Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Национальный исследовательский университет «МЭИ»

Кафедра ВМСС

**Типовой расчёт**

**«Сложный смарт-контракт»**

**Курс: Введение в технологию блокчейн**

Группа: А-07м-23

Выполнил: Кретов Н.В.

Проверил: Оцоков Ш.А.

Москва 2024

**СОДЕРЖАНИЕ**

[1. Цель 3](#_Toc167728996)

[2. Задание 3](#_Toc167728997)

[3. Разработка 3](#_Toc167728998)

[ЗАКЛЮЧЕНИЕ 6](#_Toc167728999)

[ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ 7](#_Toc167729000)

1. **Цель**

Главной задачей данной работы является закрепление навыков разработки смарт-контрактов на языке программирования Solidity, полеченных в период освоения дисциплины «Введение в технологию блокчейн».

1. **Задание**

Разработать сложный смарт-контракт на языке Solidity в рамках выбранной предметной области.

В качестве предметной области для смарт-контракта была выбрана настольная игра «Бункер». По нарративу игры на Земле произошел апокалипсис и десять выживших (участников игры) пытаются определить кто сможет попасть в бункер, где смогут пережить катастрофу (кол-во мест в бункере ограничено, так что все попасть туда не могут).

Каждый игрок получает случайным образом свои игровые роли (профессии) и пытается привести аргументы, почему он является полезным для выживания группы и почему он должен попасть в бункер. После чего происходит голосование за самого бесполезного для дальнейшего выживания по мнению всех оставшихся игроков участника.

1. **Разработка**

Contract Bunker содержит следующие переменные:

* address payable public owner – адрес владельца контракта;
* address payable[10] public players – адреса игроков;
* address payable[5] public winners – адреса победителей;
* uint public stateOfGame – состояние игры (1-не начата, 2-получение ролей, 3-открытие характеристик, 4-голосование, 5-закончена);
* uint public countRegPlayers – кол-во зарегистрированных игроков;
* uint public countCurrentPlayers – кол-во действующих сейчас игроков;
* uint public countAlivePlayers – кол-во оставшихся игроков;
* uint price - стоимость регистрации в игре;
* mapping(uint => string) stateOfGameMessages – сообщения о состоянии игры;
* mapping(address => string) playersStatusMessages – сообщения о статусе игрока;
* mapping(address => uint) playersStatus – статус игрока (1-зарегистрирован, 2-имеет роль, 3-открыл характеристику, 4-проголосовал, 5-выбыл);
* mapping(address => string) playersRole – роль игрока;
* mapping(address => uint) playersNumber – номер игрока;
* mapping(address => uint) amountOfVotes – количество голосов за изгнание игрока;
* mapping(uint => string) roles – список ролей.

Для корректной работы смарт-контракта были написаны модификаторы:

* modifier NewGameAllowed();
* modifier OnlyOwner();
* modifier RegistrationAllowed();
* modifier GetRoleAllowed();
* modifier MakeTurnAllowed();
* modifier VoteAllowed().

В рамках контракта были реализованы следующие функции:

* function changePrice(uint \_price) public OnlyOwner() – доступная только владельцу контракта функция для изменения стоимости регистрации участников в игре (доступна только до начала регистрации);
* function NewGame() public NewGameAllowed();
* function Registration() public payable RegistrationAllowed() returns (string memory) – функция регистрации участников игры;
* function GetRole() public GetRoleAllowed returns (string memory) – функция получения зарегистрированными игроками своей роли;
* function RandomNumber(uint blockNumber, uint endOfRange) private view returns (uint) – вспомогательная функция, генерирующая случайное целое не отрицательное число в диапазоне от 0 до указанной границы;
* function MakeTurn(string memory characteristic) public MakeTurnAllowed returns (string memory) – функция для совершения хода (открытия своей роли другим игрокам), доступная только пользователям, получившим игровую роль;
* function ViewRoles() public view returns (string[] memory) – функция, показывающая все текущие открытые роли участников игры;
* function Vote(uint number) public payable VoteAllowed returns (string memory) – функция голосования;
* function GetStateOfGame() public view returns (string memory) – функция для выяснения текущего состояния игры;
* function ToAsciiString(address x) internal pure returns (string memory);
* function char(bytes1 b) internal pure returns (bytes1 c);
* function getPlayers() public view returns (string[10] memory) – функция, возвращающая список участников игры.

С полным листингом смарт-контракта можно ознакомиться в приложении А.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения расчетного задания был разработан смарт-контракт на языке программирования Solidity демонстрирующий функционал настольной игры «Бункер». Было произведено ручное тестирование контракта, в ходе которого ошибок в работе обнаружено не было. Были закреплены навыки программирования на языке программирования Solidity.

# **ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ**

**Код смарт-контракта**

// SPDX-License-Identifier: MIT

pragma solidity >=0.7.12 <0.9.0;

contract Bunker

{

address payable public owner; //адрес владельца контракта

address payable[10] public players; //адреса игроков

address payable[5] public winners; //адреса победителей

uint public stateOfGame; //состояние игры (1-не начата, 2-получение ролей, 3-открытие характеристик, 4-голосование, 5-закончена)

uint public countRegPlayers; //кол-во зарегистрированных игроков

uint public countCurrentPlayers; //кол-во действующих сейчас игроков

uint public countAlivePlayers; //кол-во оставшихся игроков

uint price; //стоимость регистрации в игре

mapping(uint => string) stateOfGameMessages; //сообщения о состоянии игры

mapping(address => string) playersStatusMessages; //сообщения о статусе игрока

mapping(address => uint) playersStatus; //статус игрока (1-зарегистрирован, 2-имеет роль, 3-открыл характеристику, 4-проголосовал, 5-выбыл)

mapping(address => string) playersRole; //роль игрока

mapping(address => uint) playersNumber; //номер игрока

mapping(address => uint) amountOfVotes; //количество голосов за изгнание игрока

mapping(uint => string) roles; //список ролей

constructor()

{

owner = payable(msg.sender); //владелец тот, кто вызвал конструктор

stateOfGame = 1; //игра не начата

countRegPlayers = 0; //нет зарегистрированных игроков

countCurrentPlayers = 0; //действующих игроков в начале игры нет

countAlivePlayers = 0; //нет оставшихся игроков

price = 1;

stateOfGameMessages[1] = "Game is not started";

stateOfGameMessages[2] = "Players discover their characteristics";

stateOfGameMessages[3] = "Players vote";

stateOfGameMessages[4] = "Game is finished, winners: ";

roles[0] = "Unemployed";

roles[1] = "Doctor";

roles[2] = "Engineer";

roles[3] = "Businessman";

roles[4] = "Policeman";

roles[5] = "Postman";

roles[6] = "Firefighter";

roles[7] = "Politician";

roles[8] = "Builder";

roles[9] = "Clown";

}

modifier NewGameAllowed()

{

require(stateOfGame == 1, "You cannot start a new game before the current one is completed"); //нельзя начать новую игру до завершения текущей

require(stateOfGame == 5, "You cannot start a new game before the current one is completed"); //нельзя начать новую игру до завершения текущей

\_;

}

modifier OnlyOwner()

{

require(msg.sender == owner, "You are not the owner of the contract");

require(countRegPlayers == 0, "You cannot change the registration fee, the game has already started");

\_;

}

modifier RegistrationAllowed()

{

require(stateOfGame == 1, "Registration closed!"); //если игра уже началась или закончилась, то регистрация закрыта

require(countRegPlayers < 10, "Registration closed!"); //если уже зарегистрированы 10 игроков, то регистрация закрыта

require(msg.value >= price, "Not enough money to participate in the game");

\_;

}

modifier GetRoleAllowed()

{

require(stateOfGame == 2, "This is not the stage of role!"); //если сейчас не этап получения ролей, то получить роль нельзя

require(playersStatus[msg.sender] == 1, "You are not registered!"); //если игрок не зарегистрирован или уже имеет роль, то получить роль нельзя

\_;

}

modifier MakeTurnAllowed()

{

require(stateOfGame == 3, "This is not the stage of discovering characteristics!"); //если сейчас не этап открытия характеристик, то ходить нельзя

require(playersStatus[msg.sender] == 2, "You have already opened the characteristics!"); //если у игрока нет роли или он уже открыл характеристику, то ходить нельзя

\_;

}

modifier VoteAllowed()

{

require(stateOfGame == 4, "This is not the voting stage!"); //если сейчас не этап голосования, то ходить нельзя

require(playersStatus[msg.sender] == 2, "You have already voted!"); //если у игрока нет роли или он уже проголосовал, то ходить нельзя

\_;

}

function changePrice(uint \_price) public OnlyOwner

{

price = \_price;

}

function NewGame() public NewGameAllowed

{

stateOfGame = 1; //игра не начата

countRegPlayers = 0; //нет зарегистрированных игроков

countCurrentPlayers = 0; //нет действующих сейчас игроков

countAlivePlayers = 0; //нет оставшихся игроков

}

function Registration() public payable RegistrationAllowed() returns (string memory)

{

if(owner.send(price))

{

players[countRegPlayers] = payable(msg.sender);

playersStatusMessages[msg.sender] = "Is registrated"; //после регистрации статус игрока "зарегистрирован"

playersStatus[msg.sender] = 1; //после регистрации статус игрока 1 - "зарегистрирован"

amountOfVotes[msg.sender] = 0; //после регистрации голосов за изгнание этого игрока 0

playersNumber[msg.sender] = countCurrentPlayers;

countCurrentPlayers++;

countRegPlayers++;

if(countRegPlayers == 10) //если набралось 10 игроков

{

countCurrentPlayers = 0;

stateOfGame = 2; //то запускаается этап получение ролей

}

string memory message = "You are successfully registered";

return message;

}

else

{

string memory message = "Failed to pay for registration";

return message;

}

}

function GetRole() public GetRoleAllowed returns (string memory)

{

playersRole[msg.sender] = roles[RandomNumber(playersNumber[msg.sender], 10)];

countCurrentPlayers++;

countAlivePlayers++;

if(countAlivePlayers == 10) //если все 10 игроков получили роли

{

countCurrentPlayers = 0;

stateOfGame = 3; //то запускаается этап открытия характеристик

}

string memory message = string.concat("You got the role: ", playersRole[msg.sender]);

return (message);

}

function RandomNumber(uint blockNumber, uint endOfRange) private view returns (uint)

{

uint random\_number = uint(blockhash(blockNumber)) % endOfRange;

return random\_number;

}

function MakeTurn(string memory characteristic) public MakeTurnAllowed returns (string memory)

{

countCurrentPlayers++;

playersStatus[msg.sender] = 3;

string memory message = string.concat("You showed to players your characteristic: ", characteristic);

if(countCurrentPlayers == countAlivePlayers)

{

countCurrentPlayers = 0;

stateOfGame = 4;

for(uint i = 0; i < 10; i++)

{

if (playersStatus[players[i]] < 5)

{

playersStatus[players[i]] = 2;

}

}

}

return (message);

}

function ViewRoles() public view returns (string[] memory)

{

string[] memory messages;

for(uint i = 0; i < 10; i++)

{

if(playersStatus[players[i]] < 5)

{

messages[i] = string.concat("Player: ", ToAsciiString(players[i]), ", characteristic: ", playersRole[players[i]]);

}

}

return (messages);

}

function Vote(uint number) public payable VoteAllowed returns (string memory)

{

string memory message;

if(number < 1 || number > 10)

{

message = string.concat("There is no player with this number");

return (message);

}

if(playersStatus[players[number]] == 5)

{

message = string.concat("This player has already dropped out");

return (message);

}

countCurrentPlayers++;

playersStatus[msg.sender] = 4;

address player = players[number];

amountOfVotes[player]++;

if(countCurrentPlayers == countAlivePlayers)

{

uint max = amountOfVotes[players[0]];

address banished = players[0];

uint count;

for(uint i = 1; i < 10; i++)

{

if(amountOfVotes[players[i]] > max)

{

max = amountOfVotes[players[i]];

banished = players[i];

count = 1;

}

if(amountOfVotes[players[i]] == max)

{

count++;

}

}

if(count > 1)

{

address[] memory pretendents;

for(uint i = 1; i < 10; i++)

{

uint j = 0;

if(amountOfVotes[players[i]] == max)

{

pretendents[j] = players[i];

j++;

}

}

uint randIndex = RandomNumber(playersNumber[msg.sender], pretendents.length);

banished = pretendents[randIndex];

}

playersStatus[banished] = 5;

amountOfVotes[banished] = 0;

countAlivePlayers--;

countCurrentPlayers = 0;

for(uint i = 0; i < 10; i++)

{

amountOfVotes[players[i]] = 0;

if (playersStatus[players[i]] < 5)

{

playersStatus[players[i]] = 2;

}

}

}

if(countAlivePlayers == 5)

{

string[] memory winnersString;

uint j = 0;

for(uint i = 0; i < 10; i++)

{

if(playersStatus[players[i]] < 5)

{

winners[j] = players[i];

winnersString[j] = ToAsciiString(winners[j]);

winners[j].transfer(owner.balance/5);

j++;

}

}

stateOfGame = 5;

message = string.concat("The game is over, winners: ", winnersString[0], " ", winnersString[1], " ", winnersString[2], " ", winnersString[3], " ", winnersString[4]);

return (message);

}

message = "Your vote has been counted, wait for the end of the voting";

return (message);

}

function GetStateOfGame() public view returns (string memory)

{

string memory message;

if(stateOfGame != 5)

{

message = stateOfGameMessages[stateOfGame];

}

else

{

string[5] memory addrs;

for(uint i; i < 6; i++)

{

addrs[i] = ToAsciiString(winners[i]);

}

message = string.concat(stateOfGameMessages[stateOfGame], addrs[0], ", ", addrs[1], ", ", addrs[2], ", ", addrs[3], ", ", addrs[4]);

}

return message;

}

function ToAsciiString(address x) internal pure returns (string memory)

{

bytes memory s = new bytes(40);

for (uint i = 0; i < 20; i++)

{

bytes1 b = bytes1(uint8(uint(uint160(x)) / (2\*\*(8\*(19 - i)))));

bytes1 hi = bytes1(uint8(b) / 16);

bytes1 lo = bytes1(uint8(b) - 16 \* uint8(hi));

s[2\*i] = char(hi);

s[2\*i+1] = char(lo);

}

return string(s);

}

function char(bytes1 b) internal pure returns (bytes1 c)

{

if (uint8(b) < 10) return bytes1(uint8(b) + 0x30);

else return bytes1(uint8(b) + 0x57);

}

function getPlayers() public view returns (string[10] memory)

{

string[10] memory messages;

for(uint i = 0; i < 10; i++)

{

messages[i] = string.concat("Player: ", ToAsciiString(players[i]), ", status: ", playersStatusMessages[players[i]]);

}

return (messages);

}

}