



Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Predmet

– Digitálne meny a Blockchain –

- Dokumentácia -

Smart kontraktový systém pre námorne bitky

Ak. Rok : 2021/2022, letný semester

Cvičiaci:

Ing. Viktor Valaštín

Študent:

Martin Rudolf 97029



Bratislava, 2022.

Obsah:

1	Implementačné prostredie	2
2	Doimplementované časti kódu.....	2
a.	contract_starter.sol.....	2
i.	store_bid():	2
ii.	clear_state()	2
iii.	store_board_commitment()	3
iv.	check_one_ship()	3
v.	claim_win()	3
vi.	forfeit()	3
vii.	accuse_cheatin()	4
viii.	claim_opponent_left()	4
ix.	handle_timeout()	4
x.	claim_timeout_winnings()	4
b.	BattleshipPlayer.js	4
i.	place_bet().....	4
ii.	Initialize_board().....	5
iii.	recieve_response_to_guess().....	5
iv.	accuse_timeout()	5
v.	handle_timeout_accusation()	5
vi.	claim_timout_winnings()	5
vii.	accuse_cheating()	5
viii.	claim_win()	5
ix.	forfeit_game().....	6
3	Testovanie.....	6
4	Test coverage.....	7
5	Security analýza	8
6	Otázky	9

1 Implementačné prostredie

Ako implementačné prostredie som si zvolil Virtual Studio Code, pomocou ktorého som doimplementoval požadované funkcie v zdrojovom súbore `contract_starter.sol` v programovacom jazyku Solidity a v súbore `BattleshipPlayer.js` v jazyku JavaScript. Pre kompiláciu, deploy a testovanie som použil nástroj Truffle teda som si pre dané zadanie vytvoril truffle projekt ktorý bol pripojený na Ganache. Pre jednoduchšie overovanie funkcií smart kontraktu som využíval aj Remix IDE.

2 Doimplementované časti kódu

a. `contract_starter.sol`

i. `store_bid()`:

Táto funkcia priradí hodnoty stavovým premenným reprezentujúce adresy hráčov a uloží čiastky ktoré poslali na kontrakt. Ak hodnota stavovej premennej `player1` je default tak sa jej priradí adresa ktorá komunikuje s touto funkciou a namapuje sa na tu adresu čiastka ktorú daná adresa poslala na kontrakt. Inak sa to iste vykoná pre stavovú premennú `Player2`.

```
function store_bid() public payable{
    if (player1 == address(0)){
        player1 = msg.sender;
        playersBids[player1] = msg.value;
    }
    else if(player2 == address(0)){
        player2 = msg.sender;
        playersBids[player2] = msg.value;
    }
    if(player1 != address(0) && player2 != address(0)){
        require(playersBids[player2] >= playersBids[player1], "Player1 bids more than Player2");
        uint valueRefund = playersBids[player2] - playersBids[player1];
        if (valueRefund > 0){
            (bool sent, /*bytes memory data*/) = player2.call{value: address(this).balance}("");
            require(sent, "Failed to send Ether");
        }
    }
}
```

ii. `clear_state()`

Pomocou tejto funkcie inicializujeme stavové premenné na default hodnoty a tak pripravíme kontrakt na novú hru.

```
function clear_state() internal {
    require(gameOver, "game is still in session");
    playersBids[player1] = 0;
    playersBids[player2] = 0;
    provedHits[player1] = 0;
    provedHits[player2] = 0;
    provedShips[player1] = 0;
    provedShips[player2] = 0;
    boardCommitments[player1] = 0;
    boardCommitments[player2] = 0;
    player1 = address(0);
    player2 = address(0);
    gameOver = false;
}
```

iii.store_board_commitment()

Funkcia najprv overí či adresa ktorá volá funkciu patrí niektoremu z hráčov ak ano tak sa na ňu namapuje koreň mekrleho stromu, čiže počiatočný stav hracej dosky.

```
function store_board_commitment(bytes32 merkle_root) public {
    require(msg.sender == player1 || msg.sender == player2, "not valid adress");
    boardCommitments[msg.sender] = merkle_root;
}
```

iv.check_one_ship()

Funkcia pomocou už implementovanej funkcie verify_opening() overí na základe vstupných argumentov (opening_nonce, proof teda hashe susedov, index daného listu, a počiatočný stav hracej dosky daného hráča). Ak je adresa ktorá interaguje s kontraktom rovná owner, vstupnému argumentu funkcie, a zároveň pomocná funkcia vrátila true, tak sa zvýši counter namapovaný na adresu volajúceho o 1. Teda overím pozíciu mojej lode. To isté sa udeje keď je owner protivník ale zvýši sa counter počítajúci zásahy.

```
function check_one_ship(bytes memory opening_nonce, bytes32[] memory proof,
    uint256 guess_leaf_index, address owner) public returns (bool result) {
    checked_ship = verify_opening(opening_nonce, proof, guess_leaf_index, boardCommitments[owner]);
    if(owner == msg.sender && result){
        provedShips[msg.sender] += 1;
    }
    else if(result){
        provedHits[msg.sender] += 1;
    }
    result = checked_ship;
    return result;
}
```

v.claim_win()

Pri volaní tejto funkcií sa najprv overí, či hráč ktorý učinil volanie má počítadlá, ktoré nesú informáciu o počte trafených a počte položených lodí sú rovné 10. Ak áno tak vyplatí sa mu výhra, nastaví sa stavová premenná gameOver na true a zavolá sa funkcia clear_state().

```
function claim_win() public {
    require(provedShips[msg.sender] == 10 && provedHits[msg.sender] == 10, "not enough hits or placed ships");
    (bool sent, /*bytes memory data*/) = msg.sender.call{value: address(this).balance}("");
    require(sent, "Failed to send Ether");
    gameOver = true;
    clear_state();
}
```

vi.forfeit()

Overí či adresa ktorá volá danú funkciu a adresa zo vstupného argumentu funkcie patria hráčom. Ak áno tak sa výhra odošle oponentovi a nastaví sa kontrakt pre novú hru.

```
function forfeit(address payable opponent) public {
    require((msg.sender == player1 || msg.sender == player2) && (opponent == player1 || opponent == player2), "invalid address");
    (bool sent, /*bytes memory data*/) = opponent.call{value: address(this).balance}("");
    require(sent, "Failed to send Ether");
    gameOver = true;
    clear_state();
}
```

vii. `accuse_cheatin()`

Rovnako ako v predchádzajúcich funkciách overím či interagujúca adresa patrí jednému z hráčov a skontrolujem daný ťah či bol podvod pomocou už spomenutej funkcie `verify_opening()`. Ak pomocná funkcia vráti `false` tak sa výhra pripíše na adresu ktorá zavolała túto funkciu.

```
function accuse_cheating(bytes memory opening_nonce, bytes32[] memory proof,
    uint256 guess_leaf_index, address owner) public returns (bool result) {
    require((msg.sender == player1 || msg.sender == player2) && (owner == player1 || owner == player2));
    result = verify_opening(opening_nonce, proof, guess_leaf_index, boardCommitments[owner]);
    if (!result){
        (bool sent, /*bytes memory data*/) = msg.sender.call{value: address(this).balance}("");
        require(sent, "Failed to send Ether");
    }

    return result;
}
```

viii. `claim_opponent_left()`

Skontroluje či obviňovaný hráč už nie je náhodou obvinený a zároveň či obviňovateľ nie je rovnako obvinený. Spustí sa časovač na jednu minútu a emitne sa event hovoríaci že hráč je neprítomný alebo zdržuje.

```
function claim_opponent_left(address opponent) public {
    require(!playerLeft[opponent] && !playerLeft[msg.sender] && (opponent == player1 || opponent == player2), "Invalid address or you have to handle a timeout accusation");
    timerStart = now;
    timerEnd = timerStart + 60;
    playerLeft[opponent] = true;
    emit OpponentLeft(opponent);
}
```

ix. `handle_timeout()`

Skontroluje či čas už nevypršal, ak nie tak nastaví hodnotu namapovanú na obvineného adresu na `false` čo zastaví časovač.

```
function handle_timeout(address payable opponent) public {
    require(now <= timerEnd && (msg.sender == player1 || msg.sender == player2), "time has run out or invalid address");
    playerLeft[msg.sender] = false;
}
```

x. `claim_timeout_winnings()`

Ak je oponent stále obvinený a časovač ubehol tak sa hráčovi vyplátí výhra a nastaví sa kontrakt na novú hru.

```
function claim_timeout_winnings(address opponent) public {
    require(playerLeft[opponent] && now >= timerEnd, "timer has not run out");
    (bool sent, /*bytes memory data*/) = msg.sender.call{value: address(this).balance}("");
    require(sent, "Failed to send Ether");
    gameOver = true;
    clear_state();
}
```

b. BattleshipPlayer.js

i. `place_bet()`

Interaguje s kontraktom, konkrétne s funkciou `store_bid()` na ktorú pošle hodnotu stávky vo wei a adresu hráča.

ii.Initialize_board()

Doimplementovaná časť rovnako len interaguje so smart kontrakt funkciou `store_board_commit`.

iii.recieve_response_to_guess()

Do tejto funkcie bola doimplementovaná časť kódu, ktorý vytvorí objekt s hodnotami `proof`, `index`, a `opening_nonce` ktoré potrebujeme na overenie pravosti lode. Uložím si tento objekt ako `last_guess` a ešte skontrolujem príznak `opening`, ak je `true` tak si vytvorený objekt uloží do poľa zničených lodí.

```
var guess = {
  "proof": proof,
  "index": i*BOARD_LEN+j,
  "opening_nonce": web3.utils.fromAscii(JSON.stringify(opening) + JSON.stringify(nonce))
}
this.last_guess = guess;
if(opening){
  var des_ship = guess;
  this.des_ships.push(des_ship);
}
```

iv.accuse_timeout()

Funkcia len jednoducho volá funkciu smart kontraktu `claim_opponent_left()`.

v.handle_timeout_accusation()

Zavolá metódu smart kontraktu `handle_timeout()` a vráti `true` ak je koniec hry.

vi.claim_timeout_winnings()

Interaguje s funkciou `claim_timeout_winnings` smart kontraktu

vii.accuse_cheating()

Funkcia využíva vyššie spomínaný objekt `last_guess`, ktorý ak je definovaný tak sa odošlu jeho atribúty do funkcie smart kontraktu `accuse_cheating()`.

viii.claim_win()

Ak sa počet hráčových lodí a počet zostrelených lodí rovná 10, tak sa pravosť každej z nich overí na smart kontrakte pomocou funkcie `check_one_ship()`. Následne sa zavolá funkcia `claim_win` zo smart kontraktu.

```

if(this.des_ships.length == 10 && this.my_ships.length == 10){
  this.des_ships.forEach((ship) => {
    Battleship.methods.check_one_ship(ship.opening_nonce, ship.proof, ship.index, this.opp_addr).send({
      from: this.my_addr
    });
  });
  this.my_ships.forEach((my_ship) =>{
    let i = my_ship[0];
    let j = my_ship[1];
    let index = i * BOARD_LEN + j;
    let nonce = this.nonces[i][j];
    let opening = this.my_board[i][j];
    let opening_nonce = web3.utils.fromAscii(JSON.stringify(opening) + JSON.stringify(nonce));
    let proof = get_proof_for_board_guess(this.my_board, this.nonces, [i, j]);

    Battleship.methods.check_one_ship(opening_nonce, proof, index, this.my_addr).send({
      from: this.my_addr
    });
  });
  Battleship.methods.claim_win().send({
    from: this.my_addr
  });
}
else{
  console.log("not enough ships or hits");
}
if(this.name == 'player1') {
  end_game_ui(this.name, 0);
}
else {
  end_game_ui(this.name, 1);
}
}

```

ix.forfeit_game()

Interakcia s funkciou smartkontraktu forfeit()

3 Testovanie

Prostredie na testovanie som si zvolil truffle, kde som si v adresary testy vytvoril súbor testBattleship.test.js v ktorom som implementoval testy.

Prvý test kontroluje hodnotu stavovej premennej gameOver. Ďalší kontroluje či su inicializované správne default adresy hráčov. V treťom sa kontroluje či funkcia check_one_ship vráti správnu hodnotu pre nevalidnú loď. Nasledujúci kontroluje, či sa zmení counter, po skontrolovaní nevalidnej lode. Posledný test kontroluje správnosť hodnoty stávky poslanej na smart kontrakt.

```

const Battleship = artifacts.require("Battleship");

contract("Battleship", (accounts) => {
  let battleship;
  let expectedGame;

  before(async () => {
    battleship = await Battleship.deployed();
  });

  describe("Check if game over", async () => {
    it("check if game is over", async () => {
      const resultGame = await battleship.is_game_over({from: accounts[0]});
      expectedGame = false;
      assert.equal(resultGame, expectedGame, "At the beginning can not be over");
    });

    it("Check addresses both players", async () => {
      const players = await battleship.get_players({from: accounts[0]});

      assert.equal(players['0'], '0x0000000000000000000000000000000000000000000000000000000000000000', "The address of the player1 should be address(0) when bids was not set.");
      assert.equal(players['1'], '0x0000000000000000000000000000000000000000000000000000000000000000', "The address of the player2 should be address(0) when bids was not set.");
    });

    it("check one ship if is valid", async () => {
      await battleship.check_one_ship("0x7472756532303532333331333034", [], 0, accounts[1], {from: accounts[0]});
      const result = await battleship.get_checked_ship({from: accounts[0]});

      assert.equal(result, false, "Passed ship should not be valid");
    });

    it("should get count of proved ships", async () => {
      await battleship.check_one_ship("0x7472756532303532333331333034", [], 0, accounts[0], {from: accounts[0]});
      const count = await battleship.get_ships_count({from: accounts[0]});

      assert.equal(BigInt(count), BigInt(0), "Proved ships should be 0 if the game have not start yet");
    });

    it("Store a bid", async () => {
      await battleship.store_bid({from: accounts[0], value: 1*10**18});
      const bid = await battleship.get_player_bid({from: accounts[0]});
      assert.equal(BigInt(bid), BigInt(1*10**18), "Proved ships should be 0 if the game have not start yet");
    });
  });
});

```

4 Test coverage

Na test coverage som použil nástroj dostupný [tu](#). Priemerný test coverage je 25,9.

File	% Stmts	% Branch	% Funcs	% Lines
contracts/ contract_starter.sol	24.69	9.09	47.06	22.62
All files	24.69	9.09	47.06	22.62

5 Security analýza

Bezpečnostní analýzu prebehla v remixe s pluginom SOLIDITY STATIC ANALYSIS s takýmito výsledkami:

Check-effects-interaction: ✕

Potential violation of Checks-Effects-Interaction pattern in Battleship.claim_win(): Could potentially lead to re-entrancy vulnerability.

[more](#)

Pos: 110:4:

Inline assembly: ✕

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.

[more](#)

Pos: 40:8:

Block timestamp: ✕

Use of "now": "now" does not mean current time. "now" is an alias for "block.timestamp". "block.timestamp" can be influenced by miners to a certain degree, be careful.

[more](#)

Pos: 150:21:

Low level calls: ✕

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

[more](#)

Pos: 112:45:

Guard conditions: ✕

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 65:11:

Gas costs: ✕

Gas requirement of function Battleship.store_bid is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 46:4:

Constant/View/Pure functions: ✕

ECDSA.recover(bytes32,bytes) : Is constant but potentially should not be.

[more](#)

Pos: 26:4:

6 Otázky

Predpokladajme, že hráč 1 umiestni na hraciu plochu menej ako 10 lodí, ale nikdy neklame o zásahoch alebo minutiach. Môže hráč 2 dostať svoje peniaze späť? Prečo áno, prečo nie?

Nemôže lebo náš systém nepočíta s takouto situáciou. Jediná možnosť ako ich môže dostať späť je, že sa súper vzdá alebo nestihne zareagovať na timeout.

Prečo neobmedzujeme hráčov v umiestňovaní viac ako 10 lodí na ich dosku?

Lebo keď je pravidlo, že si ma hráč umiestniť práve 10 lodí a keďže hrá o peniaze, nebude riskovať, nemalo by to pre neho žiaden zmysel. Aj keby ich mal viac ako 10 a súper by vystrelal 10 tak súper si môže nárokovať na výhru ak ma položených svojich 10 lodí.

Nemáme mechanizmus, ktorý by hráča obviňoval z umiestnenia menej ako 10 lodí na hraciu plochu. Ako by ste ho vedeli implementovať?

Hráč ktorý ma menej ako 10 lodí nikdy nevyhrá a tak jeho súper sa môže dostať do stavu keď ostanú len tie políčka prázdne kde musia byť lode (keď uvažujem že nepodvádzal). Ak vystrelím a odpoveď je že som netrafil tak ho môžem obviniť z toho že položil menej lodí ako 10. V jedenej premennej si počítam výstrely, ak 30 výstrelův a z toho som 4 trafil, tak ostatné 6 políčka musia byť lode. Ak po nasledujúcom výstrele je odpoveď že som netrafil obviním protivníka.

Napadajú vám scenáre útoku alebo konkrétne zraniteľné miesta v niektorom z uvedených kódov, proti ktorým by ste sa nedokázali ubrániť?

Nevidím potencionálnu možnosť útoku v žiadnom z uvedených kódov.

7 Záver

V rámci tohto zadania som si osvojil praktické zručnosti s nástrojmi na vývoj decentralizovaných aplikácií (DAPP). Naučil som sa najmä aká dôležitá je bezpečnosť, a efektívnosť takto vyvíjanej aplikácie. Nikto predsa nebude používať aplikáciu v ktorá spotrebuje veľké množstvo gassu a tak produkuje veľké fička.

Oboznámil som sa taktiež s nástrojmi pre vývoj smart kontraktov, akými sú programovací jazyk Solidity, Remix IDE na jednoduché testovanie a overovanie funkčnosti smart kontrakt funkcií. Nástroj Truffle mi poslúžil na manažovanie, testovanie a deployovanie aplikácie na lokálnu eth sieť ktorú spravoval nástroj Ganache.

Pri tomto zadani som sa veľa naučil a hodnotím ho kladne. Aj keď s JavaScriptom som nebol veľký kamarát, no musím sa priznať, po tej troške čo som s ním pracoval pri tomto zadani mu prichádzam na chuť. Avšak ako veľké plus vnímam to, že som

porozumel ako fungujú DAPP a ako prebieha komunikácia medzi frontendom a kontraktom deploynutým na blockchaine.