DAAR

PROJET 3 : Touché Coulé Décentralisé

Maxime Dresler (21100187)
Soufiane Hifdi (21115089)

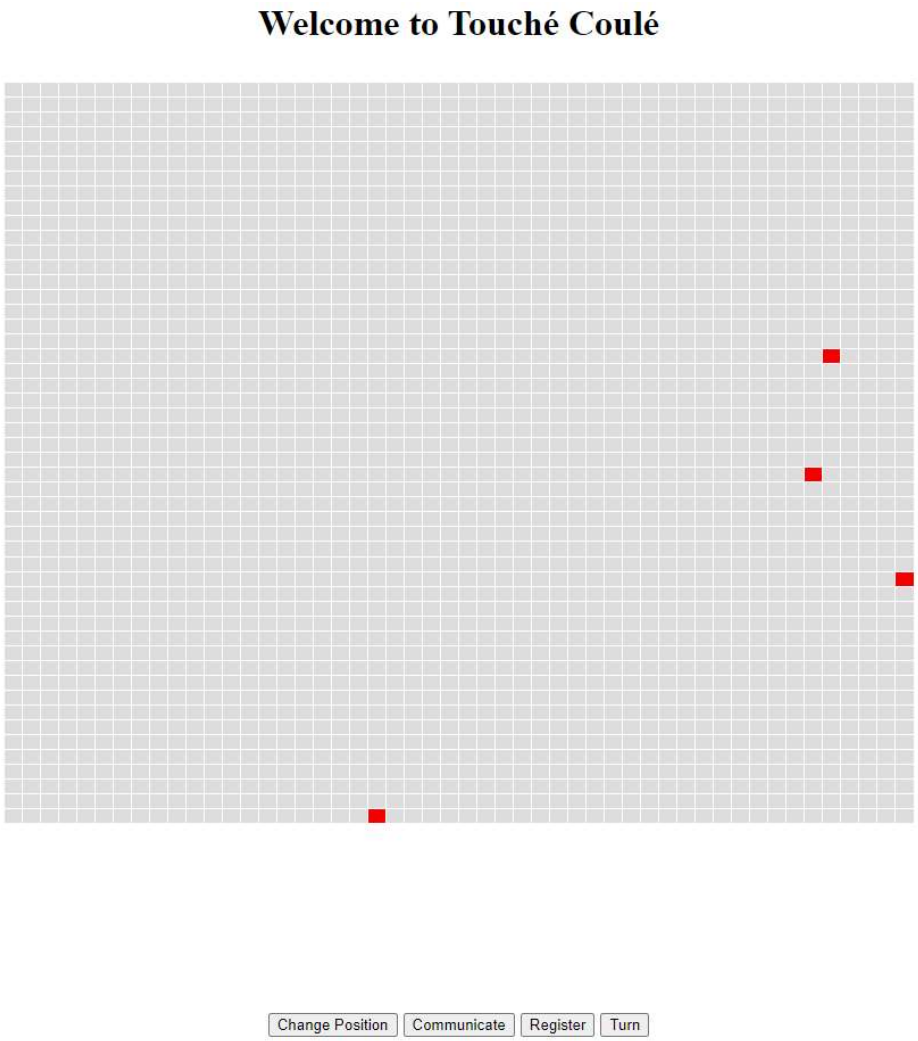
2022/2023

Implémentation du jeu

Avec Hardhat nous avons accès à plusieurs comptes sur ethereum, où chaque compte est associé à un joueur.

Chaque joueur joue selon les règles décrites dans main.sol.

Les joueurs ont accès à cette vue pour interagir avec la blockchain, chaque bouton (en bas) permettant d'utiliser les méthodes définies dans main.sol :



Un joueur fait partie du jeu lorsqu'il clique sur ***Register***. Dès lors un contrat ship pour ce joueur est déployé dans la blockchain.

Chaque joueur peut déployer au maximum 2 ships sur la grille de jeu, c'est-à-dire, qu'on peut déployer au plus 2 contrats ship par compte (Hardhat).

Lorsque l'on clique sur ***Turn***, tous les ships qui sont dans la grille de jeu, vont tirer de manière aléatoire. Lorsqu'un ship est touché, il est éliminé du jeu.

Tous les 5 tours, on peut changer de position des ships (que l'on possède) en cliquant sur le bouton : ***Change Position***.

Lorsque l'on clique sur ***Communicate***, on permet l'échange d'informations entre 2 ships du même joueur. Les 2 ships auront la même vision de la grille de jeu.

Modifications apportées

Pour expliquer notre travail , nous allons présenter les modifications faites dans les principaux fichiers du projet.

Dans Main.sol

Le premier fichier modifié est le main.sol. C'est le contrat déployé dans la blockchain qui décrit le comportement du jeu. Les joueurs font des appels distants aux méthodes de Main.sol pour changer l'état du jeu à partir de l'interface graphique.

Variables

Tout d'abord à la ligne 22, on ajoute une map ***turns*** qui retourne, pour un index de ship donné, le nombre restants de tours avant de pouvoir changer de position. A chaque turn, on décrémente ce nombre pour chaque ship (chaque index).

```
15  contract Main {
16    Game private game;
17    uint private index;
18    mapping(address => bool) private used;
19    mapping(uint => address) private ships;
20    mapping(uint => address) private owners;
21    mapping(address => uint) private count;
22+   mapping(uint => uint) private turns; //store the autorization to change positions
```

Par la suite on a ajouté un événement moved, permettant au frontend de mettre à jour la position des ships. Cet événement sera émis dans la fonction ***changePosition()***.

```
→ 32+   event Moved(
  33+     uint ship,
  34+     uint prev_x,
  35+     uint prev_y,
  36+     address indexed owner,
  37+     uint x,
  38+     uint y
  39+   );
```

Fonctionnalité Register - déploiement contrat ship

Quelques changements ont été ajoutés à la fonction ***register().***

Dans notre implémentation, dès que l'on appuie sur le bouton ***Register***, on invoque la méthode ***register()*** du main.sol, qui va exécuter les lignes suivantes :

```
48+  function register() external {
49+    address ship = address(new Ship(msg.sender, index)); // get address of the new ship
50+    require(count[msg.sender] < 2, "Only two ships");
51+    require(!used[ship], "Ship alread on the board");
52+    require(index <= game.height * game.width, "Too much ship on board");
53     count[msg.sender] += 1;
54+    turns[index] = 5; // a ship can change position every 5 turns
55     ships[index] = ship;
56     owners[index] = msg.sender;
57     (uint x, uint y) = placeShip(index);
58+    console.log("Register --> id:%s x:%s, y%s",index,x,y);
59     Ship(ships[index]).update(x, y);
60     emit Registered(index, msg.sender, x, y);
61     used[ship] = true;
62     index += 1;
63   }
```

A la ligne 49 le keyword ***new*** permet de déployer un nouveau contrat Ship (voir documentation) , qui représente l'état d'un ship d'un joueur donné. On récupère par la suite l'adresse du contrat Ship créée dans la variable ship.

La signature de la fonction a été modifiée, pour ne pas avoir à récupérer les contrats ships déployés, dans le code de l'interface graphique.

On initialise ensuite le nombre de tours avant de pouvoir changer de position à 5. (l54).

Fonctionnalité changement de position

Pour cette fonctionnalité on a une fonction ***canChangePosition()***, qui vérifie que le ship a bien attendu assez de tours avant de pouvoir changer de place.

La fonction ***changePosition()*** permet de faire le changement de position des ships du joueur (qui a appuyé sur le bouton) uniquement. Pour cela, on va mettre à jour le ***board*** en supprimant les anciennes positions puis en ajoutant les nouvelles obtenues avec ***ship.place()*** (si elles sont correctes sinon on en choisit de nouvelles). Enfin, on met à jour la map des ships en questions avec la méthode ***ship.newPlace()***.

```
function canChangePosition() private view returns (bool b){
  for (uint i = 0; i < index; i++) {
    if (game.xs[i] < 0) continue;
    if (turns[i] <= 0 && owners[i] == msg.sender){
      return true;
    }
  }
  return false;
}

/*
function to change the position of ship of the playeer
*/
function changePosition() external{
  require(canChangePosition(),"you cannot change positions");
  for (uint i = 0; i < index; i++) {
    if (game.xs[i] < 0) continue;

    // find a ship which can move
    if (turns[i] <= 0 && owners[i] == msg.sender){
      turns[i] = 5; // reset his counter
      Ship ship = Ship(ships[i]);
      uint prev_x = uint(game.xs[i]); // store previous valeur of x
      uint prev_y = uint(game.ys[i]); // store previous valeur of y
      game.board[uint(game.xs[i])][uint(game.ys[i])] = 0; // delete the position in the board

      // get a new position
      (uint x, uint y) = ship.place(game.width, game.height);
      bool invalid = true;
      while (invalid) {
        if (game.board[x][y] == 0) {
          game.board[x][y] = i;
          game.xs[i] = int(x);
          game.ys[i] = int(y);
          invalid = false;
        } else {
          uint newPlace = (x * game.width) + y + 1;
          x = newPlace % game.width;
          y = newPlace / game.width % game.height;
        }
      }
      ship.newPlace(prev_x,prev_y,x, y); // update the map of the ship
      emit Moved(uint(i), prev_x, prev_y, owners[i], uint(x), uint(y)); // to update the frontend
    }
  }
}
```

Une fois le changement effectué, on émet un événement pour indiquer le changement au frontend, et mettre à jour la vue.

Fonctionnalité de communication

L'objectif de cette fonctionnalité est de mettre en place un système de communication entre les ships d'un même joueur afin d'éviter tout tirs contre des alliés (ships du même joueur) et d'augmenter les chances de toucher un ship ennemi.

Cette fonction est invoquée par le bouton ***Communicate***, de la même manière que pour les fonctions register et turn.

Pour implémenter la fonctionnalité, on doit d'abord connaître si un owner a bien 2 ships, afin qu'ils puissent communiquer. Ceci est fait dans la fonction numberShipInPlay (L179 → L190) :

```
179+  function numberShipInPlay() private view returns (uint){
180+    uint counter_ship = 0;
181+    for (uint i = 0; i < index; i++) {
182+      if (game.xs[i] < 0) continue;
183+
184+      // get ships of the player
185+      if (owners[i] == msg.sender){
186+        counter_ship += 1;
187+      }
188+    }
189+    return counter_ship;
190+  }
```

Par la suite on cherche les 2 ships appartenant au joueur (L196 à L232), puis la fonction ***communicate()*** dans ship.sol se charge de transmettre les coordonnées d'un ship à l'autre, ainsi que les coordonnées des tirs déjà effectués précédemment par chacun des ships.

```
196+ function communication() external{
197+   require(count[msg.sender] > 0, "do not possess ships");
198+   require(numberShipInPlay() >= 2, "need at least 2 ships");
199+   uint[] memory ships_of_owner = new uint[](index);
200+   uint counter = 0;
201+
202+   // search allies
203+   for (uint i = 0; i < index; i++) {
204+     if (game.xs[i] < 0) continue;
205+
206+     // get ships of the player
207+     if (owners[i] == msg.sender){
208+       ships_of_owner[counter] = i;
209+       counter += 1;
210+     }
211+   }
212+
213+   // communication between allies
214+   for (uint i = 0; i < counter; i++) {
215+     Ship ship = Ship(ships[ships_of_owner[i]]); // get the ship at the index ships_of_owner[i]
216+     for (uint j = 0; j < counter; j++) {
217+       if (i != j){
218+         Ship allie = Ship(ships[ships_of_owner[j]]); // get the ship at the index ships_of_owner[j]
219+         // get information of all his map
220+         for (uint x; x<game.height; x+=1){
221+           for (uint y; y<game.width; y+=1){
222+             uint value = allie.getValue(x,y); // get value for the position (x,y)
223+             ship.communicate(x,y,value); // communicate --> update ship's map
224+           }
225+         }
226+       }
227+     }
228+
229+     //reset map of the ship if the map is full
230+     ship.checkReset(i);
231+   }
232+ }
```

De cette manière on ne peut pas cibler un bateau allié, ni tirer là où un bateau allié a déjà tiré. Cela permet d'augmenter grandement les performances du joueur.

Dans ship.sol

Nous avons implémenté un constructeur, qui permet d'identifier le joueur propriétaire du ship (ici o).

On voulait aussi identifier les ships du même joueur. On a donc utilisé l'index unique attribué dans main.sol pour cela (ici idx).

```
15+ constructor(address o, uint idx){
16+   owner = o;
17+   shipId = idx;
18+ }
```

Variables

Pour Ship.sol on commence par introduire les variables dont on a besoin pour implémenter nos fonctionnalités. La variable la plus importante est map, qui permet d'associer un code à chaque couple de coordonnées (x.,y). Voici la définition des codes pour la map :

- la valeur 0 représente une absence d'informations, on ne sait rien sur cette case
- la valeur 1 représente notre position (la position du ship)
- la valeur 2 représente que le ship à tiré sur cette position
- la valeur 3 représente qu'à cette position se situe un allié (un autre ship du même joueur)

De plus, pour des raisons de performances, nous maintenons le nombre de positions à "explorer" (positions ayant la valeur 0 dans notre map). Ainsi, avant de tirer, nous pouvons vérifier que cela est possible en analysant la valeur de la variable ***availablePosition.***

```
6+ contract Ship {
7+   address private owner;
8+   mapping(uint => mapping(uint => uint)) private map; // 0 : no informations ; 1 : my ship ; 2 : target fired ; 3 : ship allies ;
9+   uint private w;
10+  uint private h;
11+  uint private shipId = 0;
12+  uint private counter = 0;
13+  uint private availablePosition = 0;
14+
```

Fonctions utilitaires

Les variables sont suivies par quelques fonctions utilitaires. La fonction ***random()*** prend l'index et le owner du ship transmis par le main et un compteur pour créer des valeurs pseudo random différent à chacun des appels.

```
// ---------------------- UTILS FUNCTIONS ---------------------

/*
Function to return a random integer
*/
function random() private returns (uint){
  counter+=1024;
  return uint(keccak256(abi.encode(owner,counter*shipId)));
}

/*
Function to share a value of our map for the position (x,y)
*/
function getValue(uint x, uint y) public view returns (uint){
  return map[x][y];
}

/*
Method to print the map of the ship
*/
function printMap() public view{
  for (uint x; x<h; x+=1){
    for (uint y; y<w; y+=1){
      console.log("(%s,%s)=%s",x,y,map[x][y]);
    }
  }
}
```

Fonctions de bases

On définit ensuite les méthodes de base associées aux ships.

La Fonction ***update()*** permet de mettre à jour la map avec la position du ship déployé, et d'initialiser le nombre de cases disponibles dans la map du ship en question.

La fonction ***fire()***, permet de tirer sur une position de la map aléatoirement. Si dans la map d'un ship, on sélectionne une case ou l'on déjà tiré, alors on choisit une autre case. On décrémente aussi le nombre de cases disponibles et on marque cette position dans la map (du ship en question) pour éviter qu'un nouveau ship ne soit déployé sur cette case.

La fonction ***place()*** permet de trouver des coordonnées disponibles dans la map (du ship en question) pour un nouveau ship à déployer.

```
48+ // ---------------------- BASIS FUNCTIONS ---------------------
49+
50+ /*
51+ Function to update the position of the ship
52+ -->initial position can be impossible
53+ */
54+ function update(uint x, uint y) public{
55+   map[x][y] = 1;
56+   availablePosition = (w * h) - 1 ;
57+ }
58+
59+ /* Function to select a position to fire,
60+ this position is choosen randomly if this one
61+ have not be slected
62+ Return this position
63+ */
64+ function fire() public returns (uint, uint){
65+   uint get_h;
66+   uint get_w;
67+   bool isAlreadyTargeted = true;
68+
69+   while(isAlreadyTargeted){
70+     get_h = random() % h;
71+     get_w = random() % w;
72+
73+     if(map[get_h][get_w] == 0){
74+       isAlreadyTargeted = false;
75+     }
76+   }
77+   map[get_h][get_w] = 2;
78+   availablePosition = availablePosition - 1;
79+   return (get_h,get_w);
80+ }
```

```
81+
82+  /*
83+  Function to select a position for the ship, this position is choosen randomly
84+  This one may not be selected
85+  Return this position
86+  */
87+    function place(uint width, uint height) public returns (uint, uint){
88+    w = width;
89+    h = height;
90+
91+    uint get_h = random() % h;
92+    uint get_w = random() % w;
93+    bool found = true;
94+
95+    while(found){
96+      get_h = random() % h;
97+      get_w = random() % w;
98+
99+      if(map[get_h][get_w] == 0 || map[get_h][get_w] == 2){
100+       found = false;
101+     }
102+   }
103+
104+   return (get_h,get_w);
105+ }
```

Fonction de communication

Pour la communication entre ships du même joueur, on implémente la fonction ***communicate()***. Cette fonction permet au ship de mettre à jour sa map avec une valeur pour une position donnée. Cette valeur provient d'un ship allié (ship du même joueur). Si cette valeur est :

- 1 : cela signifie que c'est la position de ship allié alors on met à jour notre map avec la valeur 3 pour cette position.
- 2 : cela signifie que notre allié a déjà tiré sur cette position, il est alors inutile de la refaire. On met donc à jour notre map si nous n'avions pas d'informations sur cette position

```
// ---------------------- COMMUNICATION FUNCTIONS ---------------------

/*
Function to communicate with an allie. Indeed, we get knowledge from this one.
This allie send the value of his map for the position (x,y).
We can update our map.
*/
function communicate(uint x, uint y, uint value) public{

  // the position of an allie --> already seen
  if (value == 1 && map[x][y] > 0){
    map[x][y] = 3;
  }

  // the position of an allie --> never seen
  if (value == 1 && map[x][y] == 0){
    map[x][y] = 3;
    availablePosition = availablePosition - 1;
  }

  // the position of a fire of an allie
  if (value == 2 && map[x][y] == 0){
    map[x][y] = 2;
    availablePosition = availablePosition - 1;
  }
}
```

Fonction Changement de Position

Lorsque le ship change de position, il faut mettre à jour sa map.

La fonction ***newPlace()*** permet de faire cela, on supprime notre ancienne position et on update la map avec notre nouvelle position.

```
// ---------------------- CHANGE POSITION FUNCTION ---------------------

/*
Method to update a new position for the ship
*/
function newPlace(uint prev_x, uint prev_y, uint n_x, uint n_y) public{
  map[prev_x][prev_y] = 0;
  //console.log("new position (%s,%s)",n_x,n_y);
  if (map[n_x][n_y] != 0){
    availablePosition += 1;
  }
  map[n_x][n_y] = 1;
}

```

Fonctions Reset Map

Les fonctions suivantes sont liées au reset de la map associée à chaque contrat ship. En effet, plusieurs fonctions du contrats cherchent une place disponible dans la map, et celle-ci se remplit au fur et à mesure d'informations. Il est tout à fait possible que celle-ci soit pleine. Il est donc impératif d'avoir une manière de réinitialiser la map.

La fonction ***reset()*** permet d'effacer le contenu de la map à l'exception de la position du ship en question.

La fonction ***checkReset()*** permet de reset la map (à l'aide la fonction ci-dessus) si la map du ship est remplie d'informations.

```
143+  // ---------------------- RESET MAP FUNCTIONS ---------------------
144+
145+  /*
146+  Metho to reset the ship's map except his position
147+  */
148+  function reset() private{
149+    for (uint x; x<h; x+=1){
150+      for (uint y; y<w; y+=1){
151+        if (map[x][y] != 1){
152+          map[x][y] = 0;
153+        }
154+      }
155+    }
156+    availablePosition = (h * w) - 1;
157+  }
158
159+  /*
160+  Method to check if the ship must to reset his map and reset it
161+  */
162+  function checkReset(uint index) public{
163+    if (availablePosition == 0){
164+      //console.log("ship %s have to reset",index);
165+      reset();
166+    }
167+  }
168  }
```

Dans app.tsx :

Ce fichier génère l'interface graphique utilisateur.

Comme précédemment mentionné, on ajoute plusieurs boutons, qui permettent d'invoquer les méthodes dans le contrat main.sol. Ces boutons sont définis dans les lignes 175 à 178.

De plus, on implémente un listener sur l'événement ***onMoved***, qui est enclenché par la méthode ***changePosition()*** dans main.sol. Afin de pouvoir mettre à jour la grille du jeu,

avec le nouvel emplacement d'un ship, on effectue les changements des lignes 96 à 123, 156, 160 et 165.

```
96+   const onMoved = (
97+     ship: BigNumber,
98+     prev_x: BigNumber,
99+     prev_y: BigNumber,
100+    owner: string,
101+    x: BigNumber,
102+    y: BigNumber
103+  ) => {
104+    console.log('onMoved')
105+    setBoard(board => {
106+      return board.map((prev_x_, index) => {
107+        if (index !== prev_x.toNumber()) return prev_x_
108+        return prev_x_.map((prev_y_, indey) => {
109+          if (indey !== prev_y.toNumber()) return prev_y_
110+          return null
111+        })
112+      })
113+    })
114+    setBoard(board => {
115+      return board.map((x_, index) => {
116+        if (index !== x.toNumber()) return x_
117+        return x_.map((y_, indey) => {
118+          if (indey !== y.toNumber()) return y_
119+          return { owner, index: ship.toNumber() }
120+        })
121+      })
122+    })
123+  }
```

```
156+    await updateMoved()
157     console.log('Registering')
158     wallet.contract.on('Registered', onRegistered)
159     wallet.contract.on('Touched', onTouched)
160+    wallet.contract.on('Moved', onMoved)
161     return () => {
162       console.log('Unregistering')
163       wallet.contract.off('Registered', onRegistered)
164       wallet.contract.off('Touched', onTouched)
165+      wallet.contract.off('Moved', onMoved)
166     }
167   }, [wallet])
168   return board
169 }
170
171 const Buttons = ({ wallet }: { wallet: ReturnType<typeof useWallet> }) => {
172   const next = () => wallet?.contract.turn()
173   return (
174     <div style={{ display: 'flex', gap: 5, padding: 5 }}>
175+      <button onClick={() => wallet?.contract.changePosition()}>Change Position</button>
176+      <button onClick={() => wallet?.contract.communication()}>Communicate</button>
177+      <button onClick={() => wallet?.contract.register()}>Register</button>
178       <button onClick={next}>Turn</button>
179     </div>
180   )
181 }
182
```

Remarques

Il est nécessaire d'incrémenter le nonce à chaque transaction. Pour cela, il faut faire :

- aller dans MetaMask -> Paramètres -> advanced settings -> Customize transaction nonce -> enable

Bilan

En plus des fonctionnalitées de bases, nous avons implémenté 3 extensions :

- la communication entre les bateaux d'un même joueur
- le changement de position des bateaux
- sauvegarder les actions effectuées dans une map locale