SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

# Zadanie 2 - Smart kontraktový systém pre námorné bitky

Adrián Vančo
ID: 103171
Predmet: Digitálne meny a blockchain

# Zadanie

## Otázky pre zadanie a odpovede

1. 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?

   **Odpoveď**: Ak predpokladáme, že hráč 1 bude hrať (nebude donekonečna zdržiavať) a hra bude plynúť tak hráč 2 môže vyhrať. A môže vyhrať spôsobom že vystrelil na každé hracie políčko (v našom prípade 36 políčok) a následne zavolá claim_win() v kontrakte, ktorá obsahuje podmienku, ktorá tento prípad ošetruje, ale treba pridať podmienku, ktorá dovolí volanie na kontrakt v BattleshipPlayer.js.

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

   **Odpoveď**: Hráč, ktorý umiestni viac ako 10 lodí má vyššiu náročnosť a je pre neho ťažšie vyhrať. Beriem to ako funkcionalitu navyše. Avšak protihráč musí po 10 výstreloch vždy volať claim_win(), aby neprešvihol výhru nakoľko v kontrakte sa očakáva presne 10 zásahov.

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

   **Odpoveď**: V hre by som pridal ďalší stav hry, do ktorého by sa prešlo po uložení merkle_root pre každého hráča vo funkcií store_board_commitment(). V tomto pridanom stave by hráči museli do kontraktu poslať svoje opening_nonces, proofs. Kontrakt by mal funkciu pre tento stav hry, ktorá by prebehla všetky listy a zrátala počet, na koľkých listoch sa nachádzajú lode.

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

   **Odpoveď**: Hráč nemusí nikdy vykonať ťah, môže neustále odpovedať na claim_opponent_left() a sám môže obviňovať protihráča a požadovať výhru. Tento útok môže byť automatizovaný a protihráč nemá šancu vyhrať.

## Popis doimplementovaných častí

### contact_starter.sol

Hra používa na reprezentáciu svojho stavu game_state typu enum GameStates {OPEN, PLACING, RUNNING, FINISHED, DONE}. Na začiatku je inicializovaná na OPEN.
Názvy premenných v kontrakte sú samo popisujúce.

Kontrakt má zadefinované dva eventy:

```
event TimeAccusion(address plaintiff, address  accused, uint time_of_accusion);
event CheckOneShip(address caller, address board_owner, bool telling_truth);
```

**TimeAccusion()** sa používa ak hráč obvinil druhého hráča vo funkcii **claim_opponent_left();**
**CheckOneShip()** sa používa pri kontrole lode vo funkcii **check_one_ship();**

**store_bid()** – uložím si dvoch hráčov, a ich ponuky a zmením stav hry na PLACING.

**clear_state()** – vyresetujem všetky premenné a dám hru do stavu DONE.

**store_board_commitment()** – uloží merkle root pre daného hráča. Ak obaja predložili merkle root zmení sa stav hry na RUNNING a hráči môžu hrať.

**check_one_ship()** – zistím si oponent neklamal ďalej či v opening_nonce sa nachádza loď (porovnám prvé 4 bajty). Ak tam bola loď a neklamal alebo tam loď nebola a klamal tak si pridám loď do poľa unikátnych lodí a pripočítam zásah ak sa loď ešte nenachádzala v poli.

**claim_win()** – hráč si môže výhru nárokovať ak trafil 10 unikátnych lodí a sám ma na svojej mape 10 lodí, prípadne ak protihráč nemá na mape 10 lodí a hráč vystrelil aspoň raz na každé pole mapy.

**forfeit()** – pomocou tejto funkcie sa môže hráč vzdať, ak tak učiní tým výhra pripadne protihráčovi a hra skončí.

**accuse_cheating()** – ak protihráč v predchádzajúcom ťahu klamal hráč si môže nárokovať výhru.

**claim_opponent_left()** – ak si hráč myslí že protihráčovi trvá príliš dlho ťah, môže ho obviniť zo zdržiavania a po zavolaní sa uloží čas obvinenia a zmení sa bool premenná, ktorou protihráč môže zabrániť prebratiu výhry pre zdržiavanie a na záver funkcia emitne udalosť, ktorú by mal protihráč zachytiť a tým vie že je obvinený.

**handle_timeout()** – ak bol hráč obvinený tak môže zabrániť prehre za zdržiavanie ak stihne odpovedať do 60 sekúnd tak že sa zmení bool premenná, ktorá zabráni protihráčovi vyhrať obvinením zo zdržiavanie.

**claim_timeout_winnings()** – ak ubehlo aspoň 60 sekúnd a protihráč nezareagoval na obvinenie tak si hráč preberie výhru a ukončí hru.

**is_game_over()** – ak je hra ukončená (DONE) vráti true, inak false.

**BattleshipPlayer.js**

Pridané do kontštruktora:

```
this.my_ships = [];
this.hits = 0;
this.last_opening_nonce = null;
this.last_proof = null;
this.last_guess_leaf_index = null;
this.opponent_lied = false;
this.contract_resp = null;

Battleship.events.TimeAccusion({}).on('data', (event) =>{
  console.log("accused ",event.returnValues.accused);
});
```

**place_bet()** – prepočíta sa stávka v eth na wei a pošle stávku do kontraktu (funkcia store_bid()).

**initialize_board()** – pošle svoj merkle root do kontraktu (funkcia store_board_commitment()).

**receive_response_ti_quess()** – vždy si hráč overí pomocou kontraktu (funkcia check_one_ship()) či oponent mu povedal pravdu a podľa toho si pridá zásah.

**accuse_timeout()** – obviní protihráča za zdržiavanie cez kontrakt (funkcia claim_opponent_left());

**handle_timeout_accusation()** – na odpovedanie na obvinenie cez kontrakt (handle_timeout_accusation() a na zistenie konca hry is_game_over()) a vráti bool hodnotu či je hra ukončená.

**claim_timeout_winnings()** – zavolá funkcie kontraktu (claim_timeout_winnings() a is_game_over()) na získanie výhry zdržiavaním súpera a vráti bool hodnotu či je hra ukončená.

**accuse_cheating()** – zavolá funkcie kontraktu (accuse_cheating() a is_game_over()) a vráti hodnotu či je hra ukončená.

**claim_win()** – skontroluje či hráč spravil 10 zásahov a má 10 lodí na svojej mape. Následne hráč oznámi kontraktu že má 10 lodí zavolaním kontrakt funkcie (check_one_ship()) pre každú loď a zavolá sa kontrakt funkcia na získanie výhry (claim_win()) a na záver ešte is_game_over() pre vrátenie bool hodnoty či je hra ukončená.

**forfeit()** – ak sa chce hráč vzdať, zavolá sa kontrakt funkcia forfeit().

## Implementačné prostredie:

Vývojové prostredie (IDE): Visual Studio Code
Jazyk: Solidity, javascript

## Testovanie

Testovanie prebehlo pomocou solidity-coverage https://github.com/sc-forks/solidity-coverage
V testoch som sa zameral na otestovanie funkcií, ktoré boli za úlohu do implementovať. Skúšal som scenáre pre pristupovanie k funkciám v rôznych stavoch hry, pristupovanie k funkciám ak ten kto volá nie je ani jeden z hráčov a aj scenár keď je jeden z hráčov čo nastane ak sa funkcia vykoná.

Napríklad:

```
contract('Battleship', () => {

    it('Should deploy smart contract properly', async () => {
        const Battleship = await ContractStarter.deployed();
        assert(Battleship.address !== '');
    });


    it('Should return false for is_game_over()', async () => {
        const Battleship = await ContractStarter.deployed();
        var game = await Battleship.is_game_over.call();
        assert.equal(game, false);
    });
```

Na začiatku skontrolujem či je kontrakt deployed a následne otestujem či hra beží.

Test coverage:

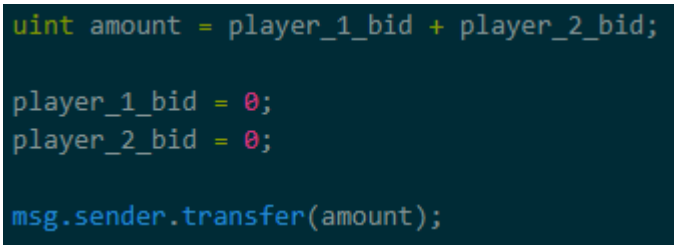
| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines |
|---|---|---|---|---|---|
| contracts\ | 63.25 | 60.87 | 100 | 61.8 | |
| contract_starter.sol | 63.25 | 60.87 | 100 | 61.8 | ... 511,513,514 |
| All files | 63.25 | 60.87 | 100 | 61.8 | |

( 63,25 + 60,87 + 100 + 61,8 ) / 4 = 71,48 %

## Bezpečnostná analýza

**store_bid()** - zraniteľnosti pri opätovnom vstupe.
**claim_win()** - zraniteľnosti pri opätovnom vstupe.
**forfeit()** - zraniteľnosti pri opätovnom vstupe.
**accuse_cheating()** - zraniteľnosti pri opätovnom vstupe.
**claim_timeout_winnings()** - zraniteľnosti pri opätovnom vstupe.

Tieto zraniteľnosti sú ošetrené nasledovne:




„now“ - neznamená aktuálny čas. "now" je alias pre "block.timestamp". "block.timestamp" môžu ťažiari do určitej miery ovplyvniť, treba byť opatrný.

# Záver

EMV (Ethereum Virtual Machine) má sloty 256bitové, do ktorých sa ukladajú premenné a teda záleží na poradí premenných ak chceme ušetriť za gas musíme sa znažiť ich uložiť čo najefektívnejšie aby sme využili čo najmenej slotov. Ďalej som sa naučil že premenné, ktoré zaberajú menej bitov napr. uint8 má oproti uint256 väčšie gas fee a na to aby sa nám vyplatilo mať uint8 musíme ich mať niekoľko v jednom slote teda, dôjdeme do bodu kedy počet uint8 sa viac vyplatí, lebo keby máme jeden uint8 tak sa nám viac oplatí uint256, kvôli relácií, ktorá je spojená s narábaním s premennou, ktorá má iný počet bitov ako 256. Ďalej som sa naučil robiť testy v .js, komunikovať s kontraktom, nasadiť ho vytvárať a zachytávať eventy.