Bootcamp documentation

*Topic: Morra game*

|  |  |
| --- | --- |
| **Full name** | ***Luka Šević*** |
| **GitHub** | ***https://github.com/lcminer/Reach-Morra-assignment*** |

# Morra game

Morra is an old hand game in which two players show each other hand with a certain number of fingers raised, while simultaneously pronouncing a number. The player which guesses the total number of raised fingers is the winner.

# Smart contract

const [ isHand, ZERO, ONE, TWO, THREE, FOUR, FIVE ] = makeEnum(6);

const [ isVocal, ZEROV, ONEV, TWOV, THREEV, FOURV, FIVEV, SIXV, SEVENV, EIGHTV, NINEV, TENV ] = makeEnum(11);

const [ isOutcome, B\_WINS, DRAW, A\_WINS ] = makeEnum(3);

This code enumerates variables isHand(fingers raised), isVocal(players` guess) and possible outcomes(Player 1 / 2 wins, or it is a draw)

const winner is a function which determines the winner of the from the given variables. It first checks if the players guessed the same number(which results in a draw), if a player guessed the right number(which makes him a winner). If none of that is true(both players guessed wrong), the result is draw.

‘forall’ functions are used to check if the results given by winner function are correct.

‘const Player’ defines the player`s available methods.

Alice.Only and Bob.only functions let players declare their variables and decrypt the values of variables using declassify() function, while Alice.Publish and Bob.Publish functions are used for paying the wager, sending the hashed values of variables and sending the deadline.

The variable ‘outcome’ gets the value of ‘winner’ function being executed for corresponding values of players variables.

The program is in a WHILE loop until the winner is decided. When a player wins, wager is transferred to his account and outcome is shown to both players.

# Console application

index.mjs:

import { loadStdlib } from '@reach-sh/stdlib';

import \* as backend from './build/index.main.mjs';

const stdlib = loadStdlib(process.env);

const startingBalance = stdlib.parseCurrency(100);

const accAlice = await stdlib.newTestAccount(startingBalance);

const accBob = await stdlib.newTestAccount(startingBalance);

const fmt = (x) => stdlib.formatCurrency(x, 4);

const getBalance = async (who) => fmt(await stdlib.balanceOf(who));

const beforeAlice = await getBalance(accAlice);

const beforeBob = await getBalance(accBob);

const ctcAlice = accAlice.contract(backend);

const ctcBob = accBob.contract(backend, ctcAlice.getInfo());

This part of code initializes Reach stdlib and declares our players blockchain accounts. It declares starting balance of accounts as well as Alice deploying the contract.

"parseCurrency" function is used to convert the crypto amount to micro algos.

const HAND = [0, 1, 2, 3, 4, 5];

const VOCAL = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

const OUTCOME = ['Bob wins', 'Draw', 'Alice wins'];

Values which represent number of raised fingers(const HAND), players guess of number of raised fingers(const VOCAL), and possible outcomes are declared.

const Player = (Who) => ({

...stdlib.hasRandom,

getHand: async () => { // <-- async now

const hand = Math.floor(Math.random() \* 6);

console.log(`${Who} raised ${HAND[hand]} fingers`);

if ( Math.random() <= 0.01 ) {

for ( let i = 0; i < 10; i++ ) {

console.log(` ${Who} takes their sweet time sending it back...`);

await stdlib.wait(1);

}

}

return hand;

},

Players number of raised fingers is randomly choosen using Math.Random and corresponding text is being print out in console.

getVocal: async (hand) => {

// vocal should be greater than or equal to number of fingers thrown

const vocal= Math.floor(Math.random() \* 6) + HAND[hand];

// occasional timeout

if ( Math.random() <= 0.01 ) {

for ( let i = 0; i < 10; i++ ) {

console.log(` ${Who} takes their sweet time sending it back...`);

await stdlib.wait(1);

}

}

console.log(`${Who} guessed total of ${vocal}`);

return vocal;

},

seeWinning: (winningNumber) => {

console.log(`Actual total fingers raised: ${winningNumber}`);

},

seeOutcome: (outcome) => {

console.log(`${Who} saw outcome ${OUTCOME[outcome]}`);

},

informTimeout: () => {

console.log(`${Who} observed a timeout`);

},

});

Methods which enable player to guess a certain number of fingers(getVocal), see the winner and outcome of the game are being declared.

await Promise.all([

ctcAlice.p.Alice({

...Player('Alice'),

wager: stdlib.parseCurrency(5),

deadline: 10,

}),

ctcBob.p.Bob({

...Player('Bob'),

acceptWager: (amt) => {

console.log(`Bob accepts the wager of ${fmt(amt)}.`); //Bob accepts wager

},

}),

]);

Backends for Alice and Bob are deployed. Alice sets the wager, and Bob accepts it. Deadline is set to 10 seconds.

const afterAlice = await getBalance(accAlice);

const afterBob = await getBalance(accBob);

console.log(`Alice went from ${beforeAlice} to ${afterAlice}.`);

console.log(`Bob went from ${beforeBob} to ${afterBob}.`);

Variables afterAlice and afterBob obtain the new values of players accounts balances, and are then displayed in console.