Gravity Finance Code Brief

This code was started by forking Sushi Swap

* audit requested

Forked Code	**Custom Code**
*UniswapV2ERC20.sol**	*EarningsManager.sol**
*UniswapV2Pair.sol**	*FeeManager.sol**
*UniswapV2Factory.sol**	*Holding.sol**
*UniswapV2Router02.sol**	*PathOracle.sol**
Context.sol	*PriceOracle.sol**
Ownable.sol	*CompounderFactory.sol**
Math.sol	*FarmFactory.sol**
SafeMath.sol	*FarmTimeLock.sol**
TransferHelper.sol	*FarmV2.sol**
*UniswapV2Library.sol**	*Flat_GFIFarm.sol*
UQ112x112.sol	*Governance.sol**
	GravityIDO.sol
	*GravityToken.sol**
	*Incinerator.sol**
	IOUToken.sol
	Locking.sol
	*Share.sol**
	*VestingV2.sol**
	WalletTimeLock.sol

This document will briefly go over every contract in the above table, and explain its function at a high level. Then it will go through the forked code with modifications and explain every modification.

Forked Code

UniswapV2ERC20.sol
Parent contract for UniswapV2Pair, used to provide ERC20 functionality for pair contract.

UniswapV2Pair.sol
Inherits from UniswapV2ERC20, implements all swap logic

UniswapV2Factory.sol
Creates swap pairs through createPair(). Serves as central storage location for variables shared by UniswapV2Pair(s), PriceOracle, PathOracle, EarningsManager, FeeManager.

UniswapV2Router02.sol
Used by front end users to route requests to appropriate pair contract.

UniswapV2Library.sol
Standard UniswapV2 library used to provide helper functions to facilitate swaps and liquidity addition/removal.

Context.sol, Ownable.sol, Math.sol, SafeMath.sol, TransferHelper.sol, UQ112x112.sol
Contracts required by above UniswapV2 contracts. These are all un edited and serve their purpose intended by the Uniswap team.

Custom Code

EarningsManager.sol
Swap pairs with GFI as one of the assets will earn their share of platform earnings distributed in wETH. This contract is in charge of taking the wETH earnings for a pair, converting it into the pairs swap assets, adding liquidity to the pair, then destroying its LP tokens. This effectively raises the value of all LP tokens.

FeeManager.sol
When users make swaps, a 0.05% governance fee is sent to this contract. This contract is responsible for taking those governance fees and swapping them into wETH and wBTC in a 50/50 split.

Holding.sol
Deployed by the pair contract when the pair is created. Responsible for temporarily holding the wETH earnings for GFI swap pairs. Currently all swap pairs deploy this contract, though it is really only needed for pairs with GFI as one of their assets.

PathOracle.sol
Stores a token mapping called ‘pathMap’. When new pairs are created, the pathMap variable is updated in order to find a way to go from the swap pairs native assets to wETH and wBTC. If no path exists(or the path is too complex) then pathMap is not updated. pathMap can be altered by the owner by calling ‘alterPath’.

PriceOracle.sol
On chain pricing oracle that uses UniswapV2 pair contracts TWAP value. Set up to have a most up to date price, and a currently maturing price. Currently maturing price is used to mitigate price down time. When the contract has a TWAP saved with a timestamp from 5 to 10 min ago, then a valid price can be determined by using the pair contracts current TWAP.

CompounderFactory.sol
Factory contract to create farm compounders for all GFI owned farms. Uses Openzeppelin Clones (with Shares.sol as the implementation) to create Shares for every farm a compounder is needed. These shares keep track of the ownership of the farm deposit tokens that the compounder owns. Only the owner can create new compounders, which is done by minting new share contracts.

FarmFactory.sol
Factory contract to create FarmV2.sol farms. Set up so that owner can specify every detail of a new farm, but allow 3rd parties to maintain control of the farm reward token up until it it sent to the newly created farm. Will also serve as a farm address repo for Gravity Finance front end.

FarmTimeLock.sol
Legacy contract used to add a week timelock to owner functions on FarmV1 contracts like Flat_GFIFarm.sol.

FarmV2.sol
Based off of Flat_GFIFarm.sol, but owner function calls have been removed. Contract was also upgraded to Solidity V8 to remove the need to use SafeMath. Also added in harvestFee logic so that Gravity Finance can take up to 5% of the harvested tokens. This logic also includes a whitelist that will be used to exempt certain Gravity Finance contracts from this harvest fee.

Flat_GFIFarm.sol
Legacy farm contract, no new farms will be deployed using this contract. Is only in the repo so that FarmTimeLock could be tested with an existing V1 farm.

Governance.sol
Upgradeable contract in charge of managing and distributing platform earnings to GFI holders. It does this by tracking token transfers, and updating a fee ledger. This contract is upgradeable because we will be adding new investment strategies in it(to grow platform earnings), Tier logic(GFI holders will get exclusive perks for holding GFI), and will potentially change the fee logic in the future if a more elegant solution becomes available. We plan to make the proxy admin of this contract timelocked in the near future, but eventually pass it to a Gravity Finance DAO.

GravityIDO.sol
Legacy Gravity Finance IDO sale contract, already deployed and successful.

GravityToken.sol
Token contract is already deployed, but Governance contract heavily relies upon it. Eventually we want to pass owner role to a DAO, because we could pause all token transfers by setting ‘applyGovernanceForwarding’ to true, and setting GOVERNANCE_ADDRESS to address(0).

Incinerator.sol
Contract accepts wETH, swaps it into GFI, and burns all of its GFI. For this deployment only the FarmV2 contracts will send their wETH earnings to the Incinerator, but we plan to use this contract in the future to burn more GFI.

IOUToken.sol
Legacy token contract used by the GravityIDO contract to keep track of how much wETH a user sent the IDO.

Locking.sol
Contract is already deployed with the team's vested GFI. GFI claiming, and Fee claiming have been tested to work with all the contracts in this repo, but it is a known issue that the withdrawAll() function will fail because a transferFrom call is used instead of a transfer.

Share.sol
Used by the CompounderFactory to create new compounders for farms. Just an ERC20 token that keeps track of users share of the compounders deposited tokens.

VestingV2.sol
Contract is already deployed with investors' vested GFI. Very similar to Locking contract except instead of all tokens being unlocked after one year, 10% of users tokens will unlocked every 30 days for ~10 months.

WalletTimeLock.sol
Contract is a work in progress, but uses very similar logic to FarmTimeLock contract, except it's goal is to add a week timelock to token transfers from project wallets.

Forked Code Modifications

UniswapV2ERC20.sol
contracts/uniswapV2/UniswapV2ERC20.sol

Lines 10-11

```
string public constant name = 'Gravity Finance LP Token';
string public constant symbol = 'GLP';
```

Updated name and symbol to correspond with our project.

Lines 96-103

```
/**
    * @dev called by the Earnings manager after wETH earnings are
converted into pool assets, and deposited into the pool
    * Note anyone can call this function and burn their LP tokens, though
I don't know why they would
    **/
    function destroy(uint value) external returns(bool){
        _burn(msg.sender, value);
        return true;
    }
```

destroy function was added to the uniswap ERC20 contract, so that the earnings manager had a way to burn its LP tokens it received from converting wETH earnings to pair assets and depositing liquidity.

UniswapV2Pair.sol
contracts/uniswapV2/UniswapV2Pair.sol

Lines 43-45

```
    Holding holder;
    address public HOLDING_ADDRESS;
    IUniswapV2Factory Factory;
```

Added new variables.

Lines 55-58

```
    modifier onlyEarningsManager() {
        require(msg.sender == Factory.earningsManager(), "Gravity Finance:
FORBBIDDEN");
        _;
    }
```

Added new modifier to check if caller is the EarningsManager contract.

Line 84

```
event HandleEarnings(uint amount);
```

New event to track earnings being added back into liquidity.

Line 89

```
Factory = IUniswapV2Factory(msg.sender);
```

Set the new Factory variable in the constructor.

Lines 98-99

```
        holder = new Holding();
        HOLDING_ADDRESS = address(holder);
```

In initialize function create a Holding contract to hold wETH earnings temporary. This function could be wrapped in an if statement so that a Holding contract isn't deployed if token0 nor token1 is GFI.

Lines 102-110

```
    /**
    * @dev called by the earningsManager in order to have the pair claim
its wETH earnings and send it to its holding contract. Then pair tells
holding contract to approve earnings manager to spend its wETH
    **/
    function handleEarnings() external onlyEarningsManager returns(uint
amount){
        require(token0 == Factory.gfi() || token1 == Factory.gfi(), "Swap
```

```
contract must have GFI as one of it's assets to claim earnings");
//Require statement not really needed
        amount =
iGovernance(Factory.governor()).delegateFee(HOLDING_ADDRESS); //Calculates
WETH fees earned by GFI in contract
        holder.approveEM(Factory.weth(), Factory.earningsManager(),
amount);
        emit HandleEarnings(amount);
    }
```

The handleEarnings function has 5 major parts.

1. Check to see if caller is the EarningsManager, revert if they are not.
2. Make sure one of this pairs tokens is GFI. This check is a bit redundant.
3. Delegate the pairs wETH earnings to its Holding contract
4. Have the Holding contract approve the EarningsManager to spend its wETH
5. Emit an event

This function will be called by the EarningsManager contract when it is processing the earnings for that pair.

Lines 130, 183

```
require(!Factory.paused(), "Swap contract is paused, users can only remove
liquidity");
```

Owner has ability to pause the swap contracts, which makes it so that users can only remove liquidity, they will not be able to swap, or deposit liquidity. This was added for user protection in case of smart contract exploit.

Lines 195-204

```
        if (amount0Out > 0){
            uint old_amount0Out = amount0Out;
            amount0Out = amount0Out.mul(9995) / (10000); //Remove 0.05%
gov fee
             _safeTransfer(_token0, Factory.feeManager(),
old_amount0Out.sub(amount0Out)); // optimistically transfer tokens
        }
        if (amount1Out > 0){
            uint old_amount1Out = amount1Out;
            amount1Out = amount1Out.mul(9995) / (10000); //Remove 0.05%
gov fee
             _safeTransfer(_token1, Factory.feeManager(),
old_amount1Out.sub(amount1Out)); // optimistically transfer tokens
        }
```

Used to send the 0.05% governance fee to the FeeManager contract.

Lines 218-220

```
        uint balance0Adjusted =
balance0.mul(10000).sub(amount0In.mul(25));
        uint balance1Adjusted =
balance1.mul(10000).sub(amount1In.mul(25));
        require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(10000**2), 'Gravity Finance: K');
```

Updated so that swap pair only checks new TVL is 0.25% greater, rather then 0.3%. Note, '25' was originally '3', and the '10000's were originally '1000'.

That covers all the additions to the Pair contract, the last major change is removal of the _mintFee function. This was removed because the fee is being transferred to the FeeManager so _mintFee is no longer needed.

UniswapV2Factory.sol
contracts/uniswapV2/UniswapV2Factory.sol

Lines 17-27

```
    address public override governor;//Should never change
    address public override weth;//Should never change
    address public override wbtc;//Should never change
    address public override gfi;//Should never change
    address public override pathOracle;
    address public override priceOracle;
    address public override earningsManager;
    address public override feeManager;
    address public override dustPan;
    bool public override paused;
    uint public override slippage;
```

New variables that are shared by all swap pairs.

Lines 61-65

```
        IPathOracle(pathOracle).appendPath(token0, token1);
        IFeeManager(feeManager).catalougeTokens(token0, token1);
        if(token0 == gfi || token1 == gfi){ //Only add pairs that has GFI
as one of the tokens, otehrwise they won't have any earnings
            IEarningsManager(earningsManager).addSwapPair(pair);
        }
```

This code was added to the createPair function, once a pair is created, appendPath is called to try to find a valid swap path to go from the pair assets to wETH and wBTC, catalougeTokens adds the tokens to a token ledger in the FeeManager, it won't be used for this version of the FeeManager, but will be used for future versions, finally addSwapPair adds the pair to a pair ledger in EarningsManager, it won't be used for this version of the EarningsManager, but will be used for future versions.

Lines 87-125

```
    function setGovernor(address _governor) external {
        require(msg.sender == feeToSetter, 'Gravity Finance: FORBIDDEN');
        governor = _governor;
    }

    function setPathOracle(address _pathOracle) external {
        require(msg.sender == feeToSetter, 'Gravity Finance: FORBIDDEN');
        pathOracle = _pathOracle;
    }

    function setPriceOracle(address _priceOracle) external {
        require(msg.sender == feeToSetter, 'Gravity Finance: FORBIDDEN');
        priceOracle = _priceOracle;
    }

    function setEarningsManager(address _earningsManager) external {
        require(msg.sender == feeToSetter, 'Gravity Finance: FORBIDDEN');
        earningsManager = _earningsManager;
    }

    function setFeeManager(address _feeManager) external {
        require(msg.sender == feeToSetter, 'Gravity Finance: FORBIDDEN');
        feeManager = _feeManager;
    }

    function setDustPan(address _dustPan) external {
        require(msg.sender == feeToSetter, 'Gravity Finance: FORBIDDEN');
        dustPan = _dustPan;
    }

    function setPaused(bool _paused) external {
        require(msg.sender == feeToSetter, 'Gravity Finance: FORBIDDEN');
        paused = _paused;
    }
```

```
    function setSlippage(uint _slippage) external {
        require(msg.sender == feeToSetter, 'Gravity Finance: FORBIDDEN');
        slippage = _slippage;
    }
```

Functions only callable by the contract owner. These will be used so that the FeeManager and EarningsManager can be updated in the future to allow for the public to call their functions. Also so that PriceOracle and PathOracle can be updated if need be.

UniswapV2Router02.sol
contracts/uniswapV2/UniswapV2Router02.sol

Lines 239, 254, 270, 288, 306, 324

```
amounts[amounts.length - 1] = amounts[amounts.length - 1] * 9995/10000;
//Added this line to account for the 0.05% gov fee not being subtracted
from the final output amount
```

Since the 0.05% fee is being handled in a much different way compared to normal Uniswap, the last value in the amounts array is not accounting for the 0.05% fee.

UniswapV2Library.sol
contracts/uniswapV2/libraries/UniswapV2Library.sol

Line 26

```
hex'4a15fa71b7ffad80026ae5651db176877f454f08c1609ce20c7ab662086a4d14' //
init code hash
```

Changed this hash to be the hash of the new pair function, else pairFor will return the wrong address.