1 Implementation

This chapter provides an overview of the implementation of the Donuts Games dApp, starting with the smart contract responsible for managing Donut Tokens (DNT), including functions for token creation, transfers, buying, selling, and reward distribution. It then describes the backend integration with the Ethereum blockchain using `ethers.js`, enabling interactions with the smart contract. Lastly, the chapter covers the testing strategy used to verify the correct operation of the smart contract.

1.1 Smart Contract - Token.sol

The core of the dApp is the Donut Token smart contract, which manages token creation, transfers, and NFT minting. Written in Solidity, the contract defines the platform's token economy, with 1,000,000 DNT tokens initially assigned to the contract owner.

1.1.1 Token Creation and Initialization

The contract initializes the name, symbol, and total supply of the token, assigning the full supply to the deployer's address.

```
constructor() {
    balances[msg.sender] = totalSupply;
    owner = msg.sender;
}
```

1.1.2 Token Transfer

The `transfer` function allows users to send tokens to other Ethereum addresses, ensuring the sender has enough tokens before processing the transfer.

```
function transfer(address to, uint256 amount) external {
    require(balances[msg.sender] >= amount, "Not enough tokens");
    balances[msg.sender] -= amount;
    balances[to] += amount;
    emit Transfer(msg.sender, to, amount);
}
```

1.1.3 Buying Tokens

Users can purchase DNT by sending Ether to the contract. The amount of tokens received depends on the exchange rate, and tokens are transferred from the owner's balance to the buyer.

```
function buyTokens() external payable {
    uint256 amountToBuy = msg.value / rate;
    require(amountToBuy > 0, "You need to send some Ether");
    require(balances[owner] >= amountToBuy, "Not enough tokens available");
    balances[owner] -= amountToBuy;
    balances[msg.sender] += amountToBuy;
    emit Transfer(owner, msg.sender, amountToBuy);
    emit Buy(msg.sender, amountToBuy);
}
```

1.1.4 Selling Tokens

The `sellToken` function enables users to sell DNT in exchange for Ether, ensuring the contract has sufficient Ether to complete the transaction.

```
function sellToken(uint256 amountToSell) external {
    require(balances[msg.sender] >= amountToSell, "Not enough tokens");
    uint256 etherAmount = amountToSell * rate;
    require(address(this).balance >= etherAmount, "Not enough Ether in the contract"
    balances[msg.sender] -= amountToSell;
    balances[owner] += amountToSell;
    payable(msg.sender).transfer(etherAmount);
    emit Transfer(msg.sender, owner, amountToSell);
}
```

1.1.5 Rewarding Users

The contract owner can reward users with tokens using the `reward` function, which transfers tokens from the owner's balance.

```
function reward(uint256 amountToReward) external {
    require(balances[owner] >= amountToReward, "Not enough tokens available");
    balances[owner] -= amountToReward;
    balances[msg.sender] += amountToReward;
    emit Transfer(owner, msg.sender, amountToReward);
}
```

1.2 Testing the Smart Contract

Testing ensures that the Donut Token contract works as expected, covering token transfers, purchases, and sales. Tests are written using **Hardhat** and **Chai** to verify the functionality.

1.2.1 Deployment Tests

Tests confirm that the contract is deployed correctly, assigning the total token supply to the owner.

```
it("Should set the right owner", async function () {
    expect(await token.owner()).to.equal(owner.address);
});

it("Should assign the total supply of tokens to the owner", async function () {
    const ownerBalance = await token.balanceOf(owner.address);
    expect(ownerBalance).to.equal(await token.totalSupply());
});
```

1.2.2 Transaction Tests

Tests verify token transfers between users and ensure that transactions fail if there are insufficient funds.

```
it("Should transfer tokens between accounts", async function () {
    await token.transfer(addr1.address, 50);
    const addr1Balance = await token.balanceOf(addr1.address);
    expect(addr1Balance).to.equal(50);
});

it("Should fail if sender doesn't have enough tokens", async function () {
    await expect(token.connect(addr1).transfer(addr2.address, ethers.parseEther("101
        .to.be.revertedWith("Not enough tokens");
});
```

1.2.3 Buy Tokens Test

This test verifies that users can purchase tokens by sending Ether, receiving the correct number of tokens based on the rate.

```
it("Should allow users to buy tokens by sending ether", async function () {
    const amountInEther = ethers.parseEther("1");
    await addr1.sendTransaction({ to: token.target, value: amountInEther });
    await token.connect(addr1).buyTokens({ value: amountInEther });
    const addr1Balance = await token.balanceOf(addr1.address);
    expect(addr1Balance).to.equal(amountInEther / (await token.rate()));
});
```

1.2.4 Withdraw Ether Test

Tests ensure only the contract owner can withdraw Ether and that their balance increases accordingly.

```
it("Should allow the owner to withdraw Ether", async function () {
    const amountInEther = ethers.parseEther("1");
    await addr1.sendTransaction({ to: token.target, value: amountInEther });
    await token.connect(owner).withdraw();
    const finalOwnerBalance = await ethers.provider.getBalance(owner.address);
    expect(finalOwnerBalance).to.be.above(initialOwnerBalance);
});

it("Should not allow non-owner to withdraw Ether", async function () {
    await expect(token.connect(addr1).withdraw()).to.be.revertedWith("Only the owner
});
```

1.2.5 Selling Tokens Test

Tests ensure that users can sell tokens for Ether and verify that transactions fail when users attempt to sell more than their balance.

```
it("Should allow a user to sell tokens and receive Ether", async function () {
    await token.connect(owner).transfer(addr1.address, 1000);
    const amountToSell = 10;
    await token.connect(addr1).sellToken(amountToSell);
    const addr1Balance = await token.balanceOf(addr1.address);
    expect(addr1Balance).to.equal(1000 - amountToSell);
});

it("Should fail if user tries to sell more tokens than they have", async function ()
    await expect(token.connect(addr1).sellToken(2000)).to.be.revertedWith("Not enoug
});
```

1.3 Deploying the Smart Contract

The deployment of the Donut Token contract is automated using a script written in **ethers.js**. The script retrieves the deployer's account, creates an instance of the contract, and deploys it to the Ethereum blockchain.

1.3.1 Deploy Script

The script defines an asynchronous `main` function to handle the contract deployment process.

```
async function main() {
    const [deployer] = await ethers.getSigners();
    const Token = await ethers.getContractFactory("Token");
    const token = await Token.deploy();
    console.log("Token deployed to:", token.target);
}
```

The script concludes by handling potential errors and ensuring the deployment completes successfully.

```
main()
    .then(() => process.exit(0))
    .catch(error => {
        console.error(error);
        process.exit(1);
    });
```

1.4 Backend.js Module

The `backend.js` module facilitates interactions between the dApp's backend and the Ethereum smart contract. Using **ethers.js**, it manages token transactions, rewards, and NFTs.

1.4.1 Configuring the Provider and Wallet

The module initializes a connection to the Ethereum network using a provider and configures the wallet using private keys.

```
const provider = ethers.getDefaultProvider('http://127.0.0.1:8545');
const privateKey = process.env.PRIVATE_KEY_1;
const wallet = new ethers.Wallet(privateKey, provider);
```

1.4.2 Contract Connection

An instance of the contract is created using its ABI and address, allowing the backend to interact with the deployed contract.

```
const contractAddress = process.env.CONTRACT_ADDRESS;
const contractABI = [...];
const contract = new ethers.Contract(contractAddress, contractABI, wallet);
```

1.4.3 Buying Tokens

The `buyToken` function allows users to buy tokens by sending Ether to the contract.

```
async function buyToken(amount) {
    const valueToSend = ethers.parseEther(amount);
    const tx = await contract.buyTokens({ value: valueToSend });
    const receipt = await tx.wait();
    await updateBalance();
}
```

1.4.4 Selling Tokens

The `sell` function enables users to sell tokens back to the contract and receive Ether.

```
async function sell(amountToSell) {
    const tx = await contract.sellToken(amountToSell);
    const receipt = await tx.wait();
    await updateBalance();
}
```

1.4.5 Rewarding Users

The `reward` function allows the backend to distribute rewards to users, transferring tokens from the owner's balance.

```
async function reward(amountToReward) {
    const tx = await contract.reward(amountToReward);
    const receipt = await tx.wait();
    await updateBalance();
}
```