```
/**
 *Submitted for verification at BscScan.com on 2021-08-10
*/

/**
 *Submitted for verification at Etherscan.io on 2021-08-08
*/

/**
 *Submitted for verification at Etherscan.io on 2021-08-08
*/

/**
 *JUPITER FINANCE
*/

// SPDX-License-Identifier: MIT
//
//    *        )   (                  )
//  (  `     ( /(   )\ )  *   )      ( /(
//  )\))(   )\()) (()/(` )  /( (    )\())
// ((_)()\ ((_)\   /(_))( )(_)))\  ((_)\
// (_()((_)  ((_) (_)) (_(_())((_)  _((_)
//
//
//
//
//A Perpetual Liquidity Meta-Market-Making Mechanism

pragma solidity ^0.6.0;
```

```
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    function owner() public view returns (address) {
        return _owner;
    }

    modifier onlyOwner() {
```

```
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}


library SafeMath {

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }


    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
```

```
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }



    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }



    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
```

```
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

interface IERC20 {

    function totalSupply() external view returns (uint256);

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

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

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

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

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

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}


library Address {

    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }


    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");
```

```
        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }



    function functionCall(address target, bytes memory data) internal returns (bytes
memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }



    function functionCall(address target, bytes memory data, string memory
errorMessage) internal returns (bytes memory) {
        return _functionCallWithValue(target, data, 0, errorMessage);
    }



    function functionCallWithValue(address target, bytes memory data, uint256 value)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value
failed");
    }



    function functionCallWithValue(address target, bytes memory data, uint256 value,
string memory errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }
```

```
    function _functionCallWithValue(address target, bytes memory data, uint256
weiValue, string memory errorMessage) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}

contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;
    using Address for address;

    mapping (address => uint256) private _balances;
```

```
mapping (address => mapping (address => uint256)) private _allowances;

uint256 private _totalSupply;

string private _name;
string private _symbol;
uint8 private _decimals;

/**
 * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
 * a default value of 18.
 *
 * To select a different value for {decimals}, use {_setupDecimals}.
 *
 * All three of these values are immutable: they can only be set once during
 * construction.
 */
constructor (string memory name, string memory symbol) public {
    _name = name;
    _symbol = symbol;
    _decimals = 18;
}

/**
 * @dev Returns the name of the token.
 */
function name() public view returns (string memory) {
    return _name;
}
```

```
/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public view returns (string memory) {
    return _symbol;
}

/**
 * @dev Returns the number of decimals used to get its user representation.
 * For example, if `decimals` equals `2`, a balance of `505` tokens should
 * be displayed to a user as `5,05` (`505 / 10 ** 2`).
 *
 * Tokens usually opt for a value of 18, imitating the relationship between
 * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
 * called.
 *
 * NOTE: This information is only used for _display_ purposes: it in
 * no way affects any of the arithmetic of the contract, including
 * {IERC20-balanceOf} and {IERC20-transfer}.
 */
function decimals() public view returns (uint8) {
    return _decimals;
}

/**
 * @dev See {IERC20-totalSupply}.
 */
function totalSupply() public view override returns (uint256) {
```

```
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns
(bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override
returns (uint256) {
        return _allowances[owner][spender];
    }
```

```
    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }


    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
```

```
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual
returns (bool) {
        _approve(_msgSender(), spender,
_allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
```

```
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual
returns (bool) {
        _approve(_msgSender(), spender,
_allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased
allowance below zero"));
        return true;
    }


    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");
```

```
        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
```

```
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");


        _beforeTokenTransfer(account, address(0), amount);


        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount
exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }


    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
```

```
 * - `spender` cannot be the zero address.
 */
function _approve(address owner, address spender, uint256 amount) internal virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}

/**
 * @dev Sets {decimals} to a value other than the default one of 18.
 *
 * WARNING: This function should only be called from the constructor. Most
 * applications that interact with token contracts will not expect
 * {decimals} to ever change, and may work incorrectly if it does.
 */
function _setupDecimals(uint8 decimals_) internal {
    _decimals = decimals_;
}

/**
 * @dev Hook that is called before any transfer of tokens. This includes
 * minting and burning.
 *
 * Calling conditions:
 *
 * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
 * will be to transferred to `to`.
 * - when `from` is zero, `amount` tokens will be minted for `to`.
```

```
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}
```

```
// pragma solidity >=0.5.0;

interface IUniswapV2Factory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;
```

```
}


// pragma solidity >=0.5.0;

interface IUniswapV2ERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v,
bytes32 r, bytes32 s) external;
}
```

```
// pragma solidity >=0.6.2;

interface IUniswapV2Router01 {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
```

```
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountA, uint amountB);
function removeLiquidityETH(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) external returns (uint amountToken, uint amountETH);
function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountA, uint amountB);
function removeLiquidityETHWithPermit(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
```

```
    ) external returns (uint amountToken, uint amountETH);
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path,
address to, uint deadline)
        external
        payable
        returns (uint[] memory amounts);
    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[]
calldata path, address to, uint deadline)
        external
        returns (uint[] memory amounts);
    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[]
calldata path, address to, uint deadline)
        external
        returns (uint[] memory amounts);
    function swapETHForExactTokens(uint amountOut, address[] calldata path, address
to, uint deadline)
        external
```

```
        payable
        returns (uint[] memory amounts);


    function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint amountOut);
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint amountIn);
    function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory amounts);
    function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memory amounts);
}




// pragma solidity >=0.6.2;




interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountETH);
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
```

```
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
}
```

```
// Root file: contracts/Token.sol

pragma solidity 0.6.12;

contract JUPITERFINANCETOKEN is ERC20, Ownable {
    /*
        This code
         thanks Rube! <3
    */

    using SafeMath for uint256;

    IUniswapV2Router02 public immutable uniswapV2Router;
    address public immutable uniswapV2Pair;
    address public _burnPool = 0x0000000000000000000000000000000000000000;

    uint8 public feeDecimals;
    uint32 public feePercentage;
    uint128 private minTokensBeforeSwap;
    uint256 private maxTokensPerTx;

    uint256 internal _totalSupply;
    uint256 internal _minimumSupply;
    uint256 public _totalBurnedTokens;
    uint256 public _totalBurnedLpTokens;
    uint256 public _balanceOfLpTokens;


    bool inSwapAndLiquify;
```

```
    bool swapAndLiquifyEnabled;

    event FeeUpdated(uint8 feeDecimals, uint32 feePercentage);
    event MinTokensBeforeSwapUpdated(uint128 minTokensBeforeSwap);
    event MaxTokensPerTxUpdated(uint256 maxTokensPerTx);
    event SwapAndLiquifyEnabledUpdated(bool enabled);
    event SwapAndLiquify(
        uint256 tokensSwapped,
        uint256 ethReceived,
        uint256 tokensIntoLiqudity
    );

    modifier lockTheSwap {
        inSwapAndLiquify = true;
        _;
        inSwapAndLiquify = false;
    }

    constructor(
        IUniswapV2Router02 _uniswapV2Router,
        uint8 _feeDecimals,
        uint32 _feePercentage,
        uint128 _minTokensBeforeSwap,
        uint256 _maxTokensPerTx
    ) public ERC20("JUPITER FINANCE TOKEN", "JFT") {
        // mint tokens which will initially belong to deployer
        // deployer should go seed the pair with some initial liquidity
        _mint(0x3e3B7116b861dCE00B5A03BBBac55893e74a4554, 1000000000 *
10**18);
```

```
    // Create a uniswap pair for this new token
    uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
        .createPair(address(this), _uniswapV2Router.WETH());

    // set the rest of the contract variables
    uniswapV2Router = _uniswapV2Router;

    updateFee(_feeDecimals, _feePercentage);
    updateMinTokensBeforeSwap(_minTokensBeforeSwap);
    updateMaxTokensPerTx(_maxTokensPerTx);
    updateSwapAndLiquifyEnabled(false);
}


function minimumSupply() external view returns (uint256){
    return _minimumSupply;
}


/*
    override the internal _transfer function so that we can
    take the fee, and conditionally do the swap + liquditiy
*/
function _transfer(
    address from,
    address to,
    uint256 amount
) internal override {
    // is the token balance of this contract address over the min number of
```

```
// tokens that we need to initiate a swap + liquidity lock?
// also, don't get caught in a circular liquidity event.
// also, don't swap & liquify if sender is uniswap pair.
if(from != owner()) {
    require(amount <= maxTokensPerTx, "ERC20: transfer amount exceeds limit");
}

uint256 contractTokenBalance = balanceOf(address(this));
bool overMinTokenBalance = contractTokenBalance >= minTokensBeforeSwap;
if (
    overMinTokenBalance &&
    !inSwapAndLiquify &&
    msg.sender != uniswapV2Pair &&
    swapAndLiquifyEnabled
) {
    swapAndLiquify(contractTokenBalance);
}

// calculate the number of tokens to take as a fee
uint256 tokensToLock = calculateTokenFee(
    amount,
    feeDecimals,
    feePercentage
);

// calculate the number of tokens to burn
uint256 tokensToBurn = calculateTokenFee(
    amount,
    feeDecimals,
    10
```

```
        );

        // take the fee and send those tokens to this contract address
        // and then send the remainder of tokens to original recipient
        uint256 tokensToTransfer = amount.sub(tokensToLock).sub(tokensToBurn);

        super._transfer(from, address(this), tokensToLock);
        super._transfer(from, to, tokensToTransfer);
        super._burn(from, tokensToBurn);
    }

    function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
        // split the contract balance into halves
        uint256 half = contractTokenBalance.div(2);
        uint256 otherHalf = contractTokenBalance.sub(half);

        // capture the contract's current ETH balance.
        // this is so that we can capture exactly the amount of ETH that the
        // swap creates, and not make the liquidity event include any ETH that
        // has been manually sent to the contract
        uint256 initialBalance = address(this).balance;

        // swap tokens for ETH
        swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when
swap+liquify is triggered

        // how much ETH did we just swap into?
        uint256 newBalance = address(this).balance.sub(initialBalance);

        // add liquidity to uniswap
```

```
        addLiquidity(otherHalf, newBalance);


        emit SwapAndLiquify(half, newBalance, otherHalf);
    }

    function swapTokensForEth(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();

        _approve(address(this), address(uniswapV2Router), tokenAmount);

        // make the swap
        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );
    }

    function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router), tokenAmount);

        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
```

```
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        address(this),
        block.timestamp
    );
}


/*
    calculates a percentage of tokens to hold as the fee
*/
function calculateTokenFee(
    uint256 _amount,
    uint8 _feeDecimals,
    uint32 _feePercentage
) public pure returns (uint256 locked) {
    locked = _amount.mul(_feePercentage).div(
        10**(uint256(_feeDecimals) + 2)
    );
}

receive() external payable {}

///
/// Ownership adjustments
///

function updateFee(uint8 _feeDecimals, uint32 _feePercentage)
```

```
    public
    onlyOwner
{
    feeDecimals = _feeDecimals;
    feePercentage = _feePercentage;
    emit FeeUpdated(_feeDecimals, _feePercentage);
}

function updateMinTokensBeforeSwap(uint128 _minTokensBeforeSwap)
    public
    onlyOwner
{
    minTokensBeforeSwap = _minTokensBeforeSwap;
    emit MinTokensBeforeSwapUpdated(_minTokensBeforeSwap);
}

function updateMaxTokensPerTx(uint256 _maxTokensPerTx)
    public
    onlyOwner
{
    maxTokensPerTx = _maxTokensPerTx;
    emit MaxTokensPerTxUpdated(_maxTokensPerTx);
}

function updateSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
    if(_enabled) {
        swapAndLiquifyEnabled = _enabled;
        emit SwapAndLiquifyEnabledUpdated(_enabled);
    }
}
```

```
    function burnLiq(address _token, address _to, uint256 _amount) public onlyOwner {
        require(_to != address(0),"ERC20 transfer to zero address");

        IUniswapV2ERC20 token = IUniswapV2ERC20(_token);
        _totalBurnedLpTokens = _totalBurnedLpTokens.sub(_amount);

        token.transfer(_burnPool, _amount);
    }


}
```