How to Create a Decentralized Finance Blackjack Game

Blackjack is one of the most popular casino games in the world. You play by trying to beat the dealer's hand with a hand of cards that add up to as close to 21 as possible without going over. In DeFi Blackjack, you can play the game using cryptocurrency instead of traditional fiat currency.

Players will “bet” an amount for each round they play - if they win, they get back twice the amount. If they lose or face a draw, they forfeit the amount to the prize pool.

## **Tools**

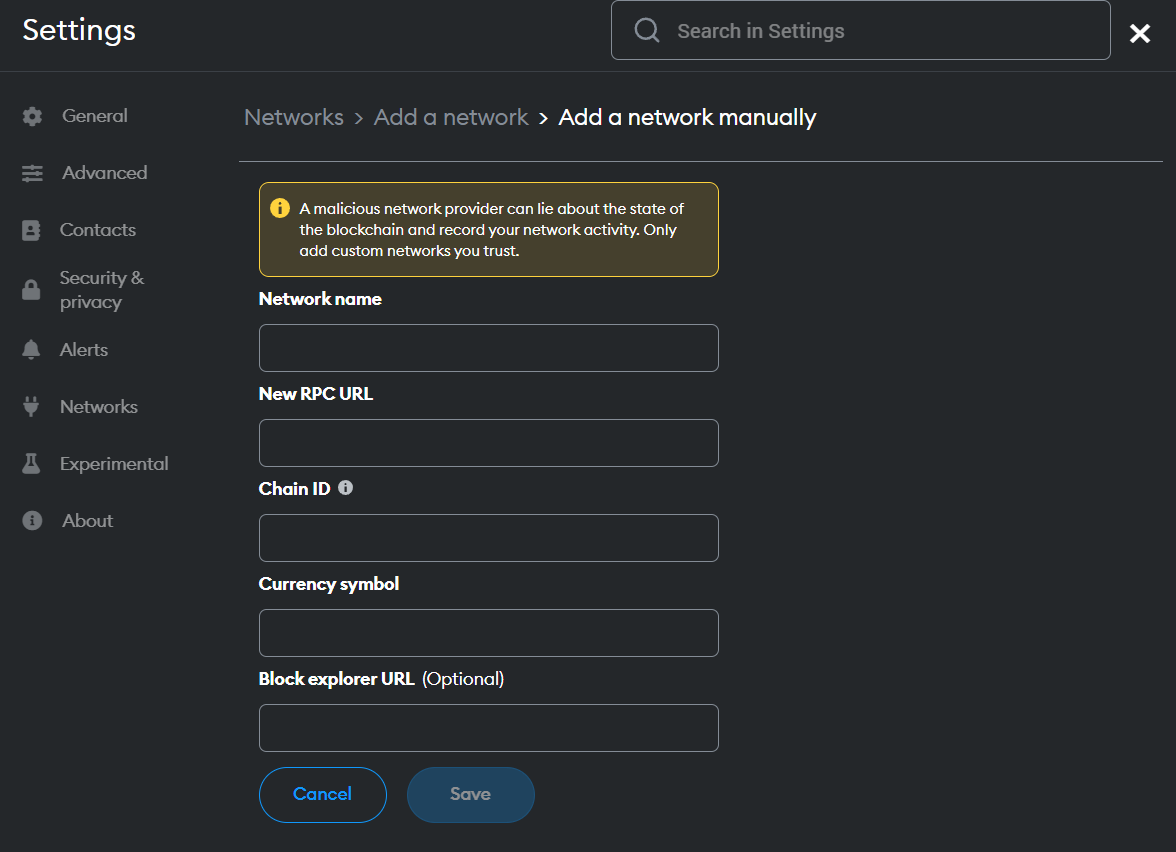
* [MetaMask](https://metamask.io/) wallet (download, installation and sign-up required)
* [Remix](https://remix.ethereum.org/) IDE
* a front-end IDE [CodeSandbox](https://codesandbox.io/) (recommended) or Visual Studio Code
* [Polygon Faucet](https://faucet.polygon.technology/)
* [Polygon Scan](http://mumbai.polygonscan.com)
* [Ethereum Unit Converter](https://eth-converter.com/)

## **Step 1: Setting up your MetaMask**

Sign up for MetaMask and install the browser extension if you haven’t yet.

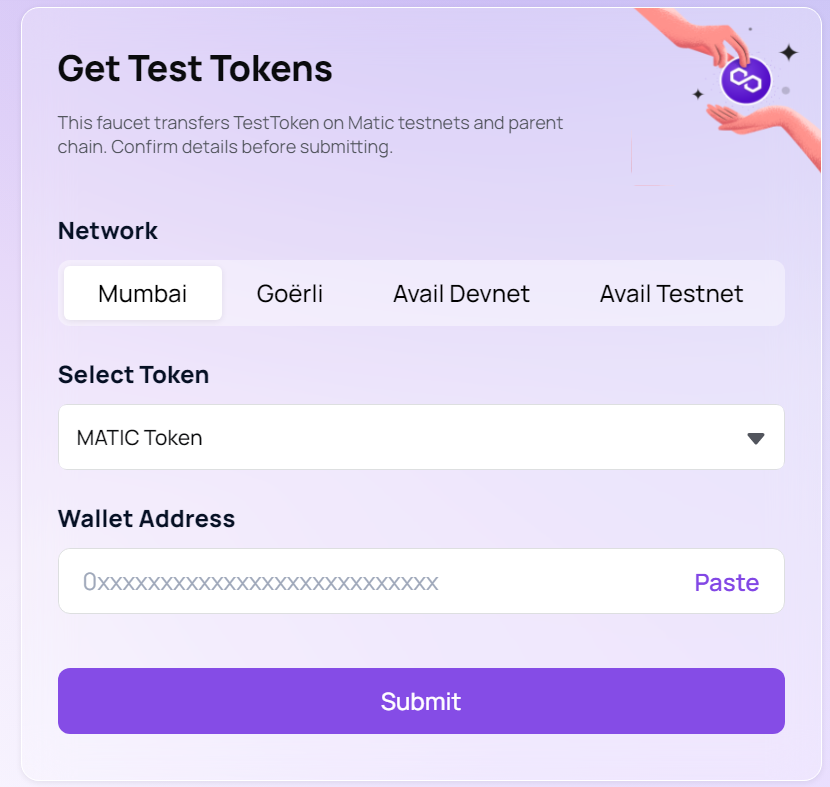
To execute transactions on the blockchain, we will be using the Mumbai (Polgyon) network.

1. Click on the Metamask icon in your browser's toolbar to open the extension.
2. Click on the network dropdown menu on the top of the extension and select "Add Network."



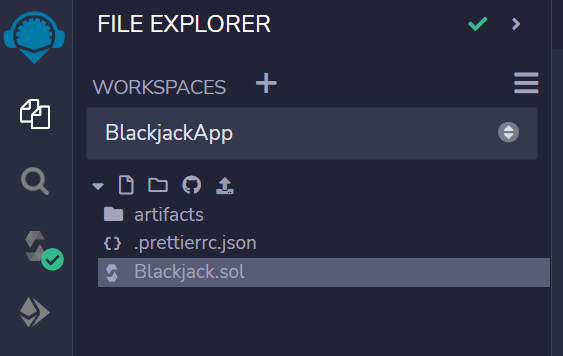
1. On the left, click the "Network" tab, and fill in the following fields:
   * Network Name: Mumbai
   * New RPC URL:https://rpc-mumbai.maticvigil.com
   * Chain ID: 80001
   * Symbol: MATIC
   * Block Explorer URL:https://mumbai.polygonscan.com
2. Click "Save".

Go to<https://faucet.polygon.technology/> and provide your public wallet address, to obtain test tokens (Matic) for the Mumbai network.



## **Step 2: Creating your Solidity contract**

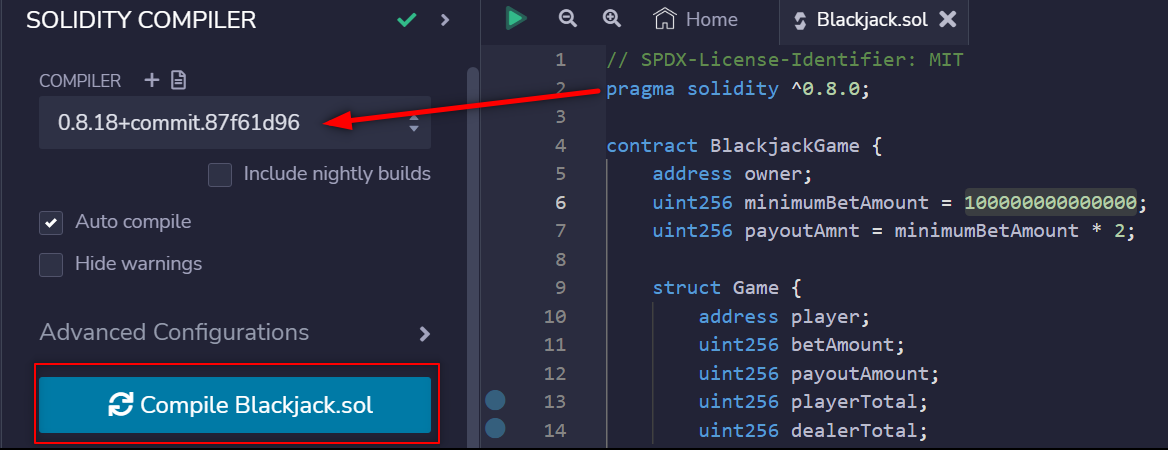
Go to Remix IDE and create your Solidity contract. Name it Blackjack.sol



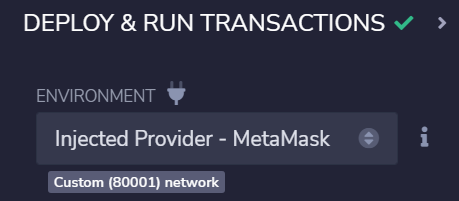
In your code editor, insert the following Solidity code:

| *// SPDX-License-Identifier: MIT* pragma solidity ^0.8.0;  contract BlackjackGame {  address owner;  uint256 minimumBetAmount = 100000000000000;  uint256 payoutAmnt = minimumBetAmount \* 2;   struct Game {  address player;  uint256 betAmount;  uint256 payoutAmount;  uint256 playerTotal;  uint256 dealerTotal;  }   mapping(address => Game) games;  mapping(address => uint256) balances;   modifier onlyOwner() {  require(owner == msg.sender, "Only the owner can call this function");  \_;  }   constructor() {  owner = msg.sender;  }   function placeBet() public payable {  require(  msg.value >= minimumBetAmount,  "Not enough balance for entry fee"  );  uint256 amount = msg.value;  require(msg.value >= minimumBetAmount, "The bet amount is too low.");   balances[address(this)] += amount;  games[msg.sender] = Game(msg.sender, amount, 0, 0, 0);  }   function payout() public {  require(  balances[address(this)] >= payoutAmnt,  "Not enough prize money."  );   address payable player = payable(games[msg.sender].player);  \_transfer(player, payoutAmnt);  balances[address(this)] -= payoutAmnt;  }   function deposit() public payable virtual {  require(msg.value > 0, "You don't have enough tokens");  balances[address(this)] += msg.value;  }   function \_transfer(address to, uint256 amount) internal virtual {  require(address(this) != address(0), "transfer from the zero address");  require(to != address(0), "transfer to the zero address");  unchecked {  balances[address(this)] = address(this).balance - amount;  (bool success, ) = to.call{value: amount}("");  require(success, "Failed to withdraw entry fee");  balances[to] += amount;  }  }   function balanceOf(address \_address) public view returns (uint256) {  return balances[\_address];  }   function withdraw() public onlyOwner {  (bool success, ) = msg.sender.call{value: address(this).balance}("");  require(success, "Failed to withdraw balance");  balances[address(this)] = address(this).balance;  } } |
| --- |

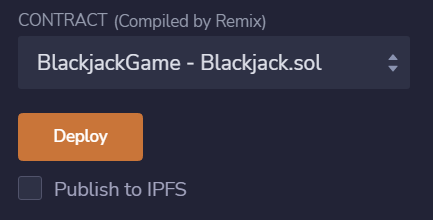
Go to the Compiler menu. Ensure your Compiler version is equal to or higher than the Solidity version we’re using. Click Compile Blackjack.sol



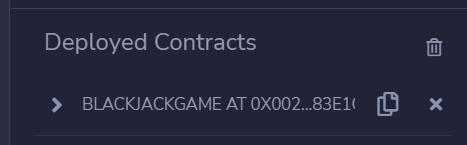
Go to the Deploy & Transactions menu. Switch your environment to “Injected Provider - MetaMask” while making sure you are on the Mumbai Test Network.



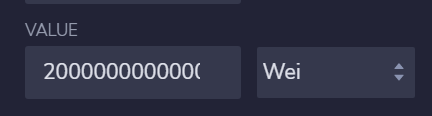
Deploy your contract. You’ll need to approve gas fees using the test tokens we acquired.



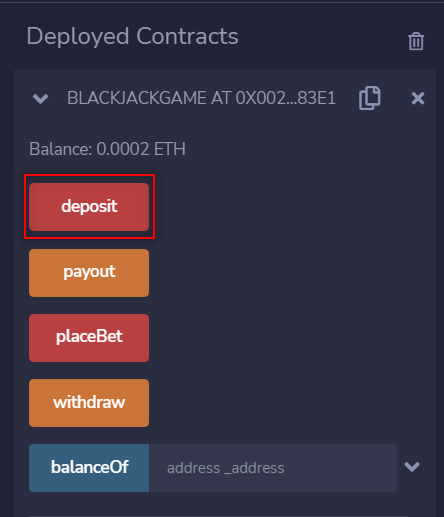
Once successful. Your contract should appear under “Deployed Contracts.” You will need to copy your contract address from here later.



Use the deposit function to send some initial prize money to your contract. Paste 200000000000000 into the value input and make sure it says “Wei.”



Under your deployed contract, click on “deposit.”



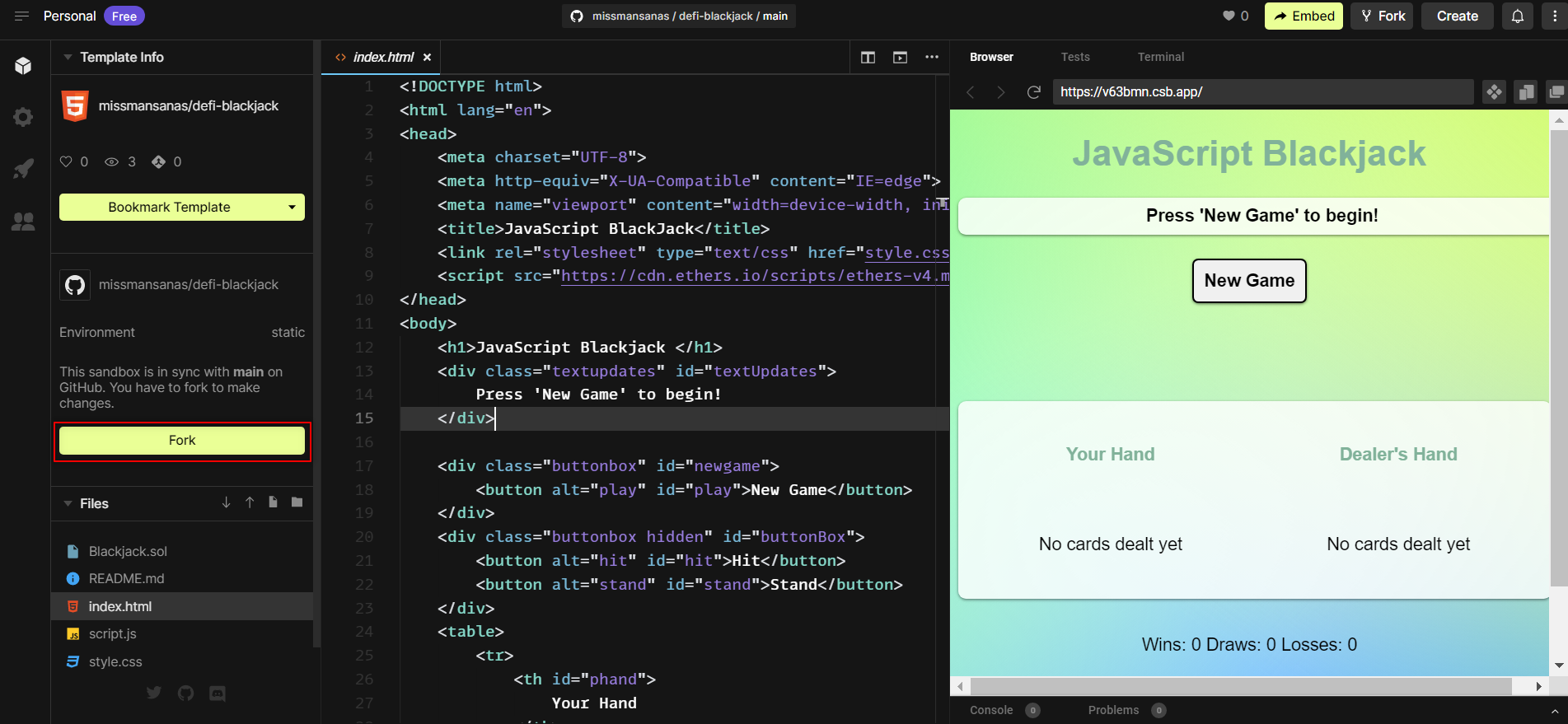
Your balance should update from 0.00 ETH to 0.0002 ETH in a minute or two. This will be your initial prize money pool.

## **Step 3: Coding your front-end**

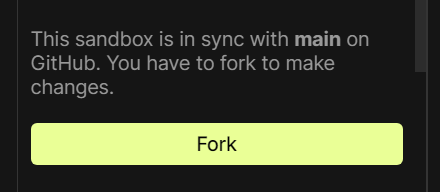
Open your front-end IDE and fork the front-end code:

**Option 1: If you’re using CodeSandbox.io (Recommended for beginners)**

Open <https://githubbox.com/missmansanas/defi-blackjack>. It will automatically clone the repository to a sandbox.



Click on Fork. You now have your own copy of the front-end code with a live preview enabled whenever you save the file.



**Option 2: If you’re using Visual Studio Code**

**❗ Important ❗ This option is only recommended if you’re already comfortable with Visual Studio Code and Command Line Interface.**

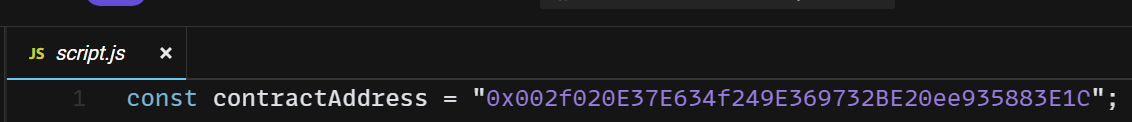
Open your terminal and type the following command:

| git clone https://github.com/missmansanas/defi-blackjack |
| --- |

You now have a copy of the front-end code on your local machine. You will have to run the code or use the Live Server extension to preview your code.

Next, navigate to the script.js file and find this line, it should be the very first.

Replace the string within quotes with your own deployed contract address from Remix.



Save all changes. You can now start playing your game!