Creating Smart Contracts with Solidity

Creating simple lottery contract

Designing a lottery game

What is a Lottery

- A gambling game where prizes are won based on some random number game
- Has many different variants but most common is buying tickets with numbers on them , and after some time winning numbers are drawn and prizes are distributed based on that.

Desired characteristic

- We want a lottery with the following characteristics defined
  - Chance of winning - what is the chance of winning a prize
  - Win ratio – in case of winning how much is the winner taking

Disclaimer

- Not a real lottery
- Since we will be only exercising, we are going to simplify a lot our lottery
  - No numbers drawing, no tickets buying - you send money and there is instant decision whether you win or loose, and the prize is sent to your account straight away

Rules

- Player sends currency to the contract address
- A random decision with certain probabilities is made whether the player wins or not - 50% chance to win
- If winning
  - Second random decision is made for the size of the prize – prizes should vary from 1/5th of the ticket price to 2x the ticket price
  - The prize amount is sent to player address
- If loosing nothing happens

What do we need

- Randomness
- Receiving currency
- Sending currency

Randomness in EVM and Solidity

- Doesn’t exist! There is no source of random numbers in EVM.
- Pseudo-random generators used in a lot of languages are not suitable because they are based on a seed. If the seed is public anybody can predict our lottery outcomes, making our lottery pointless

So how do we get random numbers?

- The most commonly used trick to get something that resembles random numbers is to use hashing function with input that has enough entropy in it
- We then use the output of the hashing function and teat it as integer, apply modulo and we get seemingly random integer
- The quality of the randomness depends only on the input to the hashing function

How does it look

```
uint(
    keccak256(
        abi.encodePacked(
            block.prevrandao,
            block.timestamp,
            .... // some more state variables
        )
    )
)
```

Let’s break it down

- abi.encodePacked – this function is simply creating a bytes array with all it’s arguments packed in a binary form
- keccak256 – this is main hashing algorithm used for hashing blocks in the Ethereum blockchain, it takes any amount of bytes and returns 256bit uint containing the hash value
- block.prevrandao – this property of the current block is a random number put there by the current signers of the block, it is required for the PoS algorithm and we can conveniently use it for our purpses
- block.timestamp – the current time of the block

Receiving currency

- We mentioned previously that each smart contract is also Ethereum address so it can receive currency. But how do we know when and how our contract is being sent currency ?
- payable modifier marks a function as the function that is able to receive currency. In the function we can use msg.value to get the value we received

Sending currency

- payable modifier can also be applied to and address , and in this case the address gets two additional functions transfer and send that allow sending currency and transactions accordingly
- The syntax for converting a regular address to address payable is : payable(<address>)

Let’s play