600436106100915760003560e01
```

[illegible]

65b5060006001819055506001600360006101000a81548160ff021916908315150217905550505
050565b60005481565b60028054111561073b576040517f08c379a0000000000000000000000
000815260040161073290611bf7565b60405180910390fd5
b3373fff16600460000160009054906101000a900
473fff1673fffffffffffffffffffffffffffffffffffff
fffffffffff16141580156107ed57503373fff16600
960000160009054906101000a900473fff1673fff
fff1614155b61082c576040517f08c379a0000000000
000815260040161082390611c57565b604
05180910390fd5b600081511415610871576040517f08c379a00000000000000000000000000
000815260040161086890611b97565b60405180910390fd5b600
01515600460020160009054906101000a900460ff161515141561092f573360046000016000610
1000a81548173fff021916908373ffffffffffffff
fffffffffffffffffffffffffffffffff1602179055506001600460020160006101000a81548160ff021
916908315150217905550806004600301908051906020019061090b9291906116dc565b5042600
480018190555060016002546109249190611db9565b6002819055506109cb565b3360096000016
0006101000a81548173fff021916908373ffffff
fffffffffffffffffffffffffffffffff1602179055506001600960020160006101000a8154816
0ff02191690831515021790555080600960030190805190602001906109aa9291906116dc565b5
04260096004018190555060016002546109c49190611db9565b6002819055505b50565b6000801
515600960020160009054906101000a900460ff16151514156109f75760009050610a00565b600
46001015490505b90565b60011515600360009054906101000a900460ff16151514610a5957604
0517f08c379a000815260040
1610a5090611c97565b60405180910390fd5b3373fffffffffffffffffffffffffffffffffffff
fff16600460000160009054906101000a900473fffffffffffffffffffffffffffffffffffff
f1673fffffffffffffffffffffffffffffffffffff161480610b0857503373ffffffffffffff
fffffffffffffffffffffffffffff16600960000160009054906101000a900473ffffffffffffff
fffffffffffffffffffff1673fffffffffffffffffffff16145b610
b47576040517f08c379a008
152600401610b3e90611c37565b60405180910390fd5b600081511415610b8c576040517f08c37
9a0008152600401610b83906
11b97565b60405180910390fd5b6000805414610bd0576040517f08c379a0000000000000000
0008152600401610bc790611c77565b60405180910
390fd5b60008081905550600060046003018054610be990611ee8565b905014158015610c0b575
0600060096003018054610c0690611ee8565b905014155b15610c2c576000600360006101000a8
1548160ff0219169083151502179055505b600460000160009054906101000a900473ffffff
fffffffffffffffffffff1673fffffffffffffffffffff163
373fffffffffffffffffffff161415610cad57806004600301908051906
0200190610c9e9291906116dc565b50426004800181905550610cd2565b8060096003019080519
060200190610cc69291906116dc565b50426009600401819055505b50565b3373ffffff
fffffffffffff16600460000160009054906101000a900473ffffff
fffffffffffff1673fffffffffffff161480610
d8457503373fffffffffffff166009600001600090549061010
00a900473fffffffffffff1673fffffffffffff

[illegible]

00000000000815260040161144190611bb7565b60405180910390fd5b600460000160009054906
101000a900473fff1673fffffffffffffffffffffffffff
ffffffffffffffffffffffff163373fff1614156115bd5
76000600460010154905060006004600101819055506000600460020160006101000a81548160f
f021916908315150217905550600460000160009054906101000a900473fffffffffffffffffff
ffffffffffffffffffffffff1673fff166108fc82908
1150290604051600060405180830381858888f19350505050158015611541573d6000803e3d600
0fd5b506004600080820160006101000a81549073fffffffffffffffffffffffffffffffffffff
fff021916905560018201600090556002820160006101000a81549060ff0219169055600382016
0006115989190611762565b6004820160009055505060016002546115b19190611e69565b60028
1905550506116d3565b60006009600101549050600060096001018190555060006009600201600
06101000a81548160ff021916908315150217905550600960000160009054906101000a900473f
fff1673fffffffffffffffffffffffffffffffffffff
ffffff166108fc829081150290604051600060405180830381858888f1935050505015801561165
b573d6000803e3d6000fd5b506009600080820160006101000a81549073fffffffffffffffffffff
ffffffffffffffffffff021916905560018201600090556002820160006101000a81549060ff0
2191690556003820160006116b29190611762565b6004820160009055505060016002546116cb9
190611e69565b600281905550505b60008081905550565b8280546116e890611ee8565b9060005
2602060002090601f01602090048101928261170a5760008555611751565b82601f10611723578
05160ff1916838001178555611751565b82800160010185558215611751579182015b828111156
11750578251825591602001919060010190611735565b5b50905061175e91906117a2565b50905
65b50805461176e90611ee8565b6000825580601f10611780575061179f565b601f01602090049
060005260206000209081019061179e91906117a2565b5b50565b5b808211156117bb576000816
0009055506001016117a3565b5090565b60006117d26117cd84611d57565b611d32565b9050828
152602081018484840111156117ee576117ed61206b565b5b6117f9848285611ed9565b5093925
05050565b600082601f83011261181657611815612066565b5b81356118268482602086016117b
f565b91505092915050565b60006020828403121561184557611844612075565b5b60008201356
7fffffffffffffffffff81111561186357611862612070565b5b61186f84828501611801565b91505
092915050565b61188961188482611e9d565b611f4b565b82525050565b6000815461189c81611
ee8565b6118a68186611dae565b945060018216600081146118c157600181146118d2576119055
65b60ff19831686528186019350611905565b6118db85611d88565b60005b838110156118fd578
154818901526001820191506020810190506118de565b838801955050505b50505092915050565
b600061191b602483611d9d565b915061192682612098565b604082019050919050565b6000611
93e602a83611d9d565b9150611949826120e7565b604082019050919050565b600061196160208
3611d9d565b915061196c82612136565b602082019050919050565b6000611984601f83611d9d5
65b915061198f8261215f565b602082019050919050565b60006119a7602983611d9d565b91506
119b282612188565b604082019050919050565b60006119ca601283611d9d565b91506119d5826
121d7565b602082019050919050565b60006119ed602183611d9d565b91506119f882612200565
b604082019050919050565b6000611a10601b83611d9d565b9150611a1b8261224f565b6020820
19050919050565b6000611a33601683611d9d565b9150611a3e82612278565b602082019050919
050565b6000611a56601783611d9d565b9150611a61826122a1565b602082019050919050565b6
000611a79601783611d9d565b9150611a84826122ca565b602082019050919050565b6000611a9
c601a83611d9d565b9150611aa7826122f3565b602082019050919050565b6000611abf601a836
11d9d565b9150611aca8261231c565b602082019050919050565b6000611ae2602183611d9d565

b9150611aed82612345565b604082019050919050565b611b0181611ecf565b82525050565b611
b18611b1382611ecf565b611f6f565b82525050565b6000611b2a8286611b07565b60208201915
0611b3a828561188f565b9150611b468284611878565b6014820191508190509493505050565
b60006020820190508181036000830152611b708161190e565b9050919050565b6000602082019
0508181036000830152611b9081611931565b9050919050565b600060208201905081810360008
30152611bb081611954565b9050919050565b60006020820190508181036000830152611bd0816
11977565b9050919050565b60006020820190508181036000830152611bf08161199a565b90509
19050565b60006020820190508181036000830152611c10816119bd565b9050919050565b60006
020820190508181036000830152611c30816119e0565b9050919050565b6000602082019050818
1036000830152611c5081611a03565b9050919050565b600060208201905081810360008301526
11c7081611a26565b9050919050565b60006020820190508181036000830152611c9081611a495
65b9050919050565b60006020820190508181036000830152611cb081611a6c565b90509190505
65b60006020820190508181036000830152611cd081611a8f565b9050919050565b60006020820
190508181036000830152611cf081611ab2565b9050919050565b6000602082019050818103600
0830152611d1081611ad5565b9050919050565b6000602082019050611d2c6000830184611af85
65b92915050565b6000611d3c611d4d565b9050611d488282611f1a565b919050565b600060405
1905090565b600067ffffffffffffffffffff821115611d7257611d71612037565b5b611d7b8261207
a565b9050602081019050919050565b60008190508160005260206000209050919050565b60008
2825260208201905092915050565b600081905092915050565b6000611dc482611ecf565b91506
11dcf83611ecf565b9250827fff
fffffffffffff03821115611e0457611e03611faa565b5b828201905092915050565b6000611e1a8
2611ecf565b9150611e2583611ecf565b9250817fff
fffffffffffffffffffffffffffff0483118215151615611e5e57611e5d611faa565b5b828202905
092915050565b6000611e7482611ecf565b9150611e7f83611ecf565b925082821015611e92576
11e91611faa565b5b828203905092915050565b6000611ea882611eaf565b9050919050565b600
073fffffffffffffffffffffffffffffffffffff82169050919050565b60008190509190505
65b828183376000838301525050565b60006002820490506001821680611f0057607f8216915
05b60208210811415611f1457611f13612008565b5b50919050565b611f238261207a565b81018
1811067ffffffffffffffffffff82111715611f4257611f41612037565b5b80604052505050565b600
0611f5682611f5d565b9050919050565b6000611f688261208b565b9050919050565b600081905
0919050565b6000611f8482611ecf565b9150611f8f83611ecf565b925082611f9f57611f9e611
fd9565b5b828206905092915050565b7f4e487b71000000000000000000000000000000000000
000
000
000
b7f4e487b7100
0045260246000fd5b600080fd5b600080fd5b600080fd5b600080fd5b6000601f19601f8301169
050919050565b60008160601b9050919050565b7f506c65617365206465706f736974206174206
c6561737420666976652065746860008201527f657221210000000000000000000000000000000000
000
8657220706c61796572207769746860008201527f64726177206d6f6e6579000000000000000000
000
e652073696e676c65206c6574746572600082015250565b7f4174206c6561737420696e707574206f6
e652073696e676c65206c6574746572600082015250565b7f746865206d6f6e6579206861736e2
774206265656e2077697468647261776e00600082015250565b7f54686572652073686f756c642

[illegible]

```

        "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

```
pragma solidity >=0.7.0 <0.9.0;
contract RollDice{
    uint256 public Dice;
    uint256 Rollcount;
    uint256 player_number;
    bool game_end=false;

    // record the address which already rolled the dice'
    address lastrolled;
    struct Player{
        //player's address
        address player_address;

        // player's balance
        uint256 player_balance;

        // check if the player join the game
        bool if_join;

        // player's seed
        string player_seed;

        // the time when the player join
        uint256 join_time;
    }

    Player playerA;
    Player playerB;

    function deposit () public payable{  Infinite gas
        // check if the sender is the player in the game
        require(playerA.player_address == msg.sender || playerB.player_address == msg.sender, "You are not in the game!!!" );

        // For each game, either player A or player B needs to deposit at least 5 ether in the contract account.
        require(msg.value >=5 ether , "Please deposit at least five ether!!!");

        if(msg.sender == playerA.player_address){
            playerA.player_balance = playerA.player_balance + msg.value;
        }
        else{
            playerB.player_balance = playerB.player_balance + msg.value;
        }
    }
}
```

```
function EndGame() public  Infinite gas
{
    require(playerA.player_address== msg.sender || playerB.player_address == msg.sender, "You are not in the game !!!");
    require(game_end==true,"the money hasn't been withdrawn");
    // playerA left
    if(msg.sender == playerA.player_address){
        uint256 reward_A = playerA.player_balance;
        playerA.player_balance=0;
        playerA.if_join=false;
        payable(playerA.player_address).transfer(reward_A);
        delete playerA;
        player_number=player_number-1;
    }
    // playerB left
    else
    {
        uint256 reward_B = playerB.player_balance;
        playerB.player_balance=0;
        playerB.if_join=false;
        payable(playerB.player_address).transfer(reward_B);
        delete playerB;
        player_number=player_number-1;
    }
    Dice = 0;
}

// Each player needs to input the new player's seed to start the new game
function New(string memory seed) public  Infinite gas
{
    require(game_end==true,"The game doesn't end!!!");
    require(playerA.player_address== msg.sender || playerB.player_address == msg.sender, "You are not in the game !!!");
    require(bytes(seed).length != 0, "At least input one single letter");
    require(Dice==0,"The dice is not reset!!!");
    Dice =0;
    if (bytes(playerA.player_seed).length != 0 && bytes(playerB.player_seed).length != 0 )
    {
        game_end=false;
    }
    if(msg.sender == playerA.player_address)
    {
        playerA.player_seed= seed;
        playerA.join_time=block.timestamp;
    }
}
```

```

    else
    {
        playerB.player_seed = seed;
        playerB.join_time=block.timestamp;
    }
}
function join (string memory seed) public payable infinite gas
{
    // Check if player A's address or player B's address is the same as the sender's address, if they are both not equal to that
    // It means both A and B are already in the game,there is no spot
    require(player_number<=2,"There are no spots");
    require(playerA.player_address != msg.sender && playerB.player_address != msg.sender, "You can not join twice");

    // require the player to input something to set the seed
    require(bytes(seed).length != 0,"At least input one single letter");

    // if the spot A is empty, then the first player who joins the game is the player A
    if (playerA.if_join==false)
    {
        // set the address;
        playerA.player_address = msg.sender;

        // Now player is in the game
        playerA.if_join = true;

        // set the seed
        playerA.player_seed = seed;

        //set the time when the player join the game
        playerA.join_time=block.timestamp;
        player_number=player_number+1;
    }
    else
    {
        // set the address;
        playerB.player_address = msg.sender;

        // Now player is in the game
        playerB.if_join = true;

        // set the seed
        playerB.player_seed = seed;
    }
}

```

```

function Rolling () public payable infinite gas
{
    // check if the player is in the game
    require(playerA.player_address == msg.sender || playerB.player_address == msg.sender, "You are not in the game!!!");

    // check if player A and player B are both in the game
    require(playerA.if_join ==true && playerB.if_join ==true, "There should be two players in the game!!");

    //check if both of the players' accounts have enough money.
    require(playerA.player_balance>=5 && playerB.player_balance>=5,"The account money is not enough!!");

    // if the current game doesn't end, then it should not be rolled
    require(Rollcount==0,"One rolling in one sigle game !!!");

    if (msg.sender==playerA.player_address)
    {
        //generate the random number in the range from(1,6)
        Dice=uint(keccak256(abi.encodePacked(playerA.join_time,playerA.player_seed,playerB.player_address))) % 6;
        Dice=Dice+1;
        Rollcount=Rollcount+1;
        Lastrolled=playerA.player_address;
    }
    else
    {
        Dice=uint(keccak256(abi.encodePacked(playerB.join_time,playerB.player_seed,playerA.player_address))) % 6;
        Dice=Dice+1;
        Rollcount=Rollcount+1;
        Lastrolled=playerB.player_address;
    }
}
// Display the player's balance

```

```

// Display the player A's balance
function show_A_balance() public view returns (uint256) 4625 gas
{
    if (playerA.if_join == false)
    {
        return 0;
    }
    else
    {
        return playerA.player_balance;
    }
}

// Display the player B's balance
function show_B_balance() public view returns (uint256) 4624 gas
{
    if (playerB.if_join == false)
    {
        return 0;
    }
    else
    {
        return playerA.player_balance;
    }
}

function show_player_number() public view returns(uint256) 2460 gas
{
    return player_number;
}

// This function will be used for these players to withdraw money from the contract account
function withdraw() public infinite gas
{
    // First check if the dicenumber, make sure that the dice is already rolled.
    require(Dice != 0, "Please roll the dice first");
    // Make sure the player who wants to withdraw the money is the player in the game
    require(playerA.player_address == msg.sender || playerB.player_address == msg.sender, "You are not in the game !!!");

    // The player who has already rolled dice can not withdraw and it another palyer's turn to withdraw the money in order to save the gas fee by reducing
    // the number of operations.
    require(msg.sender != Lastrolled, "Please let the other player withdraw money");
    // dice number is regarded as 1~6 Wei in the game, so if the player wants withdraw money in units of ether, it has to multiply 10^18
    // To avoid the reentrancy attack, before using the dice number, dice number is set as zero, so a temp variable is initialised to store the Dice number.
    uint256 temp = Dice * 1000000000000000000;

```

```

uint256 temp_A_balance = playerA.player_balance;
uint256 temp_B_balance = playerB.player_balance;
//To avoid reentrancy attack, the dice number is set as zero, and all of the players' balance are set as zero to prevented being called back
playerA.player_balance = 0;
playerB.player_balance = 0;
Dice = 0;

//Dice: 1~3
if(temp >= 1000000000000000000 && temp <= 3000000000000000000){
    uint256 reward_A = temp + temp_A_balance;
    uint256 reward_B = temp_B_balance - temp;

    // In this case, player A win this game and win money of dicenumber and get his original money back
    payable(playerA.player_address).transfer(reward_A);

    // For player B, he lost this game and lost the money that A wins and get the rest of his money back
    payable(playerB.player_address).transfer(reward_B);
}
// Dice:4~6
else{
    uint256 reward_A = temp_A_balance + 3000000000000000000 - temp;
    uint256 reward_B = temp_B_balance + temp - 3000000000000000000;
    payable(playerA.player_address).transfer(reward_A);
    payable(playerB.player_address).transfer(reward_B);
}
// After each game, each player's seed is set as empty.
playerA.player_seed = "";
playerB.player_seed = "";
Rollcount=0;
game_end=true;
}
}

```