

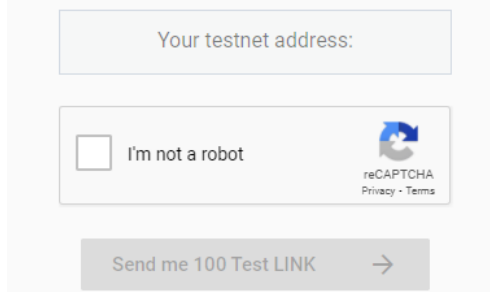
Reference

<https://docs.chain.link/docs/vrf-contracts/>

Rinkeby

LINK contract address: 0x01BE23585060835E02B77ef475b0Cc51aA1e0709

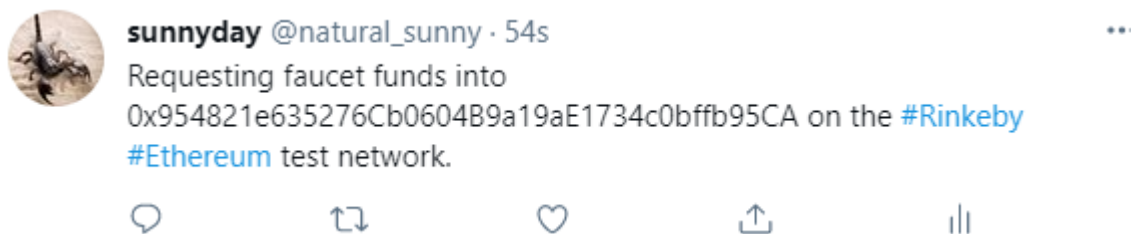
Chainlink Rinkeby Faucet

A screenshot of the Chainlink Rinkeby Faucet web form. It features a title 'Chainlink Rinkeby Faucet' at the top. Below the title is a text input field labeled 'Your testnet address:'. Underneath that is a reCAPTCHA verification box with the text 'I'm not a robot' and a checkbox. To the right of the checkbox is the reCAPTCHA logo and links for 'Privacy' and 'Terms'. At the bottom of the form is a button labeled 'Send me 100 Test LINK' with a right-pointing arrow.

LINK faucet: https://rinkeby.chain.link/?_ga=2.221617247.693657221.1623225557-1592464978.1623225557

ETH faucet: <https://faucet.rinkeby.io/>
<http://rinkeby-faucet.com/>

To get eth from the first address, create a tweet like this, get the tweet url and put in the faucet box.



TASK – Deploy Smart Contract

Deploy the following Random Number Generator with ChainLink on Rinkeby Testnet. You need ETH and LINK tokens. Remember to give LINK tokens to the contract.

Test that it generates a random number between 0 and 99.

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.6.6;

import "https://github.com/smartcontractkit/chainlink/blob/master/evm-
contracts/src/v0.6/VRFConsumerBase.sol";
import "https://github.com/smartcontractkit/chainlink/blob/master/evm-
contracts/src/v0.6/Owned.sol";

/**
 * PLEASE DO NOT USE THIS CODE IN PRODUCTION.
 */
contract RandomNumberConsumer is VRFConsumerBase {

    bytes32 internal keyHash;
    uint256 internal fee;
    bytes32 public reqId;
    uint256 public randomResult;

    event RandomNumberReceived(bytes32 indexed requestId, uint256 indexed results);

    /**
     * Constructor inherits VRFConsumerBase
     *
     * Network: Rinkeby
     * Chainlink VRF Coordinator address: 0xb3dCcb4Cf7a26f6cf6B120Cf5A73875B7BBc655B
     * LINK token address: 0x01BE23585060835E02B77ef475b0Cc51aA1e0709
     * Key Hash: 0x2ed0feb3e7fd2022120aa84fab1945545a9f2ffc9076fd6156fa96eaff4c1311
     *
     * REMEMBER to SEND Contract some LINK Token
     */
    constructor()
        VRFConsumerBase(
            0xb3dCcb4Cf7a26f6cf6B120Cf5A73875B7BBc655B, // VRF Coordinator
            0x01BE23585060835E02B77ef475b0Cc51aA1e0709 // LINK Token
        ) public
    {
        keyHash = 0x2ed0feb3e7fd2022120aa84fab1945545a9f2ffc9076fd6156fa96eaff4c1311;
        fee = 0.1 * 10 ** 18; // 0.1 LINK (Varies by network)
    }

    /**
     * Requests randomness from a user-provided seed
     */
    function getRandomNumber(uint256 userProvidedSeed) public returns (bytes32 requestId) {
        require(LINK.balanceOf(address(this)) >= fee, "Not enough LINK - fill contract with faucet");
        return requestRandomness(keyHash, fee, userProvidedSeed);
    }

```

```

    }

    /**
     * Callback function used by VRF Coordinator
     */
    function fulfillRandomness(bytes32 requestId, uint256 randomness) internal override {
        reqId=requestId;
        randomResult = randomness % 100;
        emit RandomNumberReceived(reqId, randomResult);
    }
}

```

TASK – Create a Blockchain Game

Create a game contract called **RandomBattle**. This contract acts as a database to store our hero details and a battle arena where heroes can fight against each other.

Create a hero data structure with name, DNA, level, win count, and loss count. Each hero will have a DNA number. Just like human DNA, each part of this number denotes different traits of a hero. For example, the first two digit map to the hero's looks, the second two digits maps to the flying ability, and so on.

```

struct Hero {
    string name;
    uint dna;
    uint32 level;
    uint16 winCount;
    uint16 lossCount;
}
Hero[] public heroes;

```

To identify the owner of each hero, create a mapping with the herold as the key and the owner address as the value. This mapping can also be used in a function modifier for restricting hero access.

Declare a function to create new heroes. The function should accept a name and DNA to create a hero. Create a new hero using the input and push it to the heroes array. A push operation in solidity will return the new array index. We can use this index as the herold for easy identification. Save the new ID to hero to owner mapping.

```

/**
 * @dev Create a new hero
 * @param _name Name of the hero
 * @param _dna DNA of the hero
 */
function createHero(string calldata _name, uint _dna) external

```

We have a battle system where one hero can attack another to advance in level. Create another function that can be used to battle. Use the modifier to restrict it only to the hero's owner.

Let's design the battle system in such a way that the attacker has a 60% chance of winning against the target hero. Create a *random* number from 0 to 100 using **Chainlink VRF**. The winner is chosen by comparing the random number and probability of winning. The win/loss count and hero level should be changed based on the result. You may vary this procedure according to your needs.

Finally, emit an event to notify the listeners.

```
/**
 * @dev Adds single address to whitelist.
 * @param _herold Attacker hero id
 * @param _targetId Target hero id
 */
function attack(uint _herold, uint _targetId) external ownerOf(_herold)
```

Also create also a function that returns the current winning hero ID.

Note:

Some of the variables are not used. They may be used in future updates.

A version of the above implementation is available at Rinkeby Testnet Address:

0x6Bbc18Bc2463acA3a29Da27d14B3f79E51132Be7

If you want to load the above test contract, copy this abi to remix before you load the smart contract at the given address above. Open a new file as an abi file, copy and paste from below, then load the contract address (you need to keep the abi tab active).

```
[
  {
    "inputs": [],
    "name": "acceptOwnership",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [],
    "stateMutability": "nonpayable",
    "type": "constructor"
  },
  {
    "anonymous": false,
    "inputs": [
```

```

        {
            "indexed": true,
            "internalType": "address",
            "name": "from",
            "type": "address"
        },
        {
            "indexed": true,
            "internalType": "address",
            "name": "to",
            "type": "address"
        }
    ],
    "name": "OwnershipTransferRequested",
    "type": "event"
},
{
    "anonymous": false,
    "inputs": [
        {
            "indexed": true,
            "internalType": "address",
            "name": "from",
            "type": "address"
        },
        {
            "indexed": true,
            "internalType": "address",
            "name": "to",
            "type": "address"
        }
    ],
    "name": "OwnershipTransferred",
    "type": "event"
},
{
    "inputs": [
        {
            "internalType": "address",
            "name": "_to",
            "type": "address"
        }
    ],
    "name": "transferOwnership",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},

```

```
{
  "inputs": [],
  "name": "owner",
  "outputs": [
    {
      "internalType": "address",
      "name": "",
      "type": "address"
    }
  ],
  "stateMutability": "view",
  "type": "function"
}
]
```