FAIRYPROOF

Macaron DEX

AUDIT REPORT

Version 1.0.0

Serial No. 2024041300012019

Presented by Fairyproof

April 13, 2024

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Macaron DEX project.

Audit Start Time:

April 10, 2024

Audit End Time:

April 13, 2024

Audited Code's Github Repository:

https://github.com/Macaromswap/contract-swap/tree/main

Audited Code's Github Commit Number When Audit Started:

be82f3a1b85cb7b9d72401b45a48b7359b205734

Audited Code's Github Commit Number When Audit Ended:

301f2e551f10924bd9837e0e657c2c2067488895

Audited Source Files:

The source files audited include all the files as follows:

```
contracts
├── ERC20.sol
├── Factory.sol
├── Multicall.sol
├── Pair.sol
├── Router.sol
├── interfaces
|   ├── IERC20.sol
|   ├── IFactory.sol
|   ├── IPair.sol
|   ├── IRouter01.sol
|   ├── IRouter02.sol
|   ├── ISwapCallee.sol
|   ├── ISwapMining.sol
|   ├── ITradingPairRestrictions.sol
|   └── IWBTC.sol
└── libraries
    ├── EnumerableSet.sol
    ├── Math.sol
    ├── PairOperations.sol
    ├── SafeMath.sol
    ├── TransferHelper.sol
    └── UQ112x112.sol

2 directories, 20 files
```

The goal of this audit is to review Macaron's solidity implementation for its DEX function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Macaron team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code Review, Including:

- Project Diagnosis

Understanding the size, scope and functionality of your project's source code based on the specifications, sources, and instructions provided to Fairyproof.

- Manual Code Review

Reading your source code line-by-line to identify potential vulnerabilities.

- Specification Comparison

Determining whether your project's code successfully and efficiently accomplishes or executes its functions according to the specifications, sources, and instructions provided to Fairyproof.

2. Testing and Automated Analysis, Including:

- Test Coverage Analysis

Determining whether the test cases cover your code and how much of your code is exercised or executed when test cases are run.

- Symbolic Execution

Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.

3. Best Practices Review

Reviewing the source code to improve maintainability, security, and control based on the latest established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following source(s) of truth about how the token issuance function should work:

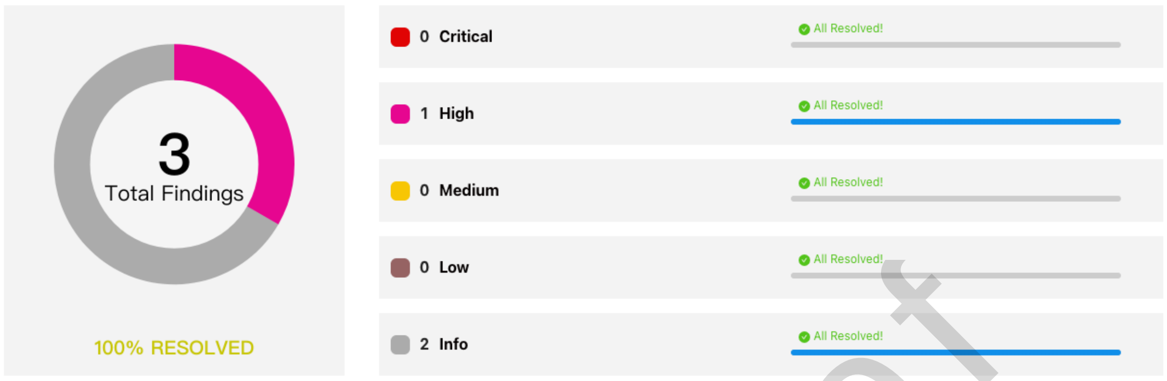
Website:https://macaron.xyz/

Source Code: https://github.com/Macaromswap/contract-swap/tree/main

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Macaron team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2024041300012019	Fairyproof Security Team	Apr 10, 2024 - Apr 13, 2024	Passed



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, one issue of high-severity and two issues of info-severity were uncovered. The Macaron team fixed all the issues.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to Macaron

Macaron is a decentralized exchange similar to UniswapV2.

The above description is quoted from relevant documents of Macaron.

04. Major functions of audited code

The audited code mainly implements a UniswapV2 alike DEX and has the following features:

- The transaction fees charged by the DEX are not the fixed rate of 0.3% but are set by the team. The rates for different trading pairs can be different.
- The transaction fees that the DEX can take are not the fixed rate of 1/6 but are set by the team. The rates for different trading pairs can be different.

05. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Access Control
- Admin Rights
- Arithmetic Precision
- Code Improvement
- Contract Upgrade/Migration
- Delete Trap
- Design Vulnerability
- DoS Attack
- EOA Call Trap
- Fake Deposit
- Function Visibility
- Gas Consumption
- Implementation Vulnerability
- Inappropriate Callback Function
- Injection Attack
- Integer Overflow/Underflow
- IsContract Trap
- Miner's Advantage
- Misc
- Price Manipulation
- Proxy selector clashing
- Pseudo Random Number
- Re-entrancy Attack

- Replay Attack
- Rollback Attack
- Shadow Variable
- Slot Conflict
- Token Issuance
- Tx.origin Authentication
- Uninitialized Storage Pointer

06. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

07. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Function Implementation

We checked whether or not the functions were correctly implemented.
We found one issue, for more details please refer to [FP-1] in "09. Issue description".

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by owner or administrator
We didn't find issues or risks in these functions or areas at the time of writing.

- Token Issuance & Transfer

We examined token issuance and transfers for situations that could harm the interests of holders.
We didn't find issues or risks in these functions or areas at the time of writing.

- State Update

We checked some key state variables which should only be set at initialization.
We didn't find issues or risks in these functions or areas at the time of writing.

- Asset Security

We checked whether or not all the functions that transfer assets were safely handled.
We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We checked the code for optimization and robustness.
We found some issues, for more details please refer to [FP-2,FP-3] in "09. Issue description".

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Incorrect Validation of K Value	Implementation Vulnerability	High	✓ Fixed
FP-2	Incomplete Log Events	Code Improvement	Info	✓ Fixed
FP-3	Unused Code	Code Improvement	Info	✓ Fixed

09. Issue descriptions

[FP-1] Incorrect Validation of K Value

Implementation Vulnerability High ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In the `swap` function of `Pair.sol`, the validation of `K` should be `10000**2` instead of `1000**2` otherwise the assets can be purchased at 1% of their correct prices.

Recommendation:

Consider changing to `10000**2`

Update/Status:

The Macaron team has fixed the issue.

[FP-2] Incomplete Log Events

Code Improvement Info ✓ Fixed

Issue/Risk: Code Improvement

Description:

In the `mint` function of `Pair.sol`, the `MintLiquidity` event will eventually be triggered. The event's first parameter was `msg.sender`, however a more often used parameter here is `router`. It didn't record who provided the liquidity. Consider adding a `to` parameter to make this event work the same way as the `burn` function's `BurnLiquidity` event.

Recommendation:

Consider adding a `to` parameter to record who provides the liquidity.

Update/Status:

The Macaron team has fixed the issue.

[FP-3] Unused Code

Code Improvement Info ✓ Fixed

Issue/Risk: Code Improvement

Description:

In `Router.sol`, both `_swap` and `_swapSupportingFeeOnTransferTokens` had code to check `swapMining`'s threshold value. In order to make the `router` contract modularized and independent, consider moving the code to the `Swap Mining` contract that will be planned in the future. This would simplify the `router` contract as well.

Recommendation:

Consider removing the code to simplify the `router` contract, making it modularized and independent.

Update/Status:

The Macaron team has fixed the issue.

10. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Consider managing the owner's access control with great care and transfering it to a multi-sig wallet or DAO when necessary.

11. Appendices

11.1 Unit Test

1. Swap.t.js

```
const {
    loadFixture,
    anyValue,
} = require("@nomicfoundation/hardhat-toolbox/network-helpers");
const { expect, assert } = require("chai");
const { ethers } = require("hardhat");


  describe("Filx Token Unit Test", function () {

    async function deploySwapFixture() {
        const [owner, alice,bob,...users] = await ethers.getSigners();
        let feeToSetter = owner.address;
        let fee_to =  users[1].address;
        // deploy WETH and token
        const WETH9 = await ethers.getContractFactory("WETH9");
        const weth = await WETH9.deploy();
        const StandardToken = await ethers.getContractFactory("StandardToken");
        const usdt = await StandardToken.deploy("USDT","USDT",
ethers.parseEther("10000000000000"));
        const token_a = await StandardToken.deploy("TokenA","TokenA",
ethers.parseEther("1234567000"));
        const token_b = await StandardToken.deploy("TokenB","TokenB",
ethers.parseEther("2209870000"));

        // deploy factory
        const Factory = await ethers.getContractFactory("Factory");
        const factory = await Factory.deploy(feeToSetter);
        // deploy router
        const Router = await ethers.getContractFactory("Router");
        const router = await Router.deploy(factory.target,weth.target);

        // deploy callee
        const MockSwapCallee = await ethers.getContractFactory("MockSwapCallee");
        const callee = await MockSwapCallee.deploy(router.target);

        // deploy swapminit
        const MockSwapMining = await ethers.getContractFactory("MockSwapMining");
        const swap_minig = await MockSwapMining.deploy();
        // set up
        await factory.setFeeTo(fee_to);
        await factory.setFeeRateNumerator(30);
        await factory.setDefaultFeeToRate(5);

        await router.setSwapMining(swap_minig.target);

        // add pair
        await usdt.approve(router.target,ethers.MaxUint256);
        await token_a.approve(router.target,ethers.MaxUint256);
        await token_b.approve(router.target,ethers.MaxUint256);

        // deploy mock
        const MockPairOperation = await
ethers.getContractFactory("MockPairOperation");
        const operation = await MockPairOperation.deploy();

        await router.addLiquidityETH(
            usdt.target,
            ethers.parseEther("34000"),
            1,
            1,
            owner.address,
            9876543210,
            {
                value:ethers.parseEther("10")
            }
        );

        await router.addLiquidity(
            usdt.target,
            token_a.target,
            ethers.parseEther("1000000"),
            ethers.parseEther("1500000"),
            1,
            1,
            owner.address,
            9876543210
        );

        return {

 owner,alice,bob,feeToSetter,fee_to,users,usdt,token_a,token_b,factory,
            router,swap_minig,operation,weth,callee
        }
    }

    describe("Operation unit test", function() {
        it("sortTokens unit test", async () => {
            let {operation} = await loadFixture(deploySwapFixture);
            let address_f = ethers.getAddress("0x" + "0".repeat(39) + "F");
            let address_d = ethers.getAddress("0x" + "0".repeat(39) + "D");
            let r = await operation.sortTokens(address_f,address_d);
            expect(r[0]).eq(address_d);
            expect(r[1]).eq(address_f);
            r = await operation.sortTokens(address_d,address_f);
            expect(r[0]).eq(address_d);
            expect(r[1]).eq(address_f);
        });

        it("pairFor unit test", async () => {
            let {operation,usdt,token_a,factory} = await
loadFixture(deploySwapFixture);
            let pair = await factory.getPair(usdt.target,token_a.target);
            assert(pair !== ethers.ZeroAddress,"zero address");
            expect(await
operation.pairFor(factory.target,usdt.target,token_a.target)).eq(pair);
            expect(await
operation.pairFor(factory.target,token_a.target,usdt.target)).eq(pair);
            expect(await factory.getPair(token_a.target,usdt.target)).eq(pair);
        });

        it("getReserves and quote test", async () => {
            let {operation,usdt,token_a,factory} = await
loadFixture(deploySwapFixture);
            let reserve_usdt = ethers.parseEther("1000000");
            let reserve_token_a = ethers.parseEther("1500000");

            let result = await
operation.getReserves(factory.target,usdt.target,token_a.target);
            expect(result[0]).eq(reserve_usdt);
            expect(result[1]).eq(reserve_token_a);
            result = await
operation.getReserves(factory.target,token_a.target,usdt.target);
            expect(result[0]).eq(reserve_token_a);
            expect(result[1]).eq(reserve_usdt);

            for(let times=2;times<=6;) {
                let n =  ethers.getBigInt(times);
                let amountIn = reserve_usdt / n;
                let amountOut = reserve_token_a / n;
                let r = await
operation.quote(amountIn,reserve_usdt,reserve_token_a);
                assert(amountOut - r <= 1,"unmatched")
                times += 1;
            }
        });

        it("getAmountOut and getAmountIn unit test", async () => {
            let {operation,usdt,token_a,factory} = await
loadFixture(deploySwapFixture);
            let pair = await factory.getPair(usdt.target,token_a.target);
            let reserve_usdt = ethers.parseEther("1000000");
            let reserve_token_a = ethers.parseEther("1500000");
            let usdt_amountIn = ethers.parseEther("1.0");
            expect(await factory.getPairFees(pair)).eq(30);
            let amountInWithFee = usdt_amountIn * ethers.getBigInt(997);
            let numerator = amountInWithFee * reserve_token_a;
            let denominator = reserve_usdt * ethers.getBigInt(1000) +
amountInWithFee;
            let amountOut = numerator / denominator;
            expect(await
operation.getAmountOut(factory.target,usdt_amountIn,reserve_usdt,
                reserve_token_a,usdt.target,token_a.target
            )).eq(amountOut);

            expect(await operation.getAmountIn(
                factory.target,
                amountOut,
                reserve_usdt,
                reserve_token_a,
                usdt.target,
                token_a.target
            )).eq(usdt_amountIn);

            expect(await factory.isSupportPair(pair)).be.false;
            await factory.addPairs([pair]);
            expect(await factory.isSupportPair(pair)).be.true;
            expect(await factory.getPairFees(pair)).eq(0);
            await factory.setPairFees([pair],[25]);
            expect(await factory.getPairFees(pair)).eq(25);

            amountInWithFee = usdt_amountIn * ethers.getBigInt(9975);
            numerator = amountInWithFee * reserve_token_a;
            denominator = reserve_usdt * ethers.getBigInt(10000) +
amountInWithFee;
            amountOut = numerator / denominator;

            expect(await
operation.getAmountOut(factory.target,usdt_amountIn,reserve_usdt,
                reserve_token_a,usdt.target,token_a.target
            )).eq(amountOut);

            expect(await operation.getAmountIn(
                factory.target,
                amountOut,
                reserve_usdt,
                reserve_token_a,
                usdt.target,
                token_a.target
            )).eq(usdt_amountIn);
        });

        it("getAmountsIn and getAmountsOut test" ,async () => {
            let {operation,usdt,token_a,weth,factory} = await
loadFixture(deploySwapFixture);
            let eth_amountIn = ethers.parseEther("0.2");
            let usdt_amountOut = await operation.getAmountOut(
                factory.target,
                eth_amountIn,
                ethers.parseEther("10"),
                ethers.parseEther("34000"),
                weth.target,
                usdt.target
            );
            let token_a_amountOut = await operation.getAmountOut(
                factory.target,
                usdt_amountOut,
                ethers.parseEther("1000000"),
                ethers.parseEther("1500000"),
                usdt.target,
                token_a.target
            );
            let r = await operation.getAmountsOut(
                factory.target,
                eth_amountIn,
                [weth.target,usdt.target,token_a.target]
            );
            expect(r[0]).eq(eth_amountIn);
            expect(r[1]).eq(usdt_amountOut);
            expect(r[2]).eq(token_a_amountOut);

            r = await operation.getAmountsIn(
                factory.target,
                token_a_amountOut,
                [weth.target,usdt.target,token_a.target]
            );
            expect(r[0]).eq(eth_amountIn);
            expect(r[1]).eq(usdt_amountOut);
            expect(r[2]).eq(token_a_amountOut);
        });
    });


    describe("factory test", function () {
        it("Create pair twice should be failed", async () => {
            let {usdt,weth,factory} = await loadFixture(deploySwapFixture);
            await
expect(factory.createPair(weth.target,usdt.target)).to.be.revertedWith("PAIR_EXIS
TS");
            await
expect(factory.createPair(usdt.target,weth.target)).to.be.revertedWith("PAIR_EXIS
TS");
```

```
        });

        it("Create pair should emit event", async () => {
            let {operation,usdt,weth,token_a,factory} = await
loadFixture(deploySwapFixture);
            expect(await factory.allPairsLength()).eq(2);
            let pair = await
operation.pairFor(factory.target,weth.target,token_a.target);
            let [token0,token1] = await
operation.sortTokens(weth.target,token_a.target);
            await
expect(factory.createPair(weth.target,token_a.target)).to.be.emit(
                factory,"PairCreated"
            ).withArgs(token0,token1,pair,3);
            expect(await factory.allPairsLength()).eq(3);
            expect(await factory.getPair(weth.target,token_a.target)).eq(pair);
            expect(await factory.getPair(token_a.target,weth.target)).eq(pair);
        });

        it("set/get unit test", async () => {
            let {alice,bob,users,factory} = await loadFixture(deploySwapFixture);
            await factory.setFeeRateNumerator(20);
            expect(await factory.feeRateNumerator()).eq(20);

            let rate = 7;
            await factory.setDefaultFeeToRate(rate + 1);
            expect(await factory.feeToRate()).eq(rate);

            await factory.setFeeTo(alice.address);
            expect(await factory.feeTo()).eq(alice.address);

            await factory.setFeeToSetter(bob.address);
            expect(await factory.feeToSetter()).eq(bob.address);

            await expect(factory.addPairs(users.map(v =>
v.address))).to.revertedWith(
                "FORBIDDEN"
            );

            expect(await factory.getSupportListLength()).eq(0);
            await factory.connect(bob).addPairs(users.map(v => v.address));
            await factory.connect(bob).setPairFeeToRate(users.map(v =>
v.address),users.map(() => 3));
            await factory.connect(bob).setPairFees(users.map(v =>
v.address),users.map(() => 15));

            expect(await factory.getSupportListLength()).eq(users.length);

            await factory.connect(bob).delPairs([users[3].address]);
            expect(await factory.getSupportListLength()).eq(users.length - 1);

            let all_pairs = await factory.getAllSupportPairs();
            for(let i=0;i<all_pairs.length;i++) {
                expect(await factory.getSupportPair(i)).eq(all_pairs[i]);
            }
            for(let i=0;i<users.length;i++) {
                let flag = await factory.isSupportPair(users[i].address);
                let pair_rate = await factory.getPairRate(users[i].address);
                let pair_fee = await factory.getPairFees(users[i].address);
                if(i == 3) {
                    assert(!flag,"removed pair");
                    expect(pair_rate).eq(rate);
                    expect(pair_fee).eq(20);
                } else {
                    assert(flag,"inner error");
                    expect(pair_rate).eq(2);
                    expect(pair_fee).eq(15);
                }
            }
        });
    });

    describe("Router unit test", function() {
        it("set/get test", async () => {
            const {router,swap_minig,alice,bob} = await
loadFixture(deploySwapFixture);
            expect(await router.swapMining()).eq(swap_minig.target);
            await router.setSwapMining(alice.address);
            expect(await router.swapMining()).eq(alice.address);
        });

        it("add/remove Liquid test", async () => {
            const {router,owner,factory,swap_minig,operation,
                alice,bob,usdt,token_a} = await loadFixture(deploySwapFixture);
            let pair = await factory.getPair(usdt.target,token_a.target);
            let pool = await ethers.getContractAt("Pair",pair);
            let lp_supply = await pool.totalSupply();
            let usdt_reserve = ethers.parseEther("1000000");
            let token_reserve = ethers.parseEther("1500000");
            let times = ethers.getBigInt(100);
            await expect(router.addLiquidity(
                usdt.target,
                token_a.target,
                usdt_reserve / times,
                token_reserve / times,
                1,
                1,
                owner.address,
                9876543210
            )).to.emit(
                pool,"MintLiquidity"
            ).withArgs(router.target,lp_supply / times,owner.address);

            let lp_before = lp_supply;
            let [token0,] = await
operation.sortTokens(usdt.target,token_a.target);
            expect(await pool.token0()).eq(token0);
            lp_supply = await pool.totalSupply();
            await pool.approve(router.target,ethers.MaxUint256);
            let [r0,r1] = await router.removeLiquidity.staticCall(
                usdt.target,
                token_a.target,
                lp_supply - lp_before,
                1,
                1,
                alice,
                9876543210
            );
            assert(usdt_reserve / times - r0 <= 1, "cal error");
            assert(token_reserve / times - r1 <= 1, "cal error");
            await expect(router.removeLiquidity(
                usdt.target,
                token_a.target,
                lp_supply - lp_before,
                1,
                1,
                alice,
                9876543210
            )).to.emit(
                pool,"Burn"
            ).withArgs(
                router.target,
                token0 === usdt ? r0: r1,
                token0 === usdt ? r1 : r0,
                alice.address
            );
        });
    });

    describe("Swap and K value test", function(){
        it("K value test successfully", async () => {
            const amount0OutFixed = ethers.getBigInt(1000000);
            const amount1OutFixed = ethers.getBigInt(1200000);
            const {callee,token_a,usdt,factory} = await
loadFixture(deploySwapFixture);
            let pair = await factory.getPair(token_a.target,usdt.target);
            let pool = await ethers.getContractAt("Pair",pair);
            await token_a.transfer(callee.target,ethers.parseEther("1.0"));
            await usdt.transfer(callee.target,ethers.parseEther("1.0"));

            let pay = amount0OutFixed * ethers.getBigInt(1000) /
ethers.getBigInt(997)  + ethers.getBigInt(1);

            await expect(callee.KTest(pair,false)).to.emit(
                pool,"Swap"
            ).withArgs(callee.target,pay,0,amount0OutFixed,0,callee.target);
        });

        it("K value test failed", async () => {
            const {callee,token_a,usdt,factory} = await
loadFixture(deploySwapFixture);
            let pair = await factory.getPair(token_a.target,usdt.target);
            await token_a.transfer(callee.target,ethers.parseEther("1.0"));
            await usdt.transfer(callee.target,ethers.parseEther("1.0"));
            await expect(callee.KTest(pair,true)).to.be.revertedWith("K");
        });
        it("Flash load mode zero test", async () => {
            const {callee,token_a,usdt,factory} = await
loadFixture(deploySwapFixture);
            const amount0OutFixed = ethers.getBigInt(1000000);
            let pair = await factory.getPair(token_a.target,usdt.target);
            let pool = await ethers.getContractAt("Pair",pair);
            await token_a.transfer(callee.target,ethers.parseEther("1.0"));
            await usdt.transfer(callee.target,ethers.parseEther("1.0"));
            let pay = amount0OutFixed * ethers.getBigInt(1000) /
ethers.getBigInt(997)  + ethers.getBigInt(1);
            await expect(callee.loan(pair,0)).to.emit(
                pool,"Swap"
            ).withArgs(callee.target,pay,0,amount0OutFixed,0,callee.target);
        });

        it("Flash load mode one test", async () => {
            const {callee,token_a,usdt,router,factory,operation} = await
loadFixture(deploySwapFixture);
            const amount0OutFixed = ethers.getBigInt(1000000);
            let pair = await factory.getPair(token_a.target,usdt.target);
            let pool = await ethers.getContractAt("Pair",pair);
            await token_a.transfer(callee.target,ethers.parseEther("1.0"));
            await usdt.transfer(callee.target,ethers.parseEther("1.0"));
            let [token0,token1] = await
operation.sortTokens(usdt.target,token_a.target);
            let [reserveIn,reserveOut] = await operation.getReserves(
                factory.target,token1,token0
            );

            let amountIn = await operation.getAmountIn(
                factory.target,
                amount0OutFixed,
                reserveIn,
                reserveOut,
                token0,
                token1
            );

            await expect(callee.loan(pair,1)).to.emit(
                pool,"Swap"
            ).withArgs(callee.target,0,amountIn,amount0OutFixed,0,callee.target);
        });


        it("Flash load mode two test", async () => {
            const {callee,token_a,usdt,router,factory,operation} = await
loadFixture(deploySwapFixture);
            const amount0OutFixed = ethers.getBigInt(1000000);
            const amount1OutFixed = ethers.getBigInt(1200000);
            let pair = await factory.getPair(token_a.target,usdt.target);
            let pool = await ethers.getContractAt("Pair",pair);
            await token_a.transfer(callee.target,ethers.parseEther("1.0"));
            await usdt.transfer(callee.target,ethers.parseEther("1.0"));
            let pay0 = amount0OutFixed * ethers.getBigInt(1000) /
ethers.getBigInt(997)  + ethers.getBigInt(1);
```

```
            let pay1 = amount1OutFixed * ethers.getBigInt(1000) /
ethers.getBigInt(997)  + ethers.getBigInt(1);
            await expect(callee.loan(pair,2)).to.emit(
                pool,"Swap"

).withArgs(callee.target,pay0,pay1,amount0OutFixed,amount1OutFixed,callee.target)
;
        });
    });

    describe("Swap Mining Unit test", function() {
        it("swap can emit swap mining", async () => {
            const {owner,token_a,usdt,router,factory,swap_minig,operation} =
await loadFixture(deploySwapFixture);
            let amountIn = ethers.parseEther("1.0");
            let path = [token_a.target,usdt.target];
            let [,amountOut] = await
operation.getAmountsOut(factory.target,amountIn,path);
            await expect(router.swapExactTokensForTokens(
                amountIn,
                1,
                path,
                owner.address,
                9876543210
            )).emit(
                swap_minig,"SwapMine"
            ).withArgs(owner.address,path[0],path[1],amountOut);
        });

    });
});
```

2. UnitTestOutput

```
 Macaron Dex Unit Test
    Operation unit test
      ✓ sortTokens unit test (1128ms)
      ✓ pairFor unit test
      ✓ getReserves and quote test
      ✓ getAmountOut and getAmountIn unit test (42ms)
      ✓ getAmountsIn and getAmountsOut test (42ms)
    factory test
      ✓ Create pair twice should be failed
      ✓ Create pair should emit event
      ✓ set/get unit test (194ms)
    Router unit test
      ✓ set/get test
      ✓ add/remove Liquid test (189ms)
    Swap and K value test
      ✓ K value test successfully
      ✓ K value test failed (38ms)
      ✓ Flash load mode zero test
      ✓ Flash load mode one test (53ms)
      ✓ Flash load mode two test (39ms)
    Swap Mining Unit test
      ✓ swap can emit swap mining


  16 passing (2s)
```

11.2 External Functions Check Points

- N/A

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech