

TP : Jeu de Devinette de Nombres

Objectif :

Créer un smart contract qui permet à un joueur de définir un nombre secret, et un autre joueur doit deviner ce nombre. Si la devinette est correcte, le joueur qui a deviné reçoit un prix en Ether.

Étape 1 : Code du Smart Contract

Voici le code pour un jeu simple où un joueur définit un nombre secret, et un autre joueur doit deviner ce nombre pour gagner de l'Éther.

1.1. Code du Smart Contract de Jeu

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract GuessTheNumber {

address public player1; // Le joueur qui propose le nombre secret

address public player2; // Le joueur qui doit deviner

uint256 public secretNumber; // Le nombre secret choisi par player1

**uint256 public reward; // La récompense que player2 va gagner s'il devine
correctement**

bool public gameStarted; // Indique si le jeu a commencé

bool public gameEnded; // Indique si le jeu est terminé

// Événements pour notifier les actions

event GameStarted(address player1, address player2, uint256 reward);

event Player2Guessed(address player2, uint256 guessedNumber, bool correct);

event GameEnded(address winner, uint256 amount);

```
modifier onlyPlayer1() {
```

```
    require(msg.sender == player1, "Only Player 1 can perform this action.");
```

```
    _;
```

```
}
```

```
modifier onlyPlayer2() {
```

```
    require(msg.sender == player2, "Only Player 2 can perform this action.");
```

```
    _;
```

```
}
```

```
modifier gameNotStarted() {
```

```
    require(!gameStarted, "The game has already started.");
```

```
    _;
```

```
}
```

```
modifier gameStartedOnly() {
```

```
    require(gameStarted && !gameEnded, "The game has not started or has already ended.");
```

```
    _;
```

```
}
```

```
modifier gameEndedOnly() {
```

```
    require(gameEnded, "The game is still ongoing.");
```

```
    _;
```

```
}
```

```
// Fonction pour initier le jeu
```

```
function startGame(address _player2, uint256 _secretNumber) public payable  
gameNotStarted {
```

```
    require(msg.value > 0, "You must send Ether to start the game.");
```

```
    player1 = msg.sender;
```

```
    player2 = _player2;
```

```
    secretNumber = _secretNumber;
```

```
    reward = msg.value;
```

```
    gameStarted = true;
```

```
    emit GameStarted(player1, player2, reward);
```

```
}
```

```
// Fonction pour que Player 2 fasse une tentative de devinette
```

```
function guess(uint256 _guessedNumber) public payable gameStartedOnly  
onlyPlayer2 {
```

```
    require(msg.value == 0.1 ether, "You must send exactly 0.1 ETH to guess.");
```

```
    bool correct = _guessedNumber == secretNumber;
```

```
    emit Player2Guessed(msg.sender, _guessedNumber, correct);
```

```
    if (correct) {
```

```
        // Si Player 2 devine correctement, il gagne la récompense
```

```
        payable(player2).transfer(reward);
```

```
        gameEnded = true;
```

```
        emit GameEnded(player2, reward);
```

```
    } else {
```

```
        // Sinon, Player 2 perd sa tentative (il ne reçoit rien)
```

```

        payable(player1).transfer(0.1 ether); // Player1 reçoit la tentative de Player2
    }
}

// Fonction pour que Player1 révèle le nombre secret et termine le jeu
function revealGame() public onlyPlayer1 gameStartedOnly {
    gameEnded = true;

    payable(player1).transfer(address(this).balance); // Player1 récupère le montant
    restant

    emit GameEnded(player1, address(this).balance);
}

// Fonction pour vérifier l'état du jeu
function getGameStatus() public view returns (string memory) {
    if (gameEnded) {
        return "Game Ended";
    }

    if (gameStarted) {
        return "Game Started, Waiting for Player 2's Guess";
    }

    return "Game Not Started";
}
}

```

Explication du Code :

1. Déclaration des variables :

- **player1** et **player2** sont les deux joueurs du jeu. **player1** définit un nombre secret, et **player2** essaie de deviner ce nombre.
- **secretNumber** est le nombre secret choisi par **player1**.

- **reward** est le montant de l'Ether mis en jeu par **player1**, que **player2** peut gagner s'il devine correctement.
 - **gameStarted** et **gameEnded** sont des indicateurs qui montrent si le jeu est en cours ou terminé.
2. Modificateurs :
- **onlyPlayer1** : Permet d'assurer que seule l'adresse de **player1** peut exécuter certaines actions.
 - **onlyPlayer2** : Permet d'assurer que seule l'adresse de **player2** peut deviner le nombre.
 - **gameNotStarted** : Assure que le jeu n'a pas encore commencé.
 - **gameStartedOnly** : Permet de garantir que certaines actions ne peuvent être effectuées que lorsque le jeu est en cours.
 - **gameEndedOnly** : Permet de vérifier que l'action est effectuée uniquement après que le jeu ait pris fin.
3. Fonctions principales :
- **startGame** : Permet à **player1** de commencer le jeu en envoyant une certaine quantité d'Ether. **player2** doit aussi être spécifié à ce moment-là.
 - **guess** : Permet à **player2** de deviner le nombre. **player2** doit envoyer exactement **0.1 ETH** pour faire une tentative. Si la réponse est correcte, **player2** gagne la récompense, sinon, il perd la tentative (le montant de **0.1 ETH** est envoyé à **player1**).
 - **revealGame** : Si **player2** ne devine pas le nombre secret, **player1** peut appeler cette fonction pour récupérer la récompense restante.
 - **getGameStatus** : Permet à n'importe qui de vérifier l'état actuel du jeu.
-

Étape 2 : Déploiement et Interaction

Pour déployer ce contrat et interagir avec lui, vous pouvez utiliser Remix IDE comme suit :

1. Accédez à Remix IDE (Remix Ethereum IDE).
2. Créez un fichier **GuessTheNumber.sol** et collez-y le code ci-dessus.
3. Compilez le contrat :
 - Sélectionnez la version de Solidity **0.8.x** dans le compilateur.
 - Cliquez sur "Compile GuessTheNumber.sol".
4. Déployez le contrat :
 - Allez dans l'onglet "Deploy & Run Transactions".
 - Choisissez l'environnement de déploiement "JavaScript VM" pour une simulation locale.
 - Cliquez sur "Deploy".
5. Interagissez avec le contrat :

- Utilisez la fonction `startGame` pour initier une partie (en envoyant de l'Ether comme mise initiale).
 - Le joueur 2 peut deviner avec la fonction `guess` (en envoyant 0.1 ETH).
 - Vérifiez le statut du jeu avec `getGameStatus`.
 - Si le jeu est terminé, vous pouvez appeler `revealGame` pour que le joueur 1 récupère l'Ether restant.
-

Étape 3 : Tester le Jeu

1. Démarrez le jeu en appelant `startGame` avec l'adresse du joueur 2 et un nombre secret (par exemple, `42`) et en envoyant une mise (par exemple, `1 ETH`).
 2. Joueur 2 tente une devinette en appelant `guess(42)` ou toute autre valeur et en envoyant `0.1 ETH`.
 3. Vérifiez les résultats :
 - Si `player2` devine correctement, il gagne la récompense.
 - Si `player2` ne devine pas correctement, la tentative est perdue, et l'Ether est transféré à `player1`.
-

Conclusion

Vous avez maintenant un jeu décentralisé simple où un joueur propose un nombre secret, et un autre joueur doit deviner ce nombre pour gagner de l'Ether. Ce type de jeu peut être étendu avec davantage de fonctionnalités, telles que des niveaux de difficulté, un système de mise plus complexe ou un contrat plus robuste pour gérer les différentes conditions du jeu.