Smart Contract and Unity Code

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.8.7;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

interface ERC20Token {
    //This interface defines several functions, which are often used when handling ERC-20 
Tokens

    function balanceOf(address account) external view returns (uint256);

    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    function transfer(address recipient, uint256 amount)
        external
        returns (bool);

    function approve(address spender, uint256 amount) external returns (bool);

    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);
}

contract JuicyGame {
    //Here, variables are defined, as well as the tokens that are used in the 
functionality of the smart contract

    address public admin = 0xc84577ac220DC5977186b6B690469F4b75358E4E;
    mapping(address => uint256) public playerTimesDeposited;
    address[] public allPlayers;
    mapping(address => uint256) public depositedBalance;

    ERC20Token public BUSD;
    ERC20Token public IGToken;
    ERC20Token public ENTRToken;

    constructor() public {
        //The tokens addresses are assigned to some names
        BUSD = ERC20Token(0xbCe98d116cA02A87a2E6c8EDf9597CEd50f3B0a2);
        IGToken = ERC20Token(0x5f310227dd9a9e65DaEb9d92282E27DD0eFcA02E);
        ENTRToken = ERC20Token(0xBB7DFc1aBbd94d53648e9DF1F7584B898b1D57C2);
    }

    function depositTokens(uint256 amount) public {
        //This function allows to deposit BUSD test-tokens into the Smart Contract called
Juicy Game
        uint256 playersUsdcBalance = BUSD.balanceOf(address(msg.sender));
        require(amount > 0, "Amount must be greater than zero");
        require(
            playersUsdcBalance >= amount * 1e18,
            "Not enough tokens in wallet"
        );

        BUSD.transferFrom(msg.sender, address(this), amount * 1e18);
        ENTRToken.transfer(msg.sender, amount * 1e18);

        /*Here are some variables that are utilised to track some data of the player, in
the if statement below,
        the players address is added to a list of all players
        */
        depositedBalance[msg.sender] += amount * 1e18;
        playerTimesDeposited[msg.sender]++;

        if (playerTimesDeposited[msg.sender] == 1) {
            allPlayers.push(msg.sender);
        }
    }

    function withdrawalPlayers(uint256 _amount) public {
        /*This function allows the withdrawal of BUSD from the Smart Contract, to check
whether the player
        is allowed to withdraw their tokens, the balance of the InGame-Token (IGToken) is
checked
        */

        uint256 dexBalance = BUSD.balanceOf(address(this));
        uint256 playersBalance = IGToken.balanceOf(address(msg.sender));

        require(playersBalance >= _amount, "Not enough tokens in wallet");
        require(_amount <= dexBalance, "Not enough tokens in reserve");

        BUSD.transfer(msg.sender, _amount);
        ENTRToken.transferFrom(msg.sender, address(this), _amount);
        IGToken.transferFrom(msg.sender, admin, _amount);

        depositedBalance[msg.sender] = 0;
    }

    function timesDeposited(address _address) public view returns (uint256) {
        //This function is used in order to check the amount of times a player already
deposit/played the game
        return playerTimesDeposited[address];
    }
}
```

Hochlader:

```python
from brownie import JuicyGame

from scripts.helpful_scripts import get_account


def deploy_fund_me():
    account = get_account()
    fund_me = JuicyGame.deploy({"from": account}, publish_source=True)
    print(f"Contract deployed to {fund_me.address}")


def main():
    deploy_fund_me()
```

Helpful scripts:

```python
from brownie import network, accounts, config


def get_account():
    if network.show_active() == "development":
        return accounts[0]
    else:
        return accounts.add(config["wallets"]["from_key"])
```

config:

```yaml
dependencies:
  - smartcontractkit/chainlink-brownie-contracts@0.2.1
  - OpenZeppelin/openzeppelin-contracts@3.4.0
compiler:
  solc:
    remappings:
      - "@chainlink=smartcontractkit/chainlink-brownie-contracts@0.2.1"
      - "@openzeppelin=OpenZeppelin/openzeppelin-contracts@3.4.0"
dotenv: .env
networks:
  default: development
  rinkeby:
    eth_usd_price_feed: "0x035dCD3b056BdDbf82273A1b93c7B8cd25614995"
    verify: True
  goerli:
    eth_usd_price_feed: "0xD4a33860578De61DBAbDc8BFdb98FD742fA7028e"
    verify: True
  mainnet-fork-dev:
    eth_usd_price_feed: "0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419"
    verify: False
  development:
    verify: False
  ganache-local:
    verify: False
wallets:
  from_key: ${PRIVATE_KEY}
```

ChangeSceneScript:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ChangeSceneScript : MonoBehaviour
{
    public void LoadScene(string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }
}
```

ERC20BalanceOfENTR:

```
using System.Collections;
using System.Numerics;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class ERC20BalanceOfENTR : MonoBehaviour
{
    //[SerializeField] GameObject _enterButton;


    string chain = "binance";
    string network = "testnet";
    string contract = "0xBB7DFc1aBbd94d53648e9DF1F7584B898b1D57C2";

    int counter = 0;


    async void Update()
    {

        string account = PlayerPrefs.GetString("Account");
        BigInteger balanceOf = await ERC20.BalanceOf(chain, network, contract, account);
        print(balanceOf);



        if (balanceOf >= 5000000000000000000 & counter == 0)
        {
            SceneManager.LoadScene("Menu");
            //_enterButton.SetActive(true);
            counter++;
        }
        else { }

    }
}
```

PlayAgainActivation1:

```
using System.Collections;
using System.Numerics;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class PlayAgainActivation1 : MonoBehaviour
{
    [SerializeField] GameObject _withdrawButton;
    [SerializeField] GameObject _playAgainButton;


    string chain = "binance";
    string network = "testnet";
    string contract = "0x5f310227dd9a9e65DaEb9d92282E27DD0eFcA02E";

    int counter = 0;


    float timer = 0f; // variable to keep track of the elapsed time

    async void Update()
    {
        timer += Time.deltaTime; // increment the timer by the time elapsed since the last
frame

        // check if the timer has reached 3 seconds
        if (timer >= 3f)
        {
            timer = 0f; // reset the timer

            string account = PlayerPrefs.GetString("Account");
            //string account = "0x035dCD3b056BdDbf82273A1b93c7B8cd25614995";
            BigInteger balanceOf = await ERC20.BalanceOf(chain, network, contract,
account);
            print(balanceOf);

            if (balanceOf < 50000 & counter == 0 & _withdrawButton.activeInHierarchy)
            {
                _playAgainButton.SetActive(true);
                counter++;
            }
            else { }
        }
    }
}
```

WebGLApprove20My:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Newtonsoft.Json;

#if UNITY_WEBGL
public class WebGLApprove20My : MonoBehaviour
{
    [SerializeField]
    private string contract = "0xbCe98d116cA02A87a2E6c8EDf9597CEd50f3B0a2";
    [SerializeField]
    private string spender = "0x7d5A6F6313633Ba8DdbbE893f893761C90dab0eA";
    [SerializeField]
    private string amount = "1000000000000000000000";
    private readonly string abi = “…”;

    async public void Approve()
    {
        // smart contract method to call
        string method = "approve";
        // array of arguments for contract
        string[] obj = { spender, amount };
        string args = JsonConvert.SerializeObject(obj);
        // value in wei
        string value = "0";
        // gas limit OPTIONAL
        string gasLimit = "";
        // gas price OPTIONAL
        string gasPrice = "";
        // connects to user's browser wallet (metamask) to send a transaction
        try
        {
            string response = await Web3GL.SendContract(method, abi, contract, args,
value, gasLimit, gasPrice);
            Debug.Log(response);
        }
        catch (Exception e)
        {
            Debug.LogException(e, this);
        }
    }
}
#endif
```

WebGLApproveIGToken:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Newtonsoft.Json;

#if UNITY_WEBGL
public class WebGLApproveIGToken : MonoBehaviour
{
    [SerializeField]
    private string contract = "0x5f310227dd9a9e65DaEb9d92282E27DD0eFcA02E";
    [SerializeField]
    private string spender = "0x7d5A6F6313633Ba8DdbbE893f893761C90dab0eA";
    [SerializeField]
    private string amount = "1000000000000000000000";
    private readonly string abi = “…”;

    async public void Approve()
    {
        // smart contract method to call
        string method = "approve";
        // array of arguments for contract
        string[] obj = { spender, amount };
        string args = JsonConvert.SerializeObject(obj);
        // value in wei
        string value = "0";
        // gas limit OPTIONAL
        string gasLimit = "";
        // gas price OPTIONAL
        string gasPrice = "";
        // connects to user's browser wallet (metamask) to send a transaction
        try
        {
            string response = await Web3GL.SendContract(method, abi, contract, args,
value, gasLimit, gasPrice);
            Debug.Log(response);
        }
        catch (Exception e)
        {
            Debug.LogException(e, this);
        }
    }
}
#endif
```

WebGLDeposit20:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Newtonsoft.Json;

#if UNITY_WEBGL
public class WebGLDeposit20 : MonoBehaviour
{
    [SerializeField]
    private string contract = "0x7d5A6F6313633Ba8DdbbE893f893761C90dab0eA";

    [SerializeField]
    private string amount = "5";
    private readonly string abi = "…";

    async public void Deposit()
    {
        // smart contract method to call
        string method = "depositTokens";
        // array of arguments for contract
        string[] obj = { amount };
        string args = JsonConvert.SerializeObject(obj);
        // value in wei
        string value = "0";
        // gas limit OPTIONAL
        string gasLimit = "";
        // gas price OPTIONAL
        string gasPrice = "";
        // connects to user's browser wallet (metamask) to send a transaction
        try
        {
            string response = await Web3GL.SendContract(method, abi, contract, args,
value, gasLimit, gasPrice);
            Debug.Log(response);
        }
        catch (Exception e)
        {
            Debug.LogException(e, this);
        }
    }
}
#endif
```

WebGLWinnerWithdrawal1:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Newtonsoft.Json;

#if UNITY_WEBGL
public class WebGLWinnerWithdrawal1 : MonoBehaviour
{
    [SerializeField]
    private string contract = "0x7d5A6F6313633Ba8DdbbE893f893761C90dab0eA";

    [SerializeField]
    private string _amount = "5000000000000000000";
    private readonly string abi = "…";

    async public void withdrawalWinner()
    {
        // smart contract method to call
        string method = "withdrawalPlayers";
        // array of arguments for contract
        string[] obj = { _amount };
        string args = JsonConvert.SerializeObject(obj);
        // value in wei
        string value = "0";
        // gas limit OPTIONAL
        string gasLimit = "";
        // gas price OPTIONAL
        string gasPrice = "";
        // connects to user's browser wallet (metamask) to send a transaction
        try
        {
            string response = await Web3GL.SendContract(method, abi, contract, args,
value, gasLimit, gasPrice);
            Debug.Log(response);
        }
        catch (Exception e)
        {
            Debug.LogException(e, this);
        }
    }
}
#endif
```

WithdrawButtonActivation1:

```
using System.Collections;
using System.Numerics;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class WithdrawButtonActivation1 : MonoBehaviour
{
    [SerializeField] GameObject _withdrawButton;

    string chain = "binance";
    string network = "testnet";
    string contract = "0x5f310227dd9a9e65DaEb9d92282E27DD0eFcA02E";

    int counter = 0;


    float timer = 0f;
    async void Update()
    {
        timer += Time.deltaTime;


        if (timer >= 3f)
        {
            timer = 0f;

            string account = PlayerPrefs.GetString("Account");
            //string account = "0x035dCD3b056BdDbf82273A1b93c7B8cd25614995";
            BigInteger balanceOf = await ERC20.BalanceOf(chain, network, contract,
account);
            print(balanceOf);

            if (balanceOf > 50000 & counter == 0)
            {
                _withdrawButton.SetActive(true);
                counter++;
            }
            else { }
        }
    }
}
```

```
Web3PrivateKeyInGameTransaction:

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Newtonsoft.Json;

public class Web3PrivateKeyInGameTransaction : MonoBehaviour
{


    async public void Awake()
    {
        // private key of account
        string privateKey =
"679e1d009938c24d6e6cb84e93042ba992b69663997692bac7d01f3cc378e8c3";
        // set chain: ethereum, moonbeam, polygon etc
        string chain = "binance";
        // set network mainnet, testnet
        string network = "testnet";
        // smart contract method to call
        string method = "transfer";
        // account of player
        string account = Web3PrivateKey.Address(privateKey);
        // smart contract address:
https://rinkeby.etherscan.io/address/0xc7ad46e0b8a400bb3c915120d284aafba8fc4735
        string contract = "0x5f310227dd9a9e65DaEb9d92282E27DD0eFcA02E";
        // account to send to (PlayerPrefs.GetString("Account"))
        string recipient = "0x035dCD3b056BdDbf82273A1b93c7B8cd25614995";
        // amount of erc20 tokens to send. usually 18 decimals
        string amount = "5000000000000000000";
        // amount of wei to send
        string value = "0";
        // abi to interact with contract
        string abi = "…";
        // optional rpc url
        string rpc = "";

        string[] obj = { recipient, amount };
        string args = JsonConvert.SerializeObject(obj);
        string chainId = await EVM.ChainId(chain, network, rpc);
        string gasPrice = await EVM.GasPrice(chain, network, rpc);
        string data = await EVM.CreateContractData(abi, method, args);
        string gasLimit = "75000";
        string transaction = await EVM.CreateTransaction(chain, network, account,
contract, value, data, gasPrice, gasLimit, rpc);
        string signature = Web3PrivateKey.SignTransaction(privateKey, transaction,
chainId);
        string response = await EVM.BroadcastTransaction(chain, network, account,
contract, value, data, signature, gasPrice, gasLimit, rpc);
        print(response);
        Application.OpenURL("https://testnet.bscscan.com/tx/" + response);
    }
}
```