# S1960565 – Jeet Navindgi – BDL CW2 Report

**Detailed description of the high-level decisions i made for the design of my contract:**

State variables:

I have used a state variable called *gameState* which is of type *enum* which keeps track of the state of the contract/game. The possible game states are *EmptyState*, *WaitingState*, *FullState* and *RevealedState*.

*playersBalances* is a state variable which is a mapping from *address* (players address) to *uin256* (players "bank" balance). My contract acts as the bank from bank.sol from coursework 1, i.e., we allow players to have a contract (bank) balance from which they deposit / withdraw from. Note, players **cannot** deposit directly into this, they will need to do it while entering a game.

I use a commit – reveal scheme in this contract, thus I required a state variable called *commits* which keeps track of all players commitments. This is a mapping from *address* (players address) to *Commit* which is a custom type I defined to represent a commitment. The *Commit* type is defined as a struct with two fields: *bytes32 commit* (random value) and *bool revealed* (whether it has been revealed).

**General flow of the game:**

The state of the game when no players have entered is *EmptyState*. In order for the first player (player A) to join, they should call *openGame*. This function requires the caller to send at least 3.1 Ether when calling (if not already in contract bank balance). Because 3 Ether (-10 Wei for contract profit) is the max amount they can lose in a normal game and an additional 0.1 ether can be lost for unethical actions. Ether deposited into contract bank balance cannot be withdrawn until the game is back to EmptyState, i.e., until the game is over, because we don't want players to withdraw funds that is possibly owed to their opponent. If the player already has ≥3.1 Ether in the contract bank balance, then they are only required (but not limited) to send 10 Wei for contract profit, any more will be deposited into their bank balance. This function also requires an input of type *Bytes32* which will be the commitment value. This value should be pre-processed by the player offline, by doing this following: *keccak256(abi.encodePacked(randomValue, address))* where *randomValue* is a random value of type *bytes32* and *address* is the players address. This then registers the caller as player A and updates the game state to *WaitingState*. Notice how the player has committed a certain amount of ether into this game.

A second player must now join the game by calling *joinGame*. This function requires that the player trying to join the game cannot be the player A. This function can only be called when the state of the game is *WaitingState*. Again, the player follows the same procedure in terms of sending value and passing a *bytes32* argument for a commitment. This then registers the caller as player B and updates the game state to *FullState*.

Both players must now call the *reveal* function passing in their corresponding *randomValue* explained above. We require only the players of the current game to have called this and that the game state should be *FullState.* We also require that the caller hasn't already revealed their value. The last requirement is that the revealed *randomValue* passed in hashed with the caller of the function must equal their original committed value. Once both players have revealed their values, then the state of the game is updated to *RevealedState*.

Both players must now call the *playGame* function. We require callers to be players of the game and the state to be *RevealedState*. This function implements the game that is described. The dice roll will be the XOR of the two *randomValue* values from each player, mod 6, + 1. If this number is 1, 2 or 3 (call it *x*) then player B loses (*x* – (10Wei))Eth from their contract bank balance and player A gains this amount in their contract bank balance. Alternatively, if this number is 4, 5, or 6 (call it *y*) then player A loses ( (y-3) - (10Wei))Eth from their contract bank balance and player B gains this amount in their contract bank balance. The contract bank balances are only fully updated when both players call the function. Once both players have called the function, the game is over, so we set the game state to Empty and reset any state variables that should have their zero value. This is when those players can withdraw their winnings (or whatever they have left). Note, through contract bank balances, the losing player indirectly pays the winning player their reward.

**A thorough list of potential hazards and vulnerabilities that may occur in the contract. A detailed analysis of the security mechanisms used to mitigate such hazards:**

A player can see their opponents *randomValue* once revealed (by looking at the transactions on the block). That player could then be able to determine whether or not they will win the game or not before even revealing (and playing). If they find out that they will lose, they could refuse to reveal and play so that they do not need to pay the opponent or pay gas. The security mechanism I have implemented to handle such players is the timeout mechanisms. If in the *FullState* and (exactly) one player (e.g., player A) is refusing to reveal, player B can call the *StartRevealTimout* function. This function, from the time called gives the *timeout* state variable the value of the timestamp plus two minutes. Meaning, in two minutes this *timeout* will be ≤ *block.timestamp*. If two minutes has passed since player B called *StartRevealTimout,* and player A still has not revealed yet (i.e., the game state is still *FullState*) then player B will be able to call *claimRevealTimeout* which automatically updates game state so that it is over after changing players contract bank balances accordingly. In this scenario, because player A has avoided paying potential gas costs (for revealing and playing), they will lose 3.1 ether. Note the 0.1 ether penalty here to mitigate such behaviour; It will be in the players best interest to finish a game (because it will be cheaper).

Following from the above vulnerability, if both players refuse to play, then the above timeout mechanism will not work because neither of the players will call a timeout. This is a potential DoS attack on the contract because it will forever be in such a 'stale' state. For this, I allow only the owner to be able to start a timer in such a situation. The owner can call the *ownerResetStaleGameTimer* function which starts a five minute timer. If five minutes have passed and the game is still in the same stale state, then the owner will be able to remove 3.1 Ether (minus 10) from each player's contract bank balance by calling *ownerResetStaleGame*. This removed Ether will now belong to the contracts balance. Again, as before, this will reset the game state along with any variables that need resetting.

Even in RevealedState, if one player plays and another doesn't, that player is still avoiding to pay the gas for calling the *playGame* function. Note, the *playGame* function is implemented in such a way that, if the winner calls it, their balance will be updated with their winnings, but the state is still in the *RevealedState* and not over because a player still needs to play. In this situation, the owner can call *ownerClaimPlayTimeout* which uses a timer which was actually already started (inside the *playGame* function) by the one player who called *playGame. ownerClaimPlayTimeout* will take 0.1 ether (- 10Wei) penalty from the contract bank balance of said player. The game state will reset.

I use pull over push method, i.e., we keep track of how much each user is owed and allow the user to withdraw this. This means we don't need to make a (potentially failing) transfer call inside the *playGame* function which is critical to the state of the contract. So no attacker can force a DoS on state of a contract (if there was a way in the first place). This decision does decrease the user experience; however, the trade-off is that it increases the gas fairness. Instead of one of the players (for example, the last player to call the function, potentially the losing player) having to run the code for the transfer is unfair, so we allow for the winner to withdraw their earnings.

Since we use call instead of transfer in the withdraw function, we have a re-entrancy vulnerability. The security mechanism I implemented to counter this is to follow the checks effects interaction pattern. I make sure to set the players contract balance to zero before making any *call* calls which ensures that there is no devastating re-entrancy possible.

we have avoided strict equality checks with regards to the contracts balance, so there is no attack possible with regards to forcibly sending ether to the contract. My contract also uses no libraries, so we are safe against delegate call attacks.

Players cannot cheat by backing out of a game mid-way (by choosing not to interact) to avoid paying. They will also have no way of seeing the opponents random value before committing. They can't do these things because I use a commitment scheme. Players commit 3.1 Ether into their contract bank balance (which is locked in there until they are not in a game), this means that if they choose to no longer interact in the current game, they will lose this committed ether (through timeouts). Players are required to do pre-processing (as mentioned earlier) which includes hashing of their chosen random number (along with their address). Assuming we have a secure hash function, the other player will not be able to determine the original value. If they were able to do this, then they would be able to compute a value that, when xor'd with it will make them win the game. As shown in the lecture, commitment schemes like this are prone to front running on the reveal stage, however this is not possible in my game since the reveal function can only be called by players of the game. We prevent the other player from front running (revealing for the other player) by using *msg.sender* when computing the hash and checking whether it matches the commitment. This guarantees that only the player who made the commitment will be able to reveal it.

For the dice roll, we have avoided using block information as a source of randomness since these values can be manipulated by a malicious miner (who could be a player of the game). Our commitment scheme solves this issue as described earlier. We have both players commit a random 32 byte value which will be xor'd together. We use "mod 6 + 1" to emulate dice conditions. Even if one of the players tries to act maliciously, as long as one of the values are random, then the result of the xor will be random. It is important to note though that if the two numbers are equal, then theses values will xor to 0, causing player A to win 100% of the time if an attacker can force this situation. In my implementation this is impossible because both players have to commit to a value (independent of each other and completely random) in a committed (hashed) form and therefore it is impossible for a player to know what value has been committed since it is hashed (with a secure function).

**Gas fairness:**

My game could have been easily implemented so that only one player needs to run the playGame() function causing both player' balances to change correctly and mitigating the fact that one player may timeout. However, this is a trade-off. If we did this, then it would mean that that player needs to pay considerably more gas for running that function, while the other player would just wait for the outcome without running that function. For this reason, I implemented it so that both players needs to run the same number of similar functions throughout the game. Player A needs to call *openGame* while player B needs to call *joinGame*. These are functions which require similar amounts of gas to run because they do essentially the same thing (there is an additional check in the latter function which cannot be avoided). Then both players need to call *reveal* which is fair (the last player that calls this needs to update the game state, there is no other way around this). Next both players call *playGame* which is fair (the last player that calls this needs to reset the game, there is no other way around this). Each player can withdraw their bank balance independently making it fairer.

Cost of deploying contract : **0.00316675BTL_ETH**

Address of deployed contract: 0x902a4489b04AB63878D0a9938c22148819aA71D5

Please refer to appendix 1 (at the end of this file) for the transaction history of a game.

Analysis of s2457006 – Monica Stephanie:

```
function getRandomNumber() public view returns (uint) {
    return uint(keccak256(abi.encodePacked(owner, block.timestamp))); //vulnerability
}
```

Vulnerability 1: In the above code snippet, this method of calculating the random number can be exploited by players. The owner address is public and the block.timestamp is the time of the function call. Note, the *getRandomNumber* function is only called when a player calls the *Play* function. This can be exploited by doing the following: Player A calls the *Play* function in such a time such that the value of *(keccak256(abi.encodePacked(owner, block.timestamp))) % 6 + 1* is equal to either 1, 2 or 3. This would guarantee player A to win. Since the address of the owner stays the same, player A just needs to make sure the block.timestamp is a suitable value that guarantee's them the win. A way to fix this would be to use a commitment scheme where both players commit a random value which is hidden until both players are committed into the game, then it is revealed and then you can xor these two (random) numbers to get a new random number.

```
function enterPlayerA() public payable {
    require(msg.value == 3 ether, "You need to pay exactly 3 ether to play this game.");
    require(playerA.length == 0, "There's already a player.");
    if (playerB.length == 1) {
        require(msg.sender != playerB[0], "You cannot play against yourself.");
    }
    playerA.push(payable(msg.sender));
}

function enterPlayerB() public payable {
    require(msg.value == 3 ether, "You need to pay exactly 3 ether to play this game.");
    require(playerB.length == 0, "There's already a player.");
    if (playerA.length == 1) {
        require(msg.sender != playerA[0], "You cannot play against yourself.");
    }
    playerB.push(payable(msg.sender));
```

Vulnerability 2: if two malicious players enter the game and none of them decide to ever play (nobody calls *Play*), then they will cause a DoS of the contract, since no players will be able to enter the game until it is over because of the requirements that playerA/B length should be 0. A way to fix this is to allow the owner of the contract to reset such "stale" games once it is confirmed that both players will not interact (maybe after a certain amount of time).

```
    function Play() public payable {
        require(playerA.length + playerB.length == 2, "There's no enough player.");
        uint256 dice = DiceResult() * 1 ether;

        // Give back the initial 3 ether to the winner.
        uint256 trf = dice + 3 ether;

        if (dice < 4) {
            payable(playerA[0]).transfer(trf);  //vulnerabilitys

            // Transfer all the remaining eth to the loser.
            payable(playerB[0]).transfer(address(this).balance);
        }
        else {
            payable(playerB[0]).transfer(dice - 3);

            // Transfer all the remaining eth to the loser.
            payable(playerA[0]).transfer(address(this).balance);
            }

```

Vulnerability 3: There are many transfer calls in the *Play* function (which is a function that is important regarding the state of the contract). Note the limit of gas that a transfer call can use is 2300. If the price of ether increases at a later point in time, then the contract might break because 2300 might not be enough gas for a transfer. The contract breaks in this situation because no players will ever be able to complete a call to the function *Play* because it will fail every time a transfer is attempted (because the price of ether went up). A fix for this is to use *call* instead of *transfer*. It is important to protect against re-entrancy if call is used. If call is used an attacking contract could continuously call *Play* until they drain the contracts balance, so suitable protection will be needed.

Point 4: This is not a vulnerability, more so a comment on gas fairness. Only one player is required to call *Play* before the game ends which means one player will spend considerably more gas than the other, because the play function is a major part of the game. This isn't very fair in terms of gas.

The code of my contract:

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract s1960565 {

    address public playerA;
    address public playerB;
    address private owner;

    enum gameState {EmptyState, WaitingState, FullState, ReavealedState}
    //EmptyState - No players are in a game. Someone should call openGame().
    //WaitingState - One player has committed into a game. Someone else should call joinGame().
    //FullState - Two players are committed into a game. They should both proceed to call reveal().
    //RevealedState - Both players have revealed their random value. They should both proceed to call playGame().

    gameState public state; // Defaults to EmptyState.

    mapping(address => uint256) private playersBalances;
    mapping (address => Commit) private commits;

    bool public A_played;
    bool public B_played;
    bytes32 private A_randomValue;
    bytes32 private B_randomValue;

    uint public timeout = 2**256 - 1;

    struct Commit {
        bytes32 commit;
        bool revealed;
    }
```

```

    constructor(){
        owner = msg.sender;
    }

    //Helper function
    function valueCheck(uint256 msgVal, address msgSender) public payable {
        //If sender has 3 ether already in "bank" balance, they can use this.
        if (playersBalances[msgSender] >= 3.1 ether){
            require(msgVal >= 10, "10 Wei is the minimum value - for contract profit.");
        //Else they need to deposit ether such that they will have atleast 3 ether in their "bank" balance.
        } else {
            require(msgVal >= (3.1 ether - playersBalances[msgSender] + 10), "Atleast 3.1 Ether is needed in bank balance to participate.");
        }
        playersBalances[msgSender] += msgVal - 10;
    }

    modifier onlyState(gameState expectedState) {
        require(state == expectedState, "Game state is not in correct state for this function to be called.");
        _;
    }

```

```
    modifier onlyPlayers {
        require(msg.sender == playerA || msg.sender == playerB, "Only players of the current game can call this function.");
        _;
    }

    modifier onlyOwner {
        require(msg.sender == owner, "Only the owner can call this function.");
        _;
    }

    function openGame(bytes32 randomValAddrHash) public payable onlyState(gameState.EmptyState) {
        valueCheck(msg.value, msg.sender);
        commits[msg.sender].commit = randomValAddrHash; //No notion of a value check here so it is not in the valueCheck function.
        commits[msg.sender].revealed = false;
        playerA = msg.sender;
        state = gameState.WaitingState;
    }

```

```
    function joinGame(bytes32 randomValAddrHash) public payable onlyState(gameState.WaitingState) {
        require(!(msg.sender == playerA), "You can't join a game with yourself");
        valueCheck(msg.value, msg.sender);
        commits[msg.sender].commit = randomValAddrHash; //No notion of a value check here so it is not in the valueCheck functio
        commits[msg.sender].revealed = false;
        playerB = msg.sender;
        state = gameState.FullState;
    }

```

```
    function reveal(bytes32 randomValue) public onlyPlayers onlyState(gameState.FullState) {
        require(commits[msg.sender].revealed == false, "You have already revealed!");
        require(commits[msg.sender].commit == keccak256(abi.encodePacked(randomValue, msg.sender)), "Revealed random number hashe
        if (msg.sender == playerA) {
            A_randomValue = randomValue;
            commits[playerA].revealed = true;
            timeout = 2**256 - 1; // Incase owner started stale game timer
        } else {
            B_randomValue = randomValue;
            commits[playerB].revealed = true;
            timeout = 2**256 - 1;
        }

        if (commits[playerA].revealed == true && commits[playerB].revealed == true){
            state = gameState.ReavealedState;
        }
    }
```

```
    function playGame() public payable onlyPlayers onlyState(gameState.ReavealedState) returns (uint256){
        if (msg.sender == playerA){
            require (!A_played, "You have already played!");
            A_played = true;
        } else {
            require (!B_played, "You have already played!");
            B_played = true;
        }

        timeout = 2**256 - 1; // In case there was a timer initiated in waitingState or if owner started timer
        uint256 random_number = (uint(A_randomValue ^ B_randomValue) % 6) + 1;

        bool A_win;

        if (random_number <= 3) {
            A_win = true;
        } else {
            A_win = false;
        }
```

```
        if (msg.sender == playerB && !A_win){
            playersBalances[playerA] -= (((random_number - 3) * 10**18) - 10); // -10 because remember 10 wei was put into contract profits.
            playersBalances[playerB] += (((random_number - 3) * 10**18) - 10);
        } else if (msg.sender == playerA && A_win){
            playersBalances[playerB] -= (((random_number) * 10**18) - 10);
            playersBalances[playerA] += (((random_number) * 10**18) - 10);
        }

        if (!A_played || !B_played){
                timeout = block.timestamp + 120; // 2 minutes timout interval - only for owner to use
        }

        //Setting state variables to default values once game is over.
        if (A_played && B_played){
            A_played = false;
            B_played = false;
            playerA = address(0);
            playerB = address(0);
            state = gameState.EmptyState;
        }

        return random_number;
    }

```

```
    //Functions for balance operations:

    function withdraw() public payable{
        require(!(msg.sender == playerA || msg.sender == playerB), "You cannot withdraw while in a game"); // To ensure players in a game that
        uint256 b = playersBalances[msg.sender];
        playersBalances[msg.sender] = 0;
        (bool sent, ) = msg.sender.call{value: b}("");
        require(sent, "Failed to withdraw Ether");
    }

    function getBalance() public view returns (uint256){
        return playersBalances[msg.sender];
    }

```

```
    //Timeout functions to follow. These are to ensure nobody can avoid paying gas to play (because they know they have lost),
    //Or to avoid two adversarial players
    //causing a stale game, i.e. a game that never ends and enables a DoS on the contract.

    function startRevealTimeout() public onlyPlayers onlyState(gameState.FullState){
        require(commits[msg.sender].revealed == true, "You can't start this timer because you haven't revealed yet.") ;
        timeout = block.timestamp + 120; // 2 minutes timout interval
    }

```

```
    function claimRevealTimout() public onlyPlayers onlyState(gameState.FullState){
        require(commits[msg.sender].revealed == true, "You are the player who is the facing timeout timer!") ;
        require(block.timestamp >= timeout, "Timeout timer either not started yet or not finished yet.");

        if (msg.sender == playerA) {
            playersBalances[playerB] -= ((3.1 ether) - 10); //0.1 ether penalty for not revealing! To avoid players from
            playersBalances[playerA] += ((3.1 ether) - 10);
        } else {
            playersBalances[playerA] -= ((3.1 ether) - 10);
            playersBalances[playerB] += ((3.1 ether) - 10);
        }

        playerA = address(0);
        playerB = address(0);
        state = gameState.EmptyState;
        timeout = 2**256 - 1;
    }
```

```
    function ownerClaimPlayTimeout() public payable onlyOwner onlyState(gameState.ReavealedState) {
        require(block.timestamp >= timeout, "Timeout timer either not started yet or not finished yet.");
        if (!A_played) {
            playersBalances[playerA] -= (0.1 ether) - 10; //0.1 ether penalty for not playing game - goes to contract balance (i.e. no player wil
        } else {
            playersBalances[playerB] -= (0.1 ether) - 10;
        }
        timeout = 2**256 - 1;
        A_played = false;
        B_played = false;
        playerA = address(0);
        playerB = address(0);
        state = gameState.EmptyState;
    }
```

```
    function ownerResetStaleGameTimer() public payable onlyOwner {
        require(state == gameState.FullState || state == gameState.ReavealedState, "Game is not in expected state for this function.");
        if (state == gameState.FullState) {
            require(commits[playerA].revealed == false && commits[playerB].revealed == false, "Game is not in a stale condition!");
        } else {
            require(!A_played && !B_played, "Game is not in a stale condition!");
        }
        timeout = block.timestamp + 300;
        }

    function ownerResetStaleGame() public payable onlyOwner {
        require(state == gameState.FullState || state == gameState.ReavealedState, "Game is not in expected state for this function.");
        require(block.timestamp >= timeout, "Timeout timer either not started yet or not finished yet.");
        //No further checks required - by logic of code it is garunteed that game is in the SAME stale state
        //Because we reset timout variable in each new game state.

        // Both players will lose their deposited 3.1 eth and the game restarts.
        playersBalances[playerA] -= (3.1 ether) - 10; // Remains in contract balance i.e. goes to contract balance.
        playersBalances[playerB] -= (3.1 ether) - 10;
        timeout = 2**256 - 1;
        playerA = address(0);
        playerB = address(0);
        state = gameState.EmptyState;
    }
}
```

Appendix 1: transaction history JSON.

{

"accounts": {

"account{0}": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",

"account{1}": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2",

"account{2}": "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db"

},

"linkReferences": {},

"transactions": [

{

"timestamp": 1667147274020,

"record": {

"value": "0",

"inputs": "()",

"parameters": [],

"name": "",

"type": "constructor",

"abi":
"0x544bc6b2eaba4fb44b0584c30b6e32e25c56c6691d66c7893bf95c263533c370",

"contractName": "s1960565",

"bytecode":
"60806040527fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff600855348015610034576000
080fd5b5033600260006101000a81548173ffffffffffffffffffffffffffffffffffffffff021916908373fffff
ffffffffffffffffffffffffffffffffffff1602179055506137d7806100856000396000f3fe6080604052600
436106101095760003560e01c806378c85f7c11610095578063af89973511610064578063af8
99735146102925780 63b6103b41146102b0578063bf474766146102ba578063c19d93fb146
102d6578063ca69d46c146103015761010956 5b806378c85f7c146102045780 63a285c54a14
610220578063ade636441461024b578063af52b2cd146102765761010956 5b80634201fa121
16100dc5780634201fa1214610178578063 50467c50146101825780 63701fd0f11461019957
806370dea79a146101c257806374a4dda0146101ed5761010956 5b806311bb15371461010e
57806312065fe0146101395780632efa59eb146101645780633ccfd60b1461016e575b60008
0fd5b34801561011a57600080fd5b5061012361032c565b60405161013091906 12d9a565b60
405180910390f35b34801561014557600080fd5b5061014e610352565b60405161015b9190
612feb565b60405180910390f35b61016c610399565b005b6101766106d5565b005b610180

6108f5565b005b34801561018e57600080fd5b50610197610c53565b005b3480156101a5576
00080fd5b506101c0600480360381019061 01bb9190612a2f565b610e5c565b005b34801561
01ce57600080fd5b506101d76113d8565b6040516101e49190612feb565b60405180910390f
35b3480156101f957600080fd5b50610202 6113de565b005b61021e60048036038101906 10
2199190612a2f565b611946565b005b34801561022c57600080fd5b5061023561 1ad8565b60
40516102429190612d9a565b60405180910390f35b34801561025757600080fd5b50610260
611afc565b60405161026d9190612db5565b60405180910390f35b61029060048036038101
9061028b9190612a5c565b611b0f565b005b61029a611cae565b6040516102a79190612feb5
65b60405180910390f35b6102b8612436565b005b6102d4600480360381019061 02cf91906
12a2f565b6127a9565b005b3480156102e257600080fd5b506102eb6129ca565b6040516102
f89190612dd0565b60405180910390f35b34801561030d57600080fd5b506103166129dd565
b6040516103239190612db5565b60405180910390f35b600160009054906101000a900473ff
ffffffffffffffffffffffffffffffffffffff1681565b6000600360003373ffffffffffffffffffffffffffffffffffffffff167
3ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054905090565b6002600
09054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1
63373ffffffffffffffffffffffffffffffffffffffff16146104295760405 17f08c379a000000000000000000
000000000000000000000000000000000000000815260040161042090612e2b565b6040518
0910390fd5b600260038111156104 3d5761043c61323a565b5b600260149054906101000a9
00460ff16600381111561045f5761045e61323a565b5b148061049d5750600380811115610 4
7957610478 61323a565b5b600260149054906101000a900460ff16600381111561049b5761
049a61323a565b5b145b6104dc576040517f08c379a0000000000000000000000000000000
000000000000000000000000008152600401 6104d390612fab565b60405180910390fd5b60
0260038111156104f0576104ef61323a565b5b600260149054906101000a900460ff1660038
1111561051257610511 61323a565b5b1415610655576000151560046000806000905490610
1000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673fffffffffff
fffffffffffffffffffffffffffff168152602001908152602001600020600101600090549061 01000a90
0460ff1615151480156106115750600015156004600060016000905490610 1000a900473fffff
ffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffff
ffffff16815260200190815260200160002060010160009054906101000a900460ff161515145
b610650576040517f08c379a000000000000000000000000000000000000000000000000000
0000008152600401610647 90612f8b565b60405180910390fd5b6106bf565b600560009054
906101000a900460ff1615801561067f5750600560019054906101000a900460ff16155b6106
be576040517f08c379a00000000000000000000000000000000000000000000000000000000
008152600401 6106b590612f8b565b60405180910390fd5b5b61012c426106cd91906130225
65b600881905550565b60008054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673fffff
ffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff16148061077c57506001600
09054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1
63373ffffffffffffffffffffffffffffffffffffffff16145b156107bc576040517f08c379a0000000000000
00000000000000000000000000000000000000000000008152600401 6107b390612e8b565b604
05180910390fd5b6000600360003373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffff
ffffffffffffff16815260200190815260200160002054905060006003600033 73fffffffffffffffffffff
fffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081526020016000208190
55506000 3373ffffffffffffffffffffffffffffffffffffffff168260405161086b90612d85565b6000604051
8083038185875af1925050503d8060008114610 8a8576040519150601f19603f3d011682016

040523d82523d6000602084013e6108ad565b606091505b50509050806108f1576040517f0
8c379a00000000000000000000000000000000000000000000000000000000008152600401 6
108e890612f2b565b60405180910390fd5b5050565b600260009054906101000a900473ffffff
ffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373fffffffffffffffffffffffffffffffff
fffffff1614610985576040517f08c379a0000000000000000000000000000000000000000000
000000000000008152600401610 97c90612e2b565b60405180910390fd5b60026003811115
6109995761099861323a565b5b600260149054906101000a900460ff166003811115 6109bb
576109ba61323a565b5b14806109f957506003808111156109d5576109d461323a565b5b60
02601490549061 01000a900460ff166003811115 6109f7576109f661323a565b5b145b610a3
8576040517f08c379a00000000000000000000000000000000000000000000000000000000
08152600401610a2f90612fab565b60405180910390fd5b600854421015610a7d576040517f
08c379a0000000000000000000000000000000000000000000000000000000008152600401
610a7490612eab565b60405180910390fd5b672b05699353b5fff6600360008060009054906
101000a900473ffffffffffffffffffffffffffffffffffffffff1673fffffff
fffffffffffffffffffffffffffffffff168152602001908152602001600020600082825461 0af591906130d
2565b92505081905550672b05699353b5fff66003600060016000905490610 1000a900473fff
fffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffffffff
fffffff168152602001908152602001600020600082825461 0b7591906130d2565b925050819
055507fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff60088190555060008060006101000
a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff16021
790555060006001600061 01000a81548173ffffffffffffffffffffffffffffffffffffffff021916908373fffff
ffffffffffffffffffffffffffffffffff1602179055506000600260146101000a81548160ff02191690836
003811115610c4c57610c4b61323a565b5b02179055 50565b60008054906101000a900473ff
ffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373fffffffffffffffffffffffffffff
fffffffffff161480610cfa5750600160009054906101000a900473fffffffffffffffffffffffffffffffffffffff
f1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff16145b610d39576
040517f08c379a000000000000000000000000000000000000000000000000000000000815
2600401610d3090612f4b565b60405180910390fd5b6002806003811115610d4e57610d4d6
1323a565b5b600260149054906101000a900460ff166003811115610d7057610d6f61323a56
5b5b14610db0576040517f08c379a000000000000000000000000000000000000000000000
0000000000008152600401610da790612e0b565b60405180910390fd5b6001151560046000
3373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152
6020016000206001016000905490610 1000a900460ff16151514610e46576040517f08c379a
0000000000000000000000000000000000000000000000000000000008152600401610e3d9
0612e4b565b60405180910390fd5b607842610e539190613022565b60088190555050565b6
00080549061 01000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffff
ff163373ffffffffffffffffffffffffffffffffffffffff161480610f0357506001600090549061 01000a90047
3ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffff
ffffffffffffff16145b610f42576040517f08c379a0000000000000000000000000000000000000
000000000000000000008152600401610f3990612f4b565b60405180910390fd5b60028060
03811115610f5757610f5661323a565b5b600260149054906101000a900460ff16600381111
5610f7957610f7861323a565b5b14610fb9576040517f08c379a0000000000000000000000
000000000000000000000000000000000008152600401610fb090612e0b565b604051809103
90fd5b60001515600460003373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffff

fffffff16815260200190815260200160002060010160009054906101000a900460ff16151514
61104f576040517f08c379a0000000000000000000000000000000000000000000000000000
00000815260040161104690612f0b565b60405180910390fd5b81336040516020016110629
29190612d59565b60405160208183030381529060405280519060200120600460003373fffff
ffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081526020016
0002060000154146110fb576040517f08c379a0000000000000000000000000000000000000
00000000000000000000815260040161 10f290612fcb565b60405180910390fd5b600080549
06101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373f
fffffffffffffffffffffffffffffffffffffff1614156111fe578160068190555060016004600080600090549
06101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673fffff
ffffffffffffffffffffffffffffffffffff16815260200190815260200160002060010160006101000a81548
160ff0219169083151502179055507ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff60088
19055506112aa565b81600781905550600160046000600160009054906101000a900473fffff
ffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffff
ffffff16815260200190815260200160002060010160006101000a81548160ff0219169083151
502179055507ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff6008819055505b600115156
00460008060009054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffff
ffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060 01
0160009054906101000a900460ff1615151480156113a357506001151560046000600160009
054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff167
3ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020600101600090549 0610
1000a900460ff161515145b156113d4576003600260146101000a81548160ff021916908360
038111156113ce576113cd61323a565b5b02179055505b5050565b60085481565b60008054
906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373
ffffffffffffffffffffffffffffffffffffffff1614806114855750600160009054906101000a900473ffffffffff
ffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffff
ffff16145b6114c4576040517f08c379a00000000000000000000000000000000000000000000
000000000000081526004016114bb90612f4b565b60405180910390fd5b60028060038111
156114d9576114d861323a565b5b600260149054906101000a900460ff1660038111156114f
b576114fa61323a565b5b1461153b576040517f08c379a0000000000000000000000000000
00000000000000000000000000000815260040161153290612e0b565b60405180910390fd5b
60011515600460003373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16
8152602001908152602001600020600101600090549061 01000a900460ff161515146115d15
76040517f08c379a0000000000000000000000000000000000000000000000000000000008
1526004016115c890612e6b565b60405180910390fd5b600854421015611616576040517f08
c379a000000000000000000000000000000000000000000000000000000000815260040161
160d90612eab565b60405180910390fd5b60008054906101000a900473fffffffffffffffffffffffffff
fffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff16141561
176e57672b05699353b5fff66003600060016000905490610 1000a900473fffffffffffffffffffffffff
fffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602
00190815260200160002060008282546116e391906130d2565b925050819055 50672b05699
353b5fff660036000806000905490610 1000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffff
ffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160
002060008282546117629190613022565b92505081905550 61186e565b672b05699353b5fff

6600360008060009054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffff
fffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060
008282546117e691906130d2565b92505081905550672b05699353b5fff6600360006001600
09054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1
673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060008282546118669
190613022565b925050819055505b60008060006101000a81548173ffffffffffffffffffffffffffffffff
ffffffff021916908373ffffffffffffffffffffffffffffffffffffffff16021790555060006001600061 01000a
81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff160217
9055506000600260146101000a81548160ff021916908360038111156119175761191661323
a565b5b02179055507ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff60088190555050565
b600080600381111561195b5761195a61323a565b5b60026014905490610 1000a900460ff16
600381111561197d5761197c61323a565b5b146119bd576040517f08c379a0000000000000
000000000000000000000000000000000000000000008152600401 6119b490612e0b565b60
405180910390fd5b6119c73433611b0f565b81600460003373ffffffffffffffffffffffffffffffffffffffff
1673ffffffffffffffffffffffffffffffffffffffff1681526020019081526020016000206000018190555060
00600460003373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260
20019081526020016000206001016000610 1000a81548160ff0219169083151502179055503
36000806101000a81548173ffffffffffffffffffffffffffffffffffffffff021916908373fffffffffffffffffffffffff
fffffffffffffff16021790555060016002601461 01000a81548160ff02191690836003811115611a
cf57611ace61323a565b5b02179055505050565b60008054906101000a900473ffffffffffffffffff
ffffffffffffffffffffff1681565b6005600190549061 01000a900460ff1681565b672b05699353b60
000600360008373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526
020019081526020016000205410611ba657600a821015611ba1576040517f08c379a000000
0000000000000000000000000000000000000000000000000008152600401611b9890612f6
b565b60405180910390fd5b611c48565b600a600360008373ffffffffffffffffffffffffffffffffffffffff1
673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054672b05699353b60
000611bfb91906130d2565b611c059190613022565b821015611c47576040517f08c379a000
0000000000000000000000000000000000000000000000000000008152600401611c3e9061
2ecb565b60405180910390fd5b5b600a82611c5591906130d2565b600360008373ffffffffffffff
ffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020
6000828254611ca39190613022565b925050819055505050565b6000806000905490610100
0a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffff
ffffffffffffffffffffffffffff161480611d585750600160009054906101000a900473fffffffffffffffffffffff
fffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff16145
b611d97576040517f08c379a00000000000000000000000000000000000000000000000000
00000008152600401611d8e90612f4b565b60405180910390fd5b6003806003811115611dac
57611dab61323a565b5b60026014905490610 1000a900460ff166003811115611dce57611dc
d61323a565b5b14611e0e576040517f08c379a000000000000000000000000000000000000
0000000000000000000008152600401611e0590612e0b565b60405180910390fd5b6000805
4906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16337
3ffffffffffffffffffffffffffffffffffffffff161415611ed2576005600090549061 01000a900460ff16156
11eb2576040517f08c379a0000000000000000000000000000000000000000000000000000
000008152600401611ea990612deb565b60405180910390fd5b60016005600061 01000a815
48160ff021916908315150217905550611f3e565b6005600190549061 01000a900460ff16156

11f22576040517f08c379a00000000000000000000000000000000000000000000000000000
00008152600401611f1990612deb565b60405180910390fd5b6001600560016101000a8154
8160ff0219169083151502179055505b7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff60
0881905550600060016000660075460065418600 01c611f7f91906131ab565b611f899190613
022565b9050600060038211611f9e5760019050611fa3565b600090505b600160009054906
101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373fffff
ffffffffffffffffffffffffffffffffff16148015611ffe575080155b1561214d57600a670de0b6b3a7640
00060038461201b91906130d2565b6120259190613078565b61202f91906130d2565b60036
00080600090549061010000a900473ffffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffffff
fffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060008282
5461209e91906130d2565b92505081905550600a670de0b6b3a76400006003846120bd9190
6130d2565b6120c79190613078565b6120d191906130d2565b600360006001600090549061
01000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673fffffffff
ffffffffffffffffffffffffffffff16815260200190815260200160002060008282546121419190613022
565b925050819055506122d9565b600080549061010000a900473ffffffffffffffffffffffffffffffffffff
ffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff161480156121a
55750805b156122d857600a670de0b6b3a7640000836121c09190613078565b6121ca91906
130d2565b60036000600160009054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673f
fffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681526020019081526020
01600020600082825461223a91906130d2565b92505081905550600a670de0b6b3a7640000
836122579190613078565b61226191906130d2565b600360008060009054906101000a9004
73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673fffffffffffffffffffffffffff
fffffffffffff16815260200190815260200160002060008282546122d09190613022565b92505
0819055505b5b600560009054906101000a900460ff1615806123025750600560019054906
101000a900460ff16155b1561231b576078426123149190613022565b6008819055505b600
560009054906101000a900460ff16801561234357506005600190549061 01000a900460ff16
5b1561242d576000600560006101000a81548160ff021916908315150217905550600060 05
60016101000a81548160ff02191690831515021790555060008060006101000a81548173fffff
ffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff16021790555060006
00160006101000a81548173ffffffffffffffffffffffffffffffffffffffff021916908373fffffffffffffffffffffffffff
fffffffffffff1602179055506000600260146101000a81548160ff021916908360038111156124
275761242661323a565b5b02179055505b8193505050509056 5b600260009054906101000a
900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffff
ffffffffffffffffffffffff16146124c6576040517f08c379a000000000000000000000000000000000
00000000000000000000000081526004016124bd90612e2b565b60405180910390fd5b6003
8060038111156124db576124da61323a565b5b600260149054906101000a900460ff166003
8111156124fd576124fc61323a565b5b1461253d576040517f08c379a00000000000000000
000000000000000000000000000000000000000081526004016125349061 2e0b565b6040518
0910390fd5b600854421015612582576040517f08c379a00000000000000000000000000000
000000000000000000000000000081526004016125799061 2eab565b60405180910390fd5b
600560009054906101000a900460ff1661261a5767016345785d89fff6600360008060009054
906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673fff
fffffffffffffffffffffffffffffffffffff168152602001908152602001600020600082825461260e91906
130d2565b92505081905550 61269b565b67016345785d89fff660036000600160009054906 1

01000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1673fffffffff
fffffffffffffffffffffffffffffff168152602001908152602001600020600082825461269391906130d2
565b925050819055505b7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff6008819055506
000600560006101000a81548160ff021916908315150217905550600060056001610100 0a81
548160ff02191690831515021790555060008060006101000a81548173fffffffffffffffffffffffffffff
fffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff1602179055506000600160006101000
0a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff1602
17905550600060026014610100 0a81548160ff02191690836003811115612 7a1576127a0613
23a565b5b021790555050565b600180600381111561 27be576127bd61323a565b5b6002601
49054906101000a900460ff1660038111156127e0576127df61323a565b5b14612820576040
517f08c379a000000000000000000000000000000000000000000000000000000000815260
04016128179061 2e0b565b60405180910390fd5b600080549061 01000a900473fffffffffffffffff
ffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffffffffffffffffffff16
14156128af576040517f08c379a000000000000000000000000000000000000000000000000
000000000815260040161 28a690612eeb565b60405180910390fd5b6128b93433611b0f565
b81600460003373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152
602001908152602001600020600001819055506000600460003373fffffffffffffffffffffffffffffffff
fffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002060010160006
101000a81548160ff02191690831515021790555033600160006101000a81548173fffffffffffff
fffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff160217905550600280601
46101000a81548160ff02191690836003811115612 9c1576129c061323a565b5b0217905550
5050565b600260149054906101000a900460ff1681565b6005600090549061 01000a900460f
f1681565b60008135905061 29ff8161375c565b92915050565b600081359050612a14816137
73565b92915050565b600081359050612a298161378a565b92915050565b600060208284 03
1215612a4557612a44613269565b5b6000612a5384828501612a05565b915050929150505 6
5b600080604083850312156 12a7357612a72613269565b5b6000612a8185828601612a1a56
5b9250506020612a9285828601 6129f0565b9150509250929050565b612aa581613106565b
82525050565b612abc612ab782613106565b61317d565b82525050565b612acb8161311856
5b82525050565b612ae2612add82613124565b61318f565b82525050565b612af18161316b
565b82525050565b6000612b04601883613011565b9150612b0f8261327b565b6020820190
50919050565b6000612b27604283613011565b9150612b32826132a4565b6060820190509 1
9050565b6000612b4a602683613011565b9150612b5582613319565b6040820190509190 50
565b6000612b6d603c83613011565b9150612b7882613368565b604082019050919050565b
6000612b90603383613011565b9150612b9b826133b7565b604082019050919050565b600
0612bb3602383613011565b9150612bbe82613406565b604082019050919050565b600061
2bd6603983613011565b9150612be182613455565b604082019050919050565b6000612bf9
603b83613011565b9150612c04826134a4565b604082019050919050565b6000612c1c6023
83613011565b9150612c27826134f3565b604082019050919050565b6000612c3f601a83613
011565b9150612c4a82613542565b602082019050919050565b6000612c626018836130115
65b9150612c6d8261356b565b602082019050919050565b6000612c85603883613011565b9
150612c9082613594565b604082019050919050565b6000612ca8603283613011565b91506
12cb3826135e3565b604082019050919050565b6000612ccb600083613006565b9150612cd
682613632565b600082019050919050565b6000612cee602183613011565b9150612cf9826
13635565b604082019050919050565b6000612d11603083613011565b9150612d1c8261368

4565b60408201905091905050565b6000612d34604a83613011565b9150612d3f826136d3565
b60608201905091905050565b612d5381613161565b82525050565b6000612d658285612ad15
65b602082019150612d758284612aab565b601482019150819050939250505050565b6000612
d9082612cbe565b915081905091905050565b600060208201905061 2daf6000830184612a9c5
65b92915050565b600060208201905061 2dca6000830184612ac2565b92915050565b60006
02082019050612de56000830184612ae8565b92915050565b600060208201905081810360 0
0830152612e0481612af7565b9050919050565b6000602082019050818103600083015261 2
e2481612b1a565b9050919050565b600060208201905081810360008301526 12e4481612b3
d565b9050919050565b6000602082019050818103600083015261 2e6481612b60565b90509
19050565b6000602082019050818103600083015261 2e8481612b83565b9050919050565b6
00060208201905081810360008301526 12ea481612ba6565b9050919050565b60006020820
19050818103600083015261 2ec481612bc9565b9050919050565b600060208201905081810
36000830152612ee481612bec565b9050919050565b6000602082019050818103600083015
2612f0481612c0f565b9050919050565b6000602082019050818103600083015261 2f248161
2c32565b9050919050565b6000602082019050818103600083015261 2f4481612c55565b90
50919050565b6000602082019050818103600083015261 2f6481612c78565b905091905056
5b6000602082019050818103600083015261 2f8481612c9b565b9050919050565b60006020
82019050818103600083015261 2fa481612ce1565b9050919050565b600060208201905081
81036000830152612fc481612d04565b9050919050565b600060208201905081810360008 3
0152612fe481612d27565b9050919050565b6000602082019050613000600083018461 2d4a
565b92915050565b6000819050929150505 65b6000828252602082019050929150505 65b60
0061302d82613161565b915061303883613161565b9250827fffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffff0382111561306d5761306c6131dc565b5b828201905092915050565b600
061308382613161565b915061308e83613161565b9250817fffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffff04831182151516156130c7576130c66131dc565b5b8282029050929150505
65b60006130dd82613161565b91506130e883613161565b925082821015 6130fb576130fa6
131dc565b5b828203905092915050565b6000613111826131 41565b9050919050565b60008
115159050919050565b6000819050919050565b6000819050 61313c82613748565b9190505
65b600073ffffffffffffffffffffffffffffffffffffffff82169050919050565b6000819050919050565b600
06131768261312e565b9050919050565b600061318882613199565b9050919050565b60008
19050919050565b60006131a48261326e565b9050919050565b60006131b682613161565b9
1506131c183613161565b9250826131d1576131d061320b565b5b828206905092915050565
b7f4e487b7100000000000000000000000000000000000000000000000000000000006000526
01160045260246000fd5b7f4e487b7100000000000000000000000000000000000000000000
0000000000006000526012600452602460 00fd5b7f4e487b7100000000000000000000000
000000000000000000000000000000006000526021600452602460 00fd5b600080fd5b600
08160601b9050919050565b7f596f752068617665 20616c7265616479 20706c61796564210 0
00000000000000600082015250565b7f47616d65207374617465206973206e6f7420696e20
636f72726563742073746160008201527f746520666f7220746869732066756e6374696f6e2
0746f2062652063616c6c656020820152 7f642e0000000000000000000000000000000000000
00000000000000000000000604082015250565b7f4f6e6c7920746865206f776e657220636
16e2063616c6c20746869732066756e60008201527f6374696f6e2e000000000000000000000
0000000000000000000000000000000602082015250565b7f596f752063616e2774207374
6172742074686973207469 6d652062656361756 0008201527f736520796f752068617665

6e27742072657665616c6564207965742e0000000060208201525056 5b7f596f75206172652
074686520706c617965722077686f206973207468652066616000820152 7f63696e67207469
6d656f75742074696d6572210000000000000000000000000000006020820152505 65b7f596f752
063616e6e6f742077697468647261772077686696c6520696e2061206760008201527f616d65
00000000000000000000000000000000000000000000000000006020820152505 65b
7f54696d656f75742074696d65722065697468657220 6e6f742073746172746564600082015
27f20796574206f72206e6f742066696e6973686564207965742e0000000000000060208201
5250565b7f41746c6561737420332e3120457468657220697320 6e65656465642069 6e2062
60008201527f616e6b2062616c616e636520746f207061727469636970617465 2e000000000
0602082015250565b7f596f752063616e2774206a6f696e20612067616d6520776974682079
6f7572736000820152 7f656c66000000000000000000000000000000000000000000000000000
0000000060208201525056 5b7f596f7520686176652061 6c7265616479207265766561 6c65
64210000000000000060008201525056 5b7f4661696c656420746f2077697468647261772045 7
46865720000000000000000000060008201525056 5b7f4f6e6c7920706c617965727320 6f662074
68652063757272656e742067616d656000820152 7f2063616e2063616c6c207468697320 66
756e6374696f6e2e00000000000000006020820152505 65b7f3130205765692069732074686
5206d696e696d756d2076616c7565202d20666f600082015 27f7220636f6e74726163742070
726f6669742e000000000000000000000000000000006020820152505 65b50565b7f47616d6520
6973206e6f7420696e2061207374616c6520636f6e646974696f6e600082015 27f2100000000
00000000000000000000000000000000000000000000000000000060208201525056 5b7f47
616d6520697320 6e6f7420696e2065787065637465642073746174652066 6f600082015 27f7
2207468697320 66756e6374696f6e2e000000000000000000000000000000000000006020820 1525
0565b7f52657665616c65642072616e646f6d206e756d6265722068617368656420776960 00
8201527f746820796f757220616464726573732064 6f6573206e6f74206d617463682077602
08201527f69746820636f6d6d697400000000000000000000000000000000000000000000 6
04082015250565b600481106137595761375 8613 23a565b5b50565b61376581613106565b8
1146137705760008 0fd5b50565b61377c81613124565b8114613787576000 80fd5b50565b6
137938161316156 5b811461379e57600080fd5b5056fea2646970667358221 2202b3ff16214
50884168eb41e4dc315573ffcd0901264995638b3050f0a604819c64736f6c63430008070033
",

"linkReferences": {},

"from": "account{0}"

}

},

{

"timestamp": 1667147285600,

"record": {

"value": "10000000000000000000",

"inputs": "(bytes32)",

"parameters": [
"0x7290a40daf4afb983b1cf71b1151fb740841d8d2d1d06c0d62492b301b9a88b8"
],
"name": "openGame",
"type": "function",
"to": "created{1667147274020}",
"abi":
"0x544bc6b2eaba4fb44b0584c30b6e32e25c56c6691d66c7893bf95c263533c370",
"from": "account{1}"
}
},
{
"timestamp": 1667147298734,
"record": {
"value": "10000000000000000000",
"inputs": "(bytes32)",
"parameters": [
"0x66adaaf1a3af1919627105cba70d4edd6fb01db860061674ef16fc3531b42a1f"
],
"name": "joinGame",
"type": "function",
"to": "created{1667147274020}",
"abi":
"0x544bc6b2eaba4fb44b0584c30b6e32e25c56c6691d66c7893bf95c263533c370",
"from": "account{2}"
}
},
{
"timestamp": 1667147318066,
"record": {

"value": "0",
"inputs": "(bytes32)",
"parameters": [
"0x70fc4772006b8e902c709721a85d85327e21164eb13f20f32fa1533c4ca5d000"
],
"name": "reveal",
"type": "function",
"to": "created{1667147274020}",
"abi":
"0x544bc6b2eaba4fb44b0584c30b6e32e25c56c6691d66c7893bf95c263533c370",
"from": "account{1}"
}
},
{
"timestamp": 1667147326525,
"record": {
"value": "0",
"inputs": "(bytes32)",
"parameters": [
"0x7f08b95588435e62e23470bf6cd7284198a48e0e0729dbdfd25b7b401a9a0000"
],
"name": "reveal",
"type": "function",
"to": "created{1667147274020}",
"abi":
"0x544bc6b2eaba4fb44b0584c30b6e32e25c56c6691d66c7893bf95c263533c370",
"from": "account{2}"
}
},
{

"timestamp": 1667147361741,
"record": {
"value": "0",
"inputs": "()",
"parameters": [],
"name": "playGame",
"type": "function",
"to": "created{1667147274020}",
"abi":
"0x544bc6b2eaba4fb44b0584c30b6e32e25c56c6691d66c7893bf95c263533c370",
"from": "account{1}"
}
},
{
"timestamp": 1667147367406,
"record": {
"value": "0",
"inputs": "()",
"parameters": [],
"name": "playGame",
"type": "function",
"to": "created{1667147274020}",
"abi":
"0x544bc6b2eaba4fb44b0584c30b6e32e25c56c6691d66c7893bf95c263533c370",
"from": "account{2}"
}
},
{
"timestamp": 1667147528534,
"record": {

"value": "0",
"inputs": "(bytes32)",
"parameters": [
"0x7290a40daf4afb983b1cf71b1151fb740841d8d2d1d06c0d62492b301b9a88b8"
],
"name": "openGame",
"type": "function",
"to": "created{1667147274020}",
"abi":
"0x544bc6b2eaba4fb44b0584c30b6e32e25c56c6691d66c7893bf95c263533c370",
"from": "account{1}"
}
}
],
"abis": {
"0x544bc6b2eaba4fb44b0584c30b6e32e25c56c6691d66c7893bf95c263533c370": [
{
"inputs": [],
"name": "claimRevealTimout",
"outputs": [],
"stateMutability": "nonpayable",
"type": "function"
},
{
"inputs": [
{
"internalType": "bytes32",
"name": "randomValAddrHash",
"type": "bytes32"
}

],
"name": "joinGame",
"outputs": [],
"stateMutability": "payable",
"type": "function"
},
{
"inputs": [
{
"internalType": "bytes32",
"name": "randomValAddrHash",
"type": "bytes32"
}
],
"name": "openGame",
"outputs": [],
"stateMutability": "payable",
"type": "function"
},
{
"inputs": [],
"name": "ownerClaimPlayTimeout",
"outputs": [],
"stateMutability": "payable",
"type": "function"
},
{
"inputs": [],
"name": "ownerResetStaleGame",

```
  "outputs": [],
  "stateMutability": "payable",
  "type": "function"
},
{
  "inputs": [],
  "name": "ownerResetStaleGameTimer",
  "outputs": [],
  "stateMutability": "payable",
  "type": "function"
},
{
  "inputs": [],
  "name": "playGame",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "payable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "bytes32",
      "name": "randomValue",
```

```
      "type": "bytes32"
    }
  ],
  "name": "reveal",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [],
  "name": "startRevealTimeout",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [],
  "stateMutability": "nonpayable",
  "type": "constructor"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "msgVal",
      "type": "uint256"
    },
    {
      "internalType": "address",
```

"name": "msgSender",
"type": "address"
}
],
"name": "valueCheck",
"outputs": [],
"stateMutability": "payable",
"type": "function"
},
{
"inputs": [],
"name": "withdraw",
"outputs": [],
"stateMutability": "payable",
"type": "function"
},
{
"inputs": [],
"name": "A_played",
"outputs": [
{
"internalType": "bool",
"name": "",
"type": "bool"
}
],
"stateMutability": "view",
"type": "function"
},

```
{
  "inputs": [],
  "name": "B_played",
  "outputs": [
    {
      "internalType": "bool",
      "name": "",
      "type": "bool"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "getBalance",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "playerA",
```

"outputs": [
  {
    "internalType": "address",
    "name": "",
    "type": "address"
  }
],
"stateMutability": "view",
"type": "function"
},
{
"inputs": [],
"name": "playerB",
"outputs": [
  {
    "internalType": "address",
    "name": "",
    "type": "address"
  }
],
"stateMutability": "view",
"type": "function"
},
{
"inputs": [],
"name": "state",
"outputs": [
  {
    "internalType": "enum s1960565.gameState",

```
          "name": "",
          "type": "uint8"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    },
    {
      "inputs": [],
      "name": "timeout",
      "outputs": [
        {
          "internalType": "uint256",
          "name": "",
          "type": "uint256"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    }
  ]
 }
}
```