DMBLOCK zadanie 2

Fakulta informatiky a informačných technológií Slovenskej technickej univerzity

Predmet: Digitálne meny a Blockchain

Zadanie 2 - Smart kontraktový systém pre námorné bitky

Autor: Roman Bitarovský

Cvičiaci: Viktor Valaštín

11. 04. 2022

Disclaimer

Uvedené ukážky kódu v tejto dokumentácii nemusia byť 100% zhodné a syntakticky správne s reálnym kódom a boli zjednodušené pre lepšiu čitateľnosť alebo z dôvodu formátovania.

Funkčné kódy pre zadanie je možné nájsť v pribalených súboroch publikovaných spolu s touto dokumentáciou.

Obsah

Opis implementácie

Smart contract

Smartcontract je napísaný v jazyku solidity spúšťaný na lokálnej kópie siete ethereum pomocou nástroju Ganache a Ganache-cli. Pri jeho vývoji boli využívané nástroje: remix, truffle a VS code.

Potrebné importy: ECDSA.sol

Kompilátor: `pragma solidity >=0.4.22 <0.7.0;`

Základné atribúty contractu

Pre implementáciu hry a reprezentovanie potrebných náležitosti je hráč reprezentovaný pomocou “objektu” čo v solidity predstavuje použitie kľúčového slova struct.

```
struct Player {
    address addr;
    bytes32 merkle_root;
    uint32 num_ships;
    uint256[] leaf_check;
}
```

Ďalšie stavy sú prezentované pomocou nasledovných premenných:

```
// 0 -> game not starte
// 1 -> game started and is in progress
// 2 -> game is over
uint8 state = 0;
uint256 public bid;
uint public timeout_stamp;
uint constant private _TIME_LIMIT = 1 minutes;
address public winner;
```

Funkcia store_bid()

Uloží bid pre každého hráča. Zároveň zaeviduje dve unikátne adresy kt. predstavujú dvoch hráčov zúčastňujúcich sa danej hry. Funkcia ošetruje aby žiadny bid nebol menší ako nula nakoľko nechceme aby hráči hrali zadarmo. Ďalšie ošetrenia sú aby každý hráč mohol zaslať bid iba raz a aby mohli byť stávky vložené iba presne dvomi hráčmi.

V prípade ak sa druhý hráč pokúsi vložiť väčší bid ako prvý tento prevyšujúci rozdiel mu bude vrátený:

```
else if (player2.addr == address(0)) {
  // ak druhy hrac dal viac ako prvy rozdiel sa zuctuje spat
  if (msg.value > bid){
      msg.sender.transfer(msg.value - bid);
  }
}
```

Funkcia clear_state()

Tu je vhodné zdôrazniť, že funkcia je typu internal čo je logické vzhľadom nato, že resetuje atribúty contractu na znova nulové hodnoty.

Funkcia znuluje atribúty pre oboch hráčov a ďalšie stavy ktoré si uchováva hra.

Funkcia store_board_commitment()

Podľa adresy volajúceho danému hráčovi priradí merkle_root, kt. je parametrom funkcie. Samozrejme ošetruje aby s funkciou mohli interagovať iba hráči zúčastňujúci sa hry .

Ak obaja hráči majú nastavený ich počiatočný merkle root tak je zmenený stav hry reprezentovaný atribútom `state` tak aby bolo možné hru hrať.

Funkcia check_one_ship()

Je najzložitejšia časť celej implementácie. Úlohou je kontrolovať koľko lodí hráči položili a aký počet lodí zasiahli súperovi.

```
if (owner == player1.addr) {
    if (owner == msg.sender){
        player1.num_ships += 1;
    }
    else {
        player2.leaf_check.push(guess_leaf_index);
    }
}
else {
    if(owner == msg.sender){
        player2.num_ships += 1;
    }
    else {
        player1.leaf_check.push(guess_leaf_index);
    }
}
```

Samozrejme potrebné je aj kontrolovať či loď už nebola trafená nakoľko nie je možné jednu loď zarátať dvakrát.

Funkcia claim_win()

V prípade ak hráč volajúci túto funkciu spĺňa požiadavky pre vyplatenie výhry (stávky, kt. obaja hráči vložili) mu je výhra prostredníctvom tejto funkcie bezodkladne vyplatená.

Funkcia zároveň emituje event o ukončení hry, aby sa vykonali potrebné zmeny na frontende.

Funkcia forfeit()

Slúži ak sa chce jeden z hráčov vzdať, čo následne spôsobí vyplatenie výhry druhému hráčovi. Obsahuje ošetrenia aby funkciu mohli volať iba hráči zúčastňujúci sa aktuálnej hry a ďalšie aby napríklad nebolo možné vyplatiť výhru na neplatnú

adresu či to, že nie je možné zavolať funkciu sám na seba resp. vzdať sa a zároveň dostať výhru.

Funkcia accuse_cheating()

Obvinenie protivníka z podvádzania. Pomocou merkle tree skontroluje, či oponujúci hráč nepodvádza a ak áno tak hráčovi, kt. nahlasoval podvádzanie je vyplatená výhra.

Je ošetrené aby obviňovať z podvádzania mohli iba hráči zúčastňujúci sa hry a aby nebolo možné obviniť z podvádzania samého seba.

Funkcia claim_opponent_left()

Umožňuje obviniť oponenta zo zdržiavania. Vytvorí časovú pečiatku kedy bol oponent obvinený. A emituje event o tejto udalosti aby bol frontend schopný notifikovať oponenta, kt. má určitý čas na zareagovanie a zvrátenie tohto obvinenia.

Funkcia handle_timeout()

Slúži obvinenému zo zdržiavania pre zvrátenie tejto udalosti. Kontroluje či hráč volaný túto funkciu je obvinený a ak zareaguje tak anuluje časovú pečiatku.

Funkcia claim_timeout_winnings()

Zabezpečuje aby iba hráč, kt. druhého obvinil zo zdržiavania mohol dostať výhru v prípade, že od obvinenia uplynul stanovený čas.

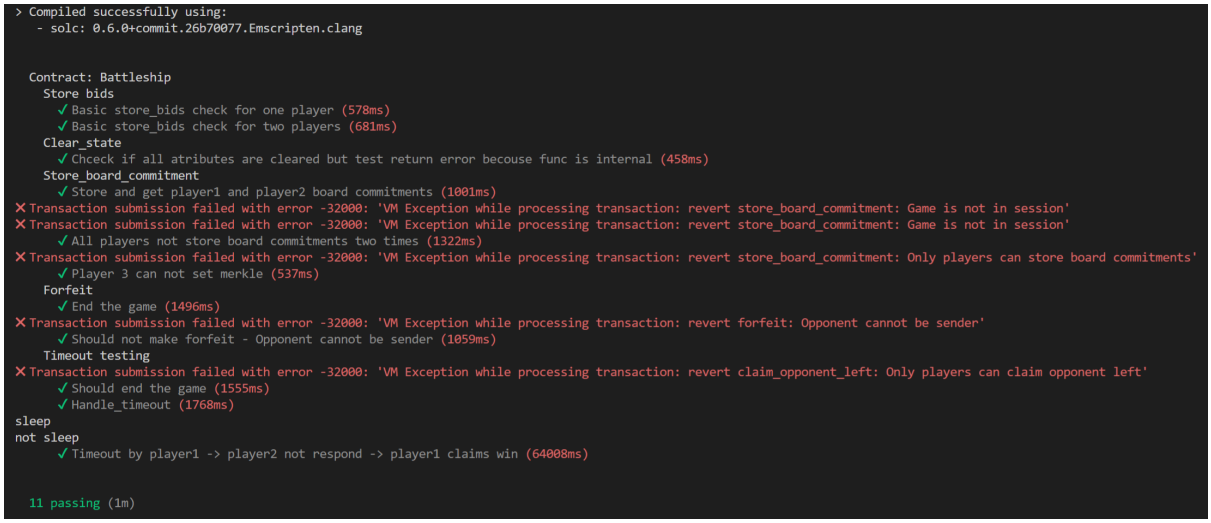
BattleshipPlayer.js

Túto časť projektu som neimplementoval ale v plnom rozsahu prebral z poskytnutého materiálu od Kika. Preto tu tento kód nebudem rozoberať.

Testovanie

Testovanie by som rozdelil na dve časti a to ručné testovanie predstavujúce hranie hry (plne funkčné) a následne pozorovanie výsledkov. A automatizované testovanie prostredníctvom vlastným JS testov spúšťaných pomocou `truffle test`.

JS testy



1. test - Store_bids

V rámci prvého testu bolo preverené, či funkcia store_bids funguje správne, Teda bola využitá na uloženie bidov, kt. boli následne pomocou get funkcie porovnané so vkladanou hodnotou.

2. test - Clear_state

V druhom teste bola vízia otestovať funkčnosť clear_state. táto funkcie je však internal a nevieme ju týmto spôsobom otestovať. Čo je logické nakoľko nie je prípustné aby bolo z vonku možné anulovať stavy conctractu. Preto sa test zameriava na to či volanie tejto funkcie vráti error.

3. test - Store_board_commitment

Test funkcie store_board_commitment sa zameriava najskôr na uloženie pre oboch hráčov a následne porovnanie uloženej hodnoty prostredníctvom getu.

Potom je pokus aby hráči uložili svoje hodnoty dvakrát po sebe čo však nie je možné.

Posledný scenár je kedy sa snaží stav uložiť hráč, kt. sa nezúčastňuje hry.

4. test - Forfeit

Je zameraný či po vzdaní sa budú znulované herné atribúty rovnako ako aj to či je prostredníctvom funkcie možné takto ukončiť hru za oponenta.

5. test - Timeout

V celku testuje funkčnosť troch funkcií zameraných na timeoutovanie.

Prvý scenár predstavuje obvinenie protivníka zo zdržiavania a jeho následná pravdivosť, teda protivník neodpovie.

Druhý scenár testuje funkčnosť kedy súper poprie obvinenie zo zdržiavania.

Test coverage

Vyhodnotenie pomocou nástroja solidity-coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	47.06	35	59.26	45.86	
ECDSA.sol	0	0	0	0	... 63,64,66,80
contract_starter.sol	51.43	37.5	64	50.35	... 364,374,377
All files	47.06	35	59.26	45.86	

Security analýza

Čo sa týka automatizovanej security analýzy použil nástroj Mythril.

Táto analýza však neukázala žiadne chyby ani odporúčania pre zlepšenie.

```
C:\Users\bitar\PycharmProjects\DMBLOCK-zadanie-2\truff\contracts>docker run -v %cd%:/tmp mythril/myth analyze /tmp/contract_starter.sol:Battleship --execution-timeout 600 -o text
The analysis was completed successfully. No issues were detected.

C:\Users\bitar\PycharmProjects\DMBLOCK-zadanie-2\truff\contracts>
```

Zodpovedanie otázok

1. otázka

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?

Podľa zadania by nemal nakoľko je písané, pri kontrole nároku na výhru sa kontroluje či má 10 zásahov súperových lodí ale aj to či položil 10 lodí.

Avšak ak sa súper vzdá prípadne sa mu poradí súpera usvedčiť zo zdržiavania tak môže vyhrať a dostať výhru.

2. otázka

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

Ak hráč položí na dosku viac lodí tak tým pomáha súperovi, čo samozrejme robiť nechceme nakoľko sa hrá o cennú výhru. Teda týmto krokom nepomáha sebe ale strane druhej.

3. otázka

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

Obaja hráči by si na začiatku hry museli uložiť dôkaz, že majú 10 lodí a musela by sa pridať funkcionalita aby mohol oponent tento dôkaz vyžiadať a nejakým spôsobom ho validovať aby bolo možné overiť, že nie je sfalšovaný.

4. otázka

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

Ak napríklad hráč opustí hru nekorektne (zatvorenie prehliadača) daná odmena ostane zaseknutá v kontrakte a druhá strana tak stratí výhru aj keď na ňu má nárok (ak ho nenapadne nahlásiť protivníka za zdržiavanie), čo však ak aj druhý hráč zavrie prehliadač napríklad nechtiac. Toto sa však deje iba v našom prostredí a dá sa to ošetriť.

5. otázka - spätná väzba

Ako by sa dal v budúcnosti vylepšiť tento projekt?

Zrušil by som tú časť kedy majú študenti implementovať BattleshipPlayer.js. Dôvod je ten že mi to nepríde ako cieľom tohto zadania a vlastne ani predmetu. Iba to všetko skomplikovalo nakoľko bolo ťažké povedať, či mi aplikácia nefunguje kvôli chybnému JS alebo kvôli chybe v solidity.

Zhodnotenie

Zadanie bolo pomerne zaujímavé a okrem cenných skúseností so solidity si odnášam aj skúseností s remixom, truffle a ďalšími tools, kt. rozbehanie dalo niekedy zabrať a vyžadovalo si podrobné skúmanie príslušnej dokumentácie.

Určite by sa s tým dalo ešte vyhrať avšak čas je neúprosný čo je celkom škoda.