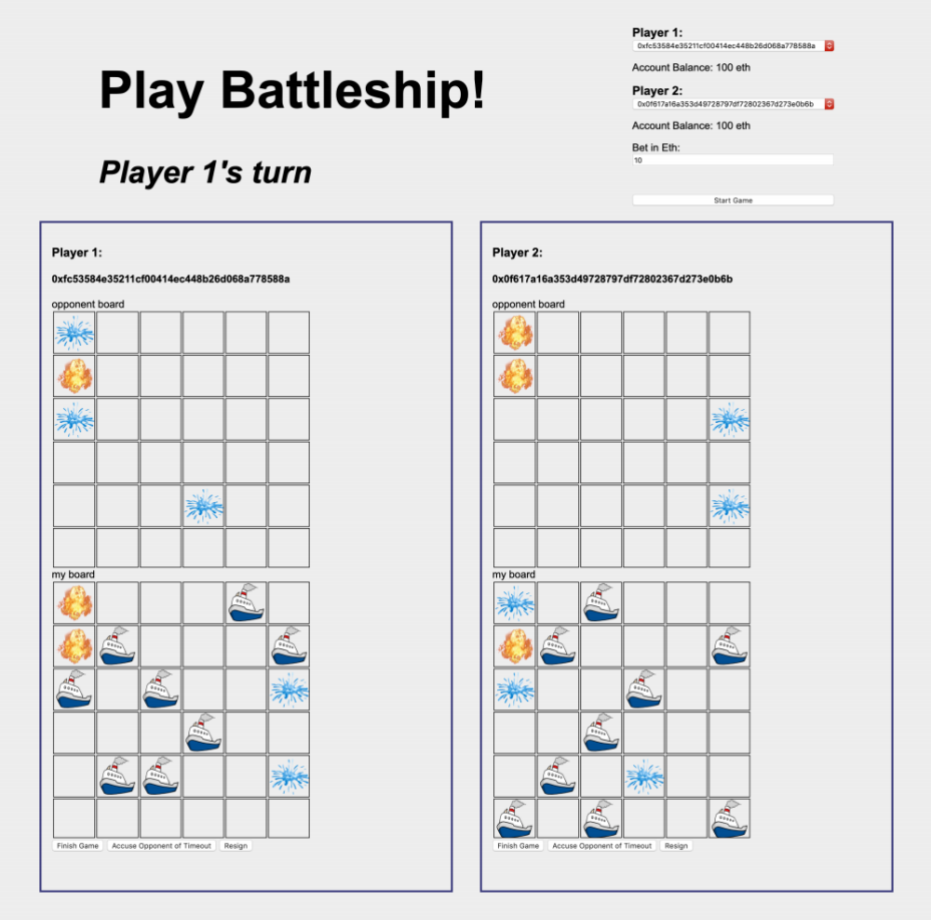
Zadanie 2 – Smart kontraktový systém pre námorné bitky

DISCLAIMER: Toto zadanie vychádza do značnej miery z open-source kurzu o blockchain technológiách fungujúcom na Univerzite v Stanforde.

Zadanie 2 je povinné! V rámci tohto projektu budete pracovať na webovom klientovi a Solidity kontrakte, ktorý umožní dvom hráčom hrať hru v námornej bitke pomocou state channel. Hráč interaguje so Solidity kontraktom raz v čase inicializácie, aby mohol začať hru, raz (ak vyhral) na konci hry s požiadavkou o výhru, a podľa potreby keď nahlasuje súperovo podvádzanie alebo obviňuje súpera, že jeho ďalší krok trvá príliš dlho.

V rámci tohto zadania sa očakáva, že zároveň budete hľadať možné spôsoby, ako by mohol používateľ so zlým úmyslom tento systém podviesť a napíšete kód, ktorý sa proti týmto útokom bráni. Nemusíte sa brániť pred všetkými možnými útokmi, ale mali by ste brať do úvahy aj nejaké útoky, ktoré nie sú explicitne požadované zadaním. **Toto budete preukazovať aj testovaním vášho riešenia.** Odpoviete tiež na niektoré otázky o bezpečnostných aspektoch, ktoré zohľadňujú útoky, proti ktorým sme sa v kóde nechránili. Snáď vám dostatočne poradíme, aby ste premýšľali o napísaní dobrého "obranného" kódu. Poďme na to!



1. Prehľad hry námorné bitky

Námorné bitky (stolová hra) je hádacia hra, ktorú hrajú dvaja hráči a ktorí si do konca hry navzájom nevidia hracie dosky. Obaja hráči zvyčajne umiestnia 5 lodí rôznych dĺžok na svoju dosku veľkosti 10x10, ktorá simuluje oceán. Potom sa striedajú pri hádaní o umiestnení lodí ich súperov. Hráč triafa miesta, keď povie súperovi súradnice [i, j], ktoré chce skontrolovať. Ak sa niektorá z lodí súpera nachádza na tomto mieste, súper odpovie, že bol zasiahnutý. V opačnom prípade odpovie, že útočník minul.

V prostredí stolových hier môžu hráči podvádzať vyhlásením, že umiestnili 5 lodí, keď vlastne neuložili žiadnu. Môžu tiež podvádzať tým, že budú klamať o zásahoch a minutiach, keď súper urobí útok.

V tejto implementácii Námorných bitiek je **hracia plocha 6x6** (celkom 36 políčok) a každý hráč musí umiestniť **10 lodí**. Každá loď zaberá práve jeden štvorec. Aby sa zabránilo podvádzaniu ohľadom počtu lodí, hráči interagujú so Solidity smart kontraktom, kde sa zaväzujú k svojmu pôvodnému stavu hracej plochy. Počas hry interagujú off-chain bez komunikácie so smart kontraktom. Môžu lokálne overovať zásahy a minutia. Ak má niektorý hráč podozrenie, že súper klamal, či už trafil, alebo netrafil, ALEBO ak má podozrenie, že súper opustil hru (príliš dlho mu trvá odpovedať), môže na uzavretie a vyriešenie týchto obvinení použiť Solidity kontrakt.

2. Ako začať

- Prepokladáme, že máte nastavené prostredie `NodeJS, Truffle, Ganache`.
- Ak by ste mali problémy s Ganache, skúste dať príkaz `npm install -g ganache-cli` a nainštalujte Ganache CLI, ktoré sa dá použiť taktiež na simuláciu skutočného uzla Ethereum na localhoste a má trochu väčšiu funkcionalitu ako čistý Ganache. Uzol spustíte pomocou `ganache-cli` do konzoly. Môžete ho kedykoľvek zastaviť pomocou Ctrl-C.
- Stiahli ste si štartovací kód z AISu a máte ho otvorený vo svojom obľúbenom IDE alebo textovom editore (Visual Studio Code ...). Zoznámte sa s kódmi. Očakáva sa, že v prípade tohto zadania vyplníte súbory `contract_starter.sol` a `BattleshipPlayer.js`. V prípade potreby môžete do týchto dvoch súborov pridávať pomocné funkcie. Nejaké pomocné funkcie, ktoré používajú používateľské rozhranie, sú aj v `battleship-game.js` a `utils.js`. Môžete si ich prečítať pre oboznámenie sa.

- Inicializujte si Truffle pomocou `truffle init` a môžete váš kontrakt spúšťať, príp. testovať. Možnosť ale je aj skúšanie kontraktu vo webovom prehliadači: otvorte stránku https://remix.ethereum.org. Na karte „Deploy&Run“ nastavte prostredie na „Injected Web3,“ ak idete cez Metamask pripojený na Ganache, alebo „Web3 Provider“ ak idete priamo z prehliadača na Ganache. Po výzve kliknite na „OK“ a potom nastavte „Web3 Provider Endpoint“ na http://localhost:7545, resp. 8545, ak máte Ganache CLI. Na karte „Solidity Compiler“ nastavte kompilátor na verziu do 0.7.0, napr. 0.6.6+[posledný dostupný commit].
- Otvorte vo svojom prehliadači `index.html` (najlepšie funguje v prehliadači Chrome), aby ste vstúpili do klientskeho rozhrania, kde si môžete vyskúšať svoju implementáciu hry. Na tejto stránke si môžete vybrať 2 hráčov podľa adresy a určiť, koľko každý z hráčov staví na hranie tejto hry, uvedené v ETH (zostatok na účte pre konkrétneho používateľa musí byť väčší alebo rovnaký). K dispozícii sú dva pohľady na hru: pohľad naľavo pozostáva z informácií, ktoré dostane hráč 1. Hráč 1 nevie umiestnenia lodí druhého hráča a naopak to platí aj pre hráča 2, ale obaja hráči môžu vidieť obe dosky vedľa seba. Je to kvôli testovaniu a malo by vám to uľahčiť overovanie vašej implementácie.
- Správy a chybové hlásenia sú veľmi užitočné a môžete si ich pozrieť v JavaScript konzole v prehliadači. Ak zatiaľ všetko funguje, nemali by ste v konzole vidieť žiadne chyby (varovania môžete bezpečne ignorovať).
- Implementujte kód pre požiadavky uvedené nižšie. Keď máte svoj kontrakt, skompilujte ho a nasaďte a potom **aktualizujte hash kontraktu (adresu) a ABI** v `BattleshipPlayer.js`. ABI je možné skopírovať na karte „Solidity Compiler“ v Remixe, resp. cez Truffle v bin adresári. Hash kontraktu sa dá skopírovať z karty „Deploy and Run Transactions“ alebo cez truffle konzolu alebo cez Ganache. Pozor, že hash kontraktu nie je hash transakcie, ktorá vytvorila kontrakt. Ak zistíte, že niektorá funkcionalita funguje iba raz a potom sa vyskytnú chyby pri ďalších pokusoch, znovu deploynite váš kontrakt a skopírujte nový hash. Môže sa stať, že solidity neaktualizoval stav, resp. ostal zaseknutý v nejakej fáze.

3. Životný cyklus vašej aplikácie

Skôr ako začnete, pozorne si prečítajte všetky časti. Niektoré časti vyžadujú implementáciu viacerých funkcií, ale celkové množstvo kódu nie je veľké. Funkcie `clear state` a `is game over` spomínané v 3.5 môžu byť užitočné pre všetky časti.

3.1. Začiatok hry

Na začiatku hry hráči uzatvárajú stávky interakciou s kontraktom. Obaja hráči ponúknu rovnakú sumu (podľa používateľského rozhrania). Po vložení stávky umiestnia svoje bojové lode a vytvoria podpis k počiatočnému stavu na hracej ploche. Aby ste toto mohli implementovať, budete musieť upraviť nasledovné v `BattleshipPlayer.js`:

- **constructor**: uložte akýkoľvek stav, ktorý by ste chceli lokálne zachovať, a inicializujte obslužné event handlery (môžete to urobiť v časti 3.4).
- **place_bet**: interakciou s kontraktom uzatvárajte stávky.
- **initialize_board**: väčšina z tohto je už urobená za vás. Pridajte interakciu s kontraktom na uloženie počiatočného stavu. Čo znamená stav dosky nájdete v časti 3.2.

V `contract-starter.sol` budete musieť upraviť nasledovné:

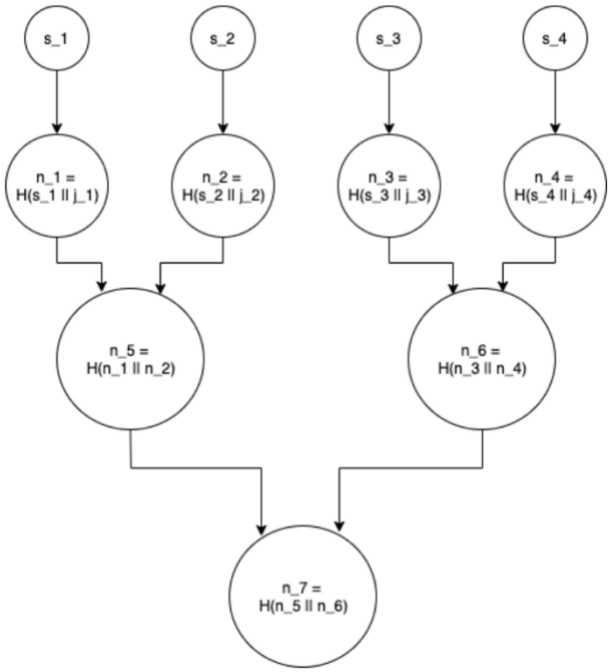
• **store_bid**: Prvý hráč, ktorý komunikuje s kontraktom, nastaví požadovanú ponuku na hranie hry. Druhý hráč musí ponúknuť túto istú sumu, inak sa hra nezačne. Vráťte prebytočné prostriedky naspäť druhému hráčovi, ak ponúkne viacej.

• **store_board_commitment**: ukladajte všetky potrebné stavy, aby ste neskôr overili, či hráči neklamali o svojich doskách.

3.2. Hranie hry off-chain

Pre túto časť nemusíte písať žiadny kód. Po inicializácii hry, hráči triafajú umiestnenie súperových lodí kliknutím na verziu súperovej dosky, ktorú vidia vo svojom ťahu. Zásahy sú zobrazené ako obrázky výbuchu a minutia sú zobrazené ako "šplechnutia". Celá táto interakcia je off-chain.

Keďže stavy dosky sú v skutočnej hre utajené, je použitá schéma commitov pomocou Merkleho stromu na hlásenie stavu hracej dosky do kontraktu. Výsledkom je, že žiadny iný subjekt nemôže zistiť, kde sú vaše lode. Tento systém využíva nasledujúcu schému:



Predpokladajme, že hráč A hrá s doskou `N x N`, celkovo s N^2 bunkami, a umiestnil na ňu nejaký počet lodí - `k`. Pre každú bunku dosky vytvoríme uzol v liste n_i pre $i \in [1, N^2]$ pomocou $n_1 = H(s_1||j_1)$, kde s_1 je stav prvej bunky a j_1 je náhodne vybraný nonce. Stav bunky je buď 1 (označujúci prítomnosť lode) alebo 0 (označujúci prázdnu bunku). Na základe týchto listových uzlov zostrojíme Merkleho strom. Napríklad pre $N=2$ (celkom 4 bunky na doske) by bol list: $n_1 = H(s_1||j_1)$ a uzol na úrovni 1 by bol: $n_5 = H(n_1||n_2)$. Diagram vyššie ukazuje, ako by vyzeral Merkleho strom pre tento príklad.

Na preukázanie toho, že sa loď nachádza na konkrétnom mieste v Merkleho strome, musí vlastník lode predložiť dôkaz o inklúzii. Tento dôkaz pozostáva z hashov uzlov pozdĺž cesty od koreňa k listu, spolu s hashmi súrodeneckých uzlov na každej úrovni.

Zvážte, čo sa stane, ak vynecháme hodnoty nonce z výpočtov hash pre listové uzly. Vzhľadom na koreň Merkleho stromu by bolo triviálne získať stav dosky „hrubou silou," pretože existuje iba N^2 štvorcov (pre malú hodnotu N) a štvorce majú binárne stavy.

3.3. Obviň protivníka z podvádzania

Ktorýkoľvek hráč môže kedykoľvek druhého hráča obviniť z podvádzania. Podvádzanie definujeme ako klamanie o polohe lode poskytnutím chybného dôkazu. Implementácia Merkleho stromu na strane klienta kontroluje, či je každý dôkaz v odpovedi na hádanie hráča overený. Vašou úlohou v tejto časti je implementovať rovnakú kontrolu pomocou kontraktu. Aby ste to implementovali, budete musieť v `BattleshipPlayer.js` upraviť nasledovné:

- **accuse_cheating**: Odošlite kontraktu to, čo potrebuje na overenie, či hráč podvádza.

Tiež budete musieť upraviť časti aj v `contract_starter.sol`:

- **accuse_cheating**: Použite to, čo klient pošle na overenie, či súper nepodvádza. Ak súper podvádza, pošlite celkovú výhru žalobcovi.

3.4. Obviň protivníka z odchodu alebo zdržiavania

Hráč môže kedykoľvek druhého hráča obviniť, že odišiel alebo zdržiava - ak napríklad súperovi trvá príliš dlho, kým urobí ťah, môže ho obviniť. Toto obvinenie by malo spustiť event v Solidity, ktorý musí súper vyriešiť v časovom limite **1 minúty**. Ak vyprší čas, môže žalobca uskutočniť druhé volanie na smart kontrakt s cieľom požadovať celkovú výhru.

Aby ste to implementovali, budete musieť v `BattleshipPlayer.js` upraviť nasledovné:

- **accuse_timeout**: Oznámte kontraktu, že chcete súpera obviniť z opustenia hry.
- **handle_timeout_accusation**: Ak vás obvinili z odchodu, interagujte s kontraktom a naznačte, že stále hráte. Toto zabráni rozdeleniu výhier, keď dôjde čas.
- **claim_timeout_winnings**: Ak ste súpera obvinili a ten včas neodpovedal, interagujte s kontraktom s nárokovaním výhry.

V `contract_starter.sol` budete musieť upraviť aj nasledovné:

- **claim_opponent_left**: Použite eventy v Solidity na spustenie časovača. Pozrite [tu] pre úvod (tiež odkazovaný v časti Referencie). Najnovšia verzia web3 má od tejto syntaxe menšie rozdiely.
- **handle_timeout**: Ak ste boli obvinení, urobte všetko preto, aby časovač neskončil.
- **claim_timeout_winnings**: Ak súperovi vypršal časovač, získajte výhru.

3.5. Vzdanie sa

Ktorýkoľvek hráč sa môže kedykoľvek vzdať. Po uzavretí svojich ponúk v kontrakte na začiatku hry už neskôr hráč, ktorý sa vzdáva, okamžite stratí svoje peniaze a súper vyhráva. Toto sa deje bez ohľadu na podvádzanie, klamanie o počte lodí atď. Aby ste to mohli implementovať, budete musieť v `BattleshipPlayer.js` upraviť:

- **forfeit_game:** Povedzte kontraktu, že sa vzdávate.

V `contract_starter.sol` budete musieť upraviť:

- **forfeit:** Odošlite všetky výhry súperovi a ukončite hru.

3.6. Nárok, že niekto vyhral

Až na konci hry hráči „vyložia karty“ potrebné na preukázanie toho, že hru vyhrali - umiestnenia 10 lodí na svojej vlastnej doske A 10 výstrelov (s dôkazmi), ktoré viedli k zásahom na súperove lode. Ak výherca nedokáže predložiť tieto dôkazy, nemôže vyhrať hru. Budete potrebovať doimplementovať funkcionalitu v `BattleshipPlayer.js`:

- **claim_win**

V `contract_starter.sol` budete musieť upraviť:

- **check_one_ship:** Skontroluje, či je jedna loď tam, kde má byť, po predložení Merkleho dôkazu. Použite ju na indikáciu umiestnenia svojich lodí a výherných zásahov.
- **claim_win:** Ak ste označili potrebné lode, táto funkcia by vám mala poslať všetky výhry.
- **clear_state:** Vymazať stav a nastaviť hru tak, aby skončila.
- **is_game_over:** Getter funkcia, aby ste mohli skontrolovať vo web3, či je koniec hry.

4. Písané otázky

Vo finálnej dokumentácii prosím odpovedzte na tieto otázky. Obmedzte svoje odpovede na **maximálne 3 vety** ku každej otázke:

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?
2. Prečo neobmedzujeme hráčov v umiestňovaní viac ako 10 lodí na ich dosku?
3. Nemáme mechanizmus, ktorý by hráča obviňoval z umiestnenia menej ako 10 lodí na hraciu plochu. Ako by ste ho vedeli implementovať?
4. Napadajú vám scenáre útoku alebo konkrétne zraniteľné miesta v niektorom z uvedených kódov, proti ktorým by ste sa nedokázali ubrániť?

Voliteľná spätná väzba: Ako by sa dal v budúcnosti vylepšiť tento projekt?

5. Referencie

- Wikipedia battleship game: https://en.wikipedia.org/wiki/Battleship (game)
- State Channels: https://arxiv.org/pdf/1702.05812.pdf
- Solidity Events - https://programtheblockchain.com/posts/2018/01/24/logging-and-watching- solidity-events/

Odovzdanie

Dokumentáciu a zdrojové kódy implementácie študent odovzdáva v elektronickom tvare do AISu v určenom termíne vo formáte *.zip alebo *.tar. **Odovzdanie aspoň nejakej časti zadania je povinné!** Aj na toto zadanie je možné využiť Late days. Všetky kódy prejdú kontrolou originality, v prípade vysokej percentuálnej zhody bude študentovi začaté disciplinárne konanie.

Súčasťou odovzdania musia byť aj vami napísané testy, ktoré pokryjú **aspoň 10% zdrojového kódu**. Nástroje na Solidity test coverage nájdete tu. Podmienkou je aj aktívne odprezentovanie programu študentom na cvičení podľa harmonogramu.

Súčasťou riešenia je aj dokumentácia, ktorá musí obsahovať najmä:

a) formálne náležitosti ako sú titulná strana, číslovanie strán;
b) odpovede na otázky z časti 4;
c) popísané doimplementované časti kódu;
d) voľbu implementačného prostredia;
e) opis prostredia na testovanie a opis vybraných testov;
f) sumár test coverage a použité nástroje;
g) výsledky security analýzy (stačí aj statická) – nástroje tu;
h) iné záležitosti, ktorými sa študent chce pochváliť;
i) záver a v ňom zhodnotenie vlastného príspevku a čo som sa naučil/a.

Hodnotenie (bez minima)

Celé riešenie - max. 15 bodov, z toho:

- max. 4 body za riešenie úlohy v contract-starter.sol – bude sa prihliadať aj na celkové gas, pamäťové nároky, optimálna práca s blockchainom;
- max. 3 body za riešenie úlohy v BattleshipPlayer.js – ako úspešne sa vám darí interagovať so smart kontraktom;
- max. 2 body za vytvorené testy v rámci truffle suite;
- max. 2 body za vykonanie security analýzy – jej spracovanie, použité nástroje, identifikované hrozby a ich adresácia;
- max. 4 body za výslednú dokumentáciu – 1 bod formálna stránka, 1 bod odpovede na otázky, 2 body zvyšné časti.

Bonusy - dajú sa získať naviac k bodom za zadanie

- max. 4 body za prerobenie jednoobrazovkovej aplikácie na dve samostatne spustiteľné obrazovky na rôznych PC (môžete testovať aj na localhoste vo virtuálke a host systéme);
- max. 4 body za celkový test coverage až do výšky 70-100% (za každých ďalších 15% test coverage nad rámec povinných 10% pridelíme 1 bod, napr. celkový test coverage 25-39% = 1 bod; test coverage 40-54% = 2 body; test coverage 55-69% = 3 body; 70-100% = 4 body);
- max. 1 bod za pridanie nejakej zaujímavej funkcionality (po konzultácii s cvičiacim);
- max. 1 bod za využitie ďalších, do teraz nespomenutých, nástrojov zo zoznamu (po konzultácii s cvičiacim);
- max. 1 bod za vytvorenie herného tokenu, ktorým sa bude do hry registrovať a pomocou ktorého sa bude aj vyplácať odmena (v aplikácii taktiež potrebná kompletná zmena ETH na token).