Assignment 2

1. High-Level decision making

The smart contract implements the rolling the dice game defined in the coursework document. In this high-level decision-making part, I will explain how to play the game with the following instructions and then in the second part of this report, I will explain why I made these choices.

- Both players click the register button to register to the game and deposit numbers of ETH (needed to be larger than 3)
- After two players have all finished register, one of the players click the startrolling button, then the contract will generate a number between 1-6.
- Then, when the button finishgame is clicked, the contract will show who wins the game and update their balance.
- Then the player should click withdrawReward button, all their balance will go to their account and variables will restore to default.

1.1 Registration

In this section both players register to play the game and send ETH amount which should be larger than 3 to the contract: the game is defined such that the winner gets N ETH (when winner is player A), N-3 ETH (when winner is player B). I decided that the registered amount should be larger than 3 is because whoever win the dice game will get amount from 1 ETH to 3 ETH, and therefore, the register process should deposit more than 3 ETH to pay to the winner. Also, I added a requirement that there should only be two players in the contract, that is because if there are more than three players in the contract, the contract will never work with the third person' s balance and address. Another function that I added in this part is no duplicate player is allowed which means if player A has registered in the contract, then he cannot register to the contract again because if one address can register for both two players, the address is playing with itself which makes the dice game meaningless.

1.2 Gaming Process and Result

After finishing registering for player A and player B, either player click startRoll(), then the function will check if these two players 'balance is larger than 3 that can pay to the winner, then the function will decide which player wins and stored in the contract. Finally, either player can click finishgame() button, and all stored variables will become default, and winner and loser' s balance will be updated and then they should

click withdrawReward button to send the amount of ETH in balance to their account. If two new players want to continue play the game, they need to both click the register button as I shown above and play the dice game.

2. Detailed description of Questions

- **Who pays for the reward of the winner?**
  The reward of the winner is paid by the loser of the game. Before start rolling, both players should confirm that their balance have enough balance to pay for the lost (larger than 3), if their balance is not enough, the game will not start until they clicked deposit button and deposited enough ETH to their balance. After playing the game, the smart contract will update the mapping balances which will send money to the winner and take the same amount of money from the loser. To conclude, loser will pay for the reward of the winner.

- **How is the reward sent to the winner?**
  After finishing the game button is clicked, the contract will update players' mapping balances and then automatically send the balance to their account. After all these processes above, the address, balance and player list will be restored to default and ready for the next two players.

- **How is it guaranteed that a player cannot cheat?**
  At first, I did not set the rolling dice time limitations and that will make players rolling dice as much as they want until they get the result they wanted, but I realize this is not how the game should work, then I added a Boolean variable called rolldone which will become true if the function has already finished rolling the dice; if the rolling has done, then either players should not be able to reroll the dice again and have to click finish game to continue the game. Second, I started getting the random value of the dice by using keccak256 function to hash the block.timestamp, msg.sender, but I realize it will make players who get the contract code knowing how it works and calculate the dice number, so I added a variable called nonce, nonce will increased by 1 each game is played, because people will never know the nonce number.

- **What data type/structures did you use and why?**
  I used mapping from address to uint256 to make play' s address become key and they balance become value, this structure will help me easily to update their

balance amount when each player is depositing or withdrawing with their address. I also used a mapping from address to Boolean to check if the given address has already registered into the game or not, this structure is used to avoid that one address can register for two players. Also, I have a private address list called players to store two player' s address.

3. Gas evaluation

3.1 The cost of deploying and interacting with your contract

The cost of deploying:

Transaction hash:

0x8cda644eaa5dcc00f0610f02763232aebe3396229c45898cf7696bc5cef26268

Gas cost:

1650695 GWEI

Deployed Contract address:

0x80b115ED656F0Ec699de6b38036d07b778F89aC6

There are few buttons that players can interact with the smart contact, using the register button will cost about 106844 Gwei, here the gas fee is quite high is because I also combined deposit function in register function which means people should have an initial deposit when they do register. Using the deposit button will cost about 28819 Gwei, this function is designed when players have not enough balance to start the game. Using withdraw button will cost about 32953 Gwei. Using startrolling button will cost about 135587 Gwei, using finishgame button will cost about 69005 Gwei, using the viewblance button will cost about 23635 Gwei, using timefortimertoend button will cost 24061 Gwei, using withdrawReward button will cost about 73342 Gwei

3.2 Fairness of the contract

The contract is pretty much fair in the sense that both players must follow the same process to play the game. Both players need to pay similar fees for registering, and deposit. However, there are two actions that may not be completely fair: finishing the game and starting the game. The problem is that at least one player must click the starting the game button and finishing the game button which means that player will pay more fees when click the buttons.

3.3 Techniques to make contract more fair

I changed data type of timeout from uint to uint32 because timestamps do not need to be more than uint48 which will make the contract more efficient, also by using the

struct date type in the contract may cost less. For the fairness part, my main idea is to separate costly part in the startrolling and finishgame functions into two parts that allow two player both click them and only one player pay for the combined part. Another possible way to make fairness of gas is to let player 1 click the startgaming button and let player 2 click finishgame button and withdrawReward button.

4. Possible hazards and vulnerabilities

4.1 Griefing

A smart contract will check for the failure of success of send()/transfer(), a send function can fail when there is not enough bas in the account to execute the default function in the smart contract. Therefore, a griefing wallet can have a default function that takes a large amount of gas to run. This wallet will lock the smart contact by engaging with them so when the contact use send()/transfer(), they fail and get disrupted. But this will not happen in my contract because by applying the "pull or push" principle, I make the contract not to send the money to the players, but the players are the one that will do withdraw from the contract.

4.2 Reentrancy

A reentrancy attack can happen when the smart contact has a function that calls an external source before reset the contract, for example, when an attacker controls the contract, they can make a recursive call to the original function, then do the interactions that will have otherwise not run after the effects were resolved. But this will not happened in my contract because I finished all internal work and then call external functions, and my reset() function will resetting the values of players when the game is finished.

4.3 Randomness

In the contract, block information can be manipulated by miner, and block information shared by all users in the same block, if same-blocks txs share randomness source, attacker can check whether conditions are favorable before acting. Therefore, we should not do the hash by only applying vairables like block.number, block.coinbase, block.timestamp, block.gasLimit, block.hash, now, block.difficulty and msg.sender. In my contract, I added one variable called nonce, and each time roll() runction is called, nonce will plus one, and that will kindly prevent it because it is hard to know how many time the roll() function has been called.

4.4 Other hazards

Front-running, transactions take some time before they are mined. An attacker can

watch the transaction pool and send a transaction, have it included in a block before the original transaction. This mechanism can be abused to re-order transactions to the attacker’ s advantage, here in the contact, I did not apply commitment scheme because there is nothing two players need to submit to the contract, the dice rolling is inside the contract, and we use players address to register the game.

5. Trade off

I made several trade-offs in the smart contract, I did not use any commitments and having the contract computing the winner and waiting for the players to withdraw their rewards, that is because using the commitment will increase the cost of the contract and the registering process in my contract only used user address to do it, but it may still face the front running hazard at some point. Another trade-off I made in the contract is that I created two different withdraw function, one used for withdrawing money before the game and another for withdraw reward, although using an extra function in the contract with similar functionality is very costly, this can avoid player withdraw their money while the dice rolling process is done.

6. Fellow student’ s contract

The code of my partner’ s code is quite different from mine, he used a struct datatype to contain all the variable in the contract like dice number, winner address and time stamp, I think the using of this data type can significantly decrease the gas value. Also, he applied a function called waiting list, which will first send all account address to the waiting list and then pop two of them each time, this is a great idea because the third person can register to the contract before the two players are finished playing the game. However, I discovered one potential hazard in his contract: in the rolldice function, he hashed roundNonce (which initialized to 0 and will plus one each time the rolldice function is called), block.difficulty, and players address, it seems that the roundNonce number will never be found by the players, however, he also implemented a list called winnerrecord to record each round winner, therefore, as each round the dice can only be rolled once, players will know the value of roundNonce by counting the length of winnerRecord list. Also, there is a unbounded problem in his for loop, because the operation requires more gas as array becomes larger and after some point, it might be impossible to execute it (beyond gas limits), some attacker may use this feature to block the contract.

7. Transaction history

```
{
  "accounts": {
    "account{0}": "0x41479D33770A1f102493d97e8fD5c897896784a5"
  },
  "linkReferences": {},
  "transactions": [
    {
      "timestamp": 1667167883929,
      "record": {
        "value": "0",
        "inputs": "()",
        "parameters": [],
        "name": "",
        "type": "constructor",
        "abi":
"0x85b639f921ae3230afc3d0821d74e7b52e9cfc3a34ea2b07a563e7d04675414d",
        "contractName": "Dice",
        "bytecode":
"60806040523480156100105760080fd5b50611d0980610020600396000f3fe60806040526
004361061007b5760003560e01c80638f6e33521161004e5780638f6e3352146100f75780639
a6d79191461010157806c885bc581461012c578063d0e30db014610143576100b565b8063
1aa3a008146100805780633ccfd60b1461008a5780633ff1e05b146100a157806344349b0114
6100cc575b600080fd5b61008861014d565b005b348015610096576100a0fd5b5061009f6103
4e565b005b3480156100ad57600080fd5b506100b66107f2565b6040516100c391906117ea5
65b60405180910390f35b3480156100d857600080fd5b506100e1610838565b6040516100ee
91906117ea565b60405180910390f35b6100ff610893565b005b34801561010d57600080fd5b
50610116610b05565b604051610123919061116e8565b60405180910390f35b3480156101385
7600080fd5b50610141610c80565b005b61014b610ff8565b005b6729a2241af62c000034116
1016157600080fd5b60016002805490501115610101aa576040517f08c379a0000000000000000
0000000000000000000000000000000000000000000000081526004016101a19061174a565b604051
80910390fd5b60001515600160003373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffff
ffffffffff16815260200190815260200160002060009054906101000a900460ff1615151461023d
576040517f08c379a0000000000000000000000000000000000000000000000000000000008152
6004016102349061170a565b60405180910390fd5b346000803373ffffffffffffffffffffffffffffffffffffffff
ffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060000828254610289b
9190611821565b9250508190555060023390806001815401808255809150506001900390600
052602060002001600090919091909161010000a81548173ffffffffffffffffffffffffffffffffffffffff021916
908373ffffffffffffffffffffffffffffffffffffffff160217905550600180600003373ffffffffffffffffffffffffffffffffffffffff
fff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060006101000a81548
```

160ff021916908315150217905550565b600015156007600090549061010 00a900460ff161515
148015610384575060001515600360019054906101000a900460ff161515145b1561045d5760
008060003373fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff16815260200190
815260200160002054905060008060003373fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff16815260200190815260200160002081905550337 3fffffffffffffffffffffffffffffffffffffff166108fc829081150290604051600060405180830381858888f193505050501580156107ad573d6000803e3d6000fd5b50506107ef565b6040517f82b42900000000000000000000000000000000000000000000000000000000081526004016040518091039 0fd5b5b565b60008060003373fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054905090565b60008060085414610884576000426101 2c63ffffffff1660085461085c9190611821565b61086691906118d1565b90506000811015610 87b57600091505061089056 5b809150506108905 65b61012c63ffffffff1690505b90565b60036000806002600815481106108ad576108ac611aa9565b5b9060005260206000200160009054906101000a900473fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054905060008060006002600815481106105b2576105b1611aa9565b5b9060005260206000200160009054906101000a900473fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff166108fc829081150290604051600060405180830381858888f193505050501580156107ad573d6000803e3d6000fd5b50506107ef565b6040517f82b42900000000000000000000000000000000000000000000000000000000081526004016040518091039 0fd5b5b565b60008060003373fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054905060008060006002600815481106106bf576106be611aa9565b5b9060005260206000200160009054906101000a900473fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff16815260200190815260200160002081905550337 3fffffffffffffffffffffffffffffffffffffff166108fc829081150290604051600060405180830381858888f193505050501580156107ad573d6000803e3d6000fd5b506107b66110e2565b50506107ef565b6040517f82b42900000000000000000000000000000000000000000000000000000000081526004016040518091039 0fd5b5b565b60008060003373fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054905090565b60008060085414610884576000426101 2c63ffffffff1660085461085c9190611821565b61086691906118d1565b90506000811015610 87b57600091505061089056 5b809150506108905 65b61012c63ffffffff1690505b90565b60036000806002600815481106108ad576108ac611aa9565b5b9060005260206000200160009054906101000a900473fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054116109535760405 17f08c379a0000000000000000000000000000000000000000000000000000000081526004016109 4a9061178a565b6040518091039 0fd5b6003600080600260018154811061096d5761096c611aa9565b5b9060005260206000200160009054906101000a900473fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054116 10a13576040517f08c379a0000000000000000000000000000000000000000000000000000000000000000000000000000000

8152600401610a0a9061176a565b60405180910390fd5b60001515600360019054906101000
a900460ff16151514610a69576040517f08c379a0000000000000000000000000000000000000
0000000000000000000008152600401610a609061172a565b60405180910390fd5b610a71611
2de565b60058190555060016003600161010000a81548160ff021916908315150217905550600
46005541015610ac65760016003600061010000a81548160ff021916908315150217905550600
55460048190555061 0b03565b60036005541115610b025760006003600061010000a81548160
ff021916908315150217905550600360055461 0afb91906118d1565b6004819055505b5b565b
606061012c63ffffffff16600854610b1d9190611821565b421180610b3d575060011515600360
019054906101000a900460ff161515145b610b4657600080fd5b60006002600081548110610b
5d57610b5c611aa9565b5b90600052602060002001600090549061010000a900473ffffffffffffffff
ffffffffffffffffffffffff1690506000600260018154811061 0ba157610ba0611aa9565b5b906000526
020600020016000905490610100 0a900473ffffffffffffffffffffffffffffffffffffffff169050610bd882826
11355565b60016007600061010000a81548160ff021916908315150217905550600360009054 9
06101000a900460ff16610c42576040518060400160405280600b8152602001 7f506c6179657
23220776f6e000000000000000000000000000000000000000000000815250610c79565b604051
806040016040528060 0b81526020017f506c617965723120776f6e000000000000000000000000
0000000000000000000008152505b9250505090565b60011515600760009054906101000a900
460ff16151514610cd6576040517f08c379a0000000000000000000000000000000000000000000
0000000000000000815260040161 0ccd906117ca565b60405180910390fd5b600080600060
026000815481 10610cf057610cef611aa9565b5b90600052602060002001600090549061010 0
0a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffff
ffffffffffffff168152602001908152602001600020549050600080600060026000815481 10610d7
257610d71611aa9565b5b90600052602060002001600090549061010 0a900473ffffffffffffffffff
ffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020
019081526020016000208190555060026000815481 10610df057610def611aa9565b5b906000
526020600020016000905490610100 0a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffff
ffffffffffffffffffff166108fc829081150290604051600060405180830381858888f1935050505015
8015610e60573d6000803e3d6000fd5b50600080600060026001815481 10610e7b57610e7a6
11aa9565b5b90600052602060002001600090549061010 0a900473ffffffffffffffffffffffffffffffffffffffff
ffff1673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602
0016000205490506000806000600260018154811061 0efd57610efc611aa9565b5b90600052
602060002001600090549061010 0a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffff
ffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020819055506
002600181548110610f7b57610f7a611aa9565b5b90600052602060002001600090549061010
00a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff166108fc8290811502
906040516000604051808303818588 8f193505050501580 15610feb573d6000803e3d6000fd
5b50610ff46110e2565b5050565b600115156001600033 73ffffffffffffffffffffffffffffffffffffff1673fff
fffffffffffffffffffffffffffffffff168152602001908152602001600020600090549061010 0a900460f
f1615151461108b576040517f08c379a0000000000000000000000000000000000000000000000000

0000000000000081526004016110829061 17aa565b60405180910390fd5b346000803373ffffffff
ffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081526020016000206
0008282546110d99190611821565b92505081905550565b600060036001610100 0a81548160
ff02191690831515021790555060006007600061010 00a81548160ff0219169083151502179 05
550600060088190555060006001600060026000815481106 1113b5761113a611aa9565b5b90
600052602060002001600090549061010 00a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffff
ffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020600
06101000a81548160ff0219169083151502179055506000600160006002600181548110 6111d
2576111d1611aa9565b5b906000526020600020016000905490610100 0a900473ffffffffffffffffff
ffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020
0190815260200160002060006101000a81548160ff021916908315150217905550600280548 0
6112605761125f611a7a565b5b600190038181906000526020600020016000610 1000a81549
073ffffffffffffffffffffffffffffffffffffffff021916905590556002805480 6112a7576112a6611a7a565b5b
600190038181906000526020600020016000610 1000a81549073ffffffffffffffffffffffffffffffffffffffff0
2191690559055565b6000426008819055506000600642336006546040516020016113029392
91906116ab565b604051602081830303815290604052805190602001206000 1c61132591906
119eb565b9050600181611334919061182156 5b905060066000815480929190611349906119
74565b919050555080915050905 65b6001151560036000905490610100 0a900460ff16151514
1561144a57670de0b6b3a76400006004546113879190611877565b6000808473fffffffffffffffffff
fffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060008 28
2546113d4919061182156 5b92505081905550670de0b6b3a764000060045461 13f191906118
77565b6000808373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020
0190815260200160002060008282546114 3e91906118d1565b9250508190555061153c565b6
000151560036000905490610100 0a900460ff161515141561153b57670de0b6b3a7640000600
45461147c9190611877565b6000808373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffff
ffffffffffff168152602001908152602001600020600082825461 14c99190611821565b92505081
905550670de0b6b3a764000060045461 14e69190611877565b6000808473fffffffffffffffffffffffffff
fffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081526020016000206000828254 6
115339190611 8d1565b92505081905550 5b5b5050565b61155161154c82611905565b6119b
d565b82525050565b60006115628261180556 5b61156c8185611810565b935061157c818560
20860161194156 5b61158581611ad8565b840191505092915050565b600061159d601c83611
810565b91506115a882611af6565b602082019050919050565b60006115c0602d836118 1056
5b91506115cb82611b1f565b604082019050919050565b60006115e3603483611810565b915
06115ee82611b6e565b604082019050919050565b6000611606602683611810565b91506116
1182611bbd565b604082019050919050565b600061162960268361181056 5b915061163482
611c0c565b604082019050919050565b600061164c601a83611810565b9150611657 82611c5
b565b602082019050919050565b600061166f6031836118 10565b915061167a82611c84565b
604082019050919050565b61168e816119375 65b82525050565b6116a56116a0826119375 65
b6119e1565b82525050565b60006116b7828661169456 5b602082019150611 6c78285611540

565b6014820191506116d78284611694565b602082019150819050949350505050565b60006
020820190508181036000830152611702818461155756 5b905092915050565b600060208201
905081810360008301526117238161159056 5b9050919050565b60006020820190508181036
000830152611743816115b3565b9050919050565b60006020820190508181036000830152611
1763816115d6565b9050919050565b60006020820190508181036000830152611783816115f
9565b9050919050565b60006020820190508181036000830152611 7a38161161c565b905091
9050565b60006020820190508181036000830152611 7c38161163f565b9050919050565b600
060208201905081810360008301526117e381611662565b9050919050565b60006020820190
506117ff6000830184611685565b92915050565b600081519050919050565b60008282526020
820190509291505 0565b600061182c82611937565b9150611837836119375 65b9250827fffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffff0382111561186c5761186b611a1c565b5b828201905
092915050565b600061188282611937565b915061188d83611937565b9250817fffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffff0483118215151615611 8c6576118c5611a1c565b5b8282029
05092915050565b60006118dc82611937565b91506118e783611937565b9250828210156118
fa576118f9611a1c565b5b828203905092915050565b600061191082611917565b9050919050
565b600073ffffffffffffffffffffffffffffffffffffffffffff8216905091905056 5b600081905091905056 5b6000
5b8381101561195f57808201518184015260208101905061194456 5b8381111561196e57600
0848401525b50505050565b600061197f82611937565b91507fffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffff8214156119b2576119b1611a1c565b5b600182019050919050565b60006119c8
826119cf565b9050919050565b60006119da82611ae9565b9050919050565b6000819050919
050565b60006119f682611937565b9150611a0183611937565b925082611a1157611a10611a
4b565b5b828206905092915050565b7f4e487b7100000000000000000000000000000000000000
0000000000000000000006000526011600452602460 00fd5b7f4e487b7100000000000000000
0000000000000000000000000000000000000000060005260126004526024600 0fd5b7f4e487b7
1000000000000000000000000000000000000000000000000000000000000600052603160045260
246000fd5b7f4e487b7100000000000000000000000000000000000000000000000000000000000000
60005260326004526024600 0fd5b6000601f19601f830116905091905056 5b60008160601b90
50919050565b7f596f75206861766520616c726561647920726567697374657265642100000 00
0060008201525056 5b7f4469636520616c726561647920726f6c6c65642120506c6561736520
636c696360008201527f6b207669657772657375 6c7421000000000000000000000000000000000
0000000060208201525056 5b7f5468657265206973206 16c726561647920322070 6c617965 72
732120596f752060008201527f63616e20656e7465722067616d65206c617465720000000000
0000000000000060208201525056 5b7f506c6179657220322c204e6f7420656e6f7567682062
616c616e636520746f2060008201527f706c617920210000000000000000000000000000000000
0000000000000000000060208201525056 5b7f506c6179657220312c204e6f7420656e6f7567
682062616c616e636520746f2060008201527f706c617920210000000000000000000000000000
0000000000000000000000000060208201525056 5b7f596f75206e65656420746f2072656769
737465722066697273740000000000006000820152505 65b7f796f75206e65656420746f2066
696e697368207468652067616d6520746f207260008201527f656365697665207468652072 65

7761726400000000000000000000000000000060208201525056fea2646970667358221220e4cda87d9cbb276cdf0c78da883afe7e98c01b268bc361cbcb2f0293f02a54ba64736f6c63430008070033",
"linkReferences": {},
"from": "account{0}"
}
},
{
"timestamp": 1667167902374,
"record": {
"value": "4000000000000000000",
"inputs": "()",
"parameters": [],
"name": "register",
"type": "function",
"to": "created{1667167883929}",
"abi": "0x85b639f921ae3230afc3d0821d74e7b52e9cfc3a34ea2b07a563e7d04675414d",
"from": "account{0}"
}
},
{
"timestamp": 1667167925978,
"record": {
"value": "4000000000000000000",
"inputs": "()",
"parameters": [],
"name": "register",
"type": "function",
"to": "created{1667167883929}",
"abi": "0x85b639f921ae3230afc3d0821d74e7b52e9cfc3a34ea2b07a563e7d04675414d",
"from": "account{0}"
}
},
{
"timestamp": 1667167939608,
"record": {
"value": "0",

```
            "inputs": "()",
            "parameters": [],
            "name": "startRolling",
            "type": "function",
            "to": "created{1667167883929}",
            "abi":
"0x85b639f921ae3230afc3d0821d74e7b52e9cfc3a34ea2b07a563e7d04675414d",
            "from": "account{0}"
        }
    },
    {
        "timestamp": 1667167955143,
        "record": {
            "value": "0",
            "inputs": "()",
            "parameters": [],
            "name": "finishgame",
            "type": "function",
            "to": "created{1667167883929}",
            "abi":
"0x85b639f921ae3230afc3d0821d74e7b52e9cfc3a34ea2b07a563e7d04675414d",
            "from": "account{0}"
        }
    },
    {
        "timestamp": 1667167973946,
        "record": {
            "value": "0",
            "inputs": "()",
            "parameters": [],
            "name": "withdrawReward",
            "type": "function",
            "to": "created{1667167883929}",
            "abi":
"0x85b639f921ae3230afc3d0821d74e7b52e9cfc3a34ea2b07a563e7d04675414d",
            "from": "account{0}"
        }
    }
],
```

```
"abis": {
    "0x85b639f921ae3230afc3d0821d74e7b52e9cfc3a34ea2b07a563e7d04675414d": [
        {
            "inputs": [],
            "name": "deposit",
            "outputs": [],
            "stateMutability": "payable",
            "type": "function"
        },
        {
            "inputs": [],
            "name": "finishgame",
            "outputs": [
                {
                    "internalType": "string",
                    "name": "",
                    "type": "string"
                }
            ],
            "stateMutability": "nonpayable",
            "type": "function"
        },
        {
            "inputs": [],
            "name": "register",
            "outputs": [],
            "stateMutability": "payable",
            "type": "function"
        },
        {
            "inputs": [],
            "name": "startRolling",
            "outputs": [],
            "stateMutability": "payable",
            "type": "function"
        },
        {
            "inputs": [],
            "name": "Unauthorized",
```

```
    "type": "error"
},
{
    "inputs": [],
    "name": "withdraw",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [],
    "name": "withdrawReward",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [],
    "name": "timeForTimerToEnd",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
},
{
    "inputs": [],
    "name": "viewBalance",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
```

```
            "stateMutability": "view",
            "type": "function"
          }
        ]
      }
}
```

8. Code of the contract

```
// SPDX-License-Identifier: UNDEFINED

pragma solidity >=0.7.0 <0.9.0;



contract Dice {
    mapping(address => uint256) balance;
    mapping(address => bool) registered;
    address[] private players;
    bool private result;
    bool private rolldone;
    uint256 private val;
    uint256 private diceval;
    uint256 private nonce;
    bool private finished;
    uint32 constant private TIMEOUT = 5 minutes; // after TIMEOUT results the game can
be finished
    uint private startTime;
    error Unauthorized();



    function timeForTimerToEnd() public view returns (uint) {
        if (startTime != 0) {
            uint remainingTime = startTime + TIMEOUT - block.timestamp;
            if (remainingTime < 0) {
                return 0;
            } else {
                return remainingTime;
            }
        }
        return TIMEOUT;
```

```
    }
    function register() public payable{
        require(msg.value >3 ether);
        require(players.length<=1,"There is already 2 players! You can enter game later");
        require(registered[msg.sender]==false,"You have already registered!");
        balance[msg.sender] += msg.value;
        players.push(msg.sender);
        registered[msg.sender]=true;
    }
    function deposit() public payable {
        require(registered[msg.sender]==true,"You need to register first");
        balance[msg.sender] += msg.value ;
    }
    function withdraw() public {
        if (finished==false && rolldone==false){
            uint256 b = balance[msg.sender];
            balance[msg.sender] = 0;
            payable(msg.sender).transfer(b);
        }else if(finished==true && rolldone==true){
            uint256 a = balance[players[0]];
            balance[players[0]] = 0;
            payable(players[0]).transfer(a);
            uint256 b = balance[players[1]];
            balance[players[1]] = 0;
            payable(players[1]).transfer(b);
            reset();
        }else{
            revert Unauthorized();
        }
    }

    function roll() internal returns (uint) {
        startTime = block.timestamp;
        uint randomnumber = uint(keccak256(abi.encodePacked(block.timestamp,
msg.sender, nonce))) % 6;
        randomnumber = randomnumber + 1;
        nonce++;
        return randomnumber;
    }
```

```
function viewBalance() public view returns(uint256){
    return balance[msg.sender];
}
modifier canComputeResult() {
    require(
        (block.timestamp > startTime + TIMEOUT) ||
        (rolldone==true)
    );
    _;
}
function finishgame() public canComputeResult returns(string memory){
    address payable address1 = payable(players[0]);
    address payable address2 = payable(players[1]);
    updateBalances(address1,address2);
    finished=true;
    return result ? "Player1 won" : "Player2 won";
}
function reset() private{
    rolldone=false;
    finished=false;
    startTime=0;
    registered[players[0]]=false;
    registered[players[1]]=false;
    players.pop();
    players.pop();
}

function startRolling() public payable{
    require(balance[players[0]] > 3 , "Player 1, Not enough balance to play !");
    require(balance[players[1]] > 3 , "Player 2, Not enough balance to play !");
    require(rolldone==false,"Dice already rolled! Please click viewresult!");
    diceval=roll();
    rolldone=true;
    if (diceval <4 ){
        result = true;
        val=diceval;
    }else if (diceval>3){
        result = false;
```

```
            val=diceval-3;
        }
    }
    function withdrawReward() public {
        require(finished==true,"you need to finish the game to receive the reward");
        uint256 a = balance[players[0]];
        balance[players[0]] = 0;
        payable(players[0]).transfer(a);
        uint256 b = balance[players[1]];
        balance[players[1]] = 0;
        payable(players[1]).transfer(b);
        reset();
    }
    function updateBalances(address payable address1, address payable address2) private {
        if (result==true){
            balance[address1] += val*1000000000000000000;
            balance[address2] -= val*1000000000000000000;
        }else if (result ==false){
            balance[address2] += val*1000000000000000000;
            balance[address1] -= val*1000000000000000000;
        }
    }
}
```