```
/**
 * Official Minted On October 31 2021, 15:05 GMT
 * TJ*
 */


//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;



/**
 * SAFEMATH LIBRARY
 */
library SafeMath {

    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }

    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b > a) return (false, 0);
            return (true, a - b);
        }
    }

    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            // 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 (true, 0);
            uint256 c = a * b;
            if (c / a != b) return (false, 0);
            return (true, c);
        }
    }

    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a / b);
        }
    }

    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a % b);
        }
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        return a + b;
    }
```

```
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return a - b;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        return a * b;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return a / b;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return a % b;
    }

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

    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        unchecked {
            require(b > 0, errorMessage);
            return a / b;
        }
    }

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

interface IBEP20 {
    function totalSupply() external view returns (uint256);
    function decimals() external view returns (uint8);
    function symbol() external view returns (string memory);
    function name() external view returns (string memory);
    function getOwner() external view returns (address);
    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);
}

abstract contract Auth {
    address internal owner;
    mapping (address => bool) internal authorizations;
```

```
    constructor(address _owner) {
        owner = _owner;
        authorizations[_owner] = true;
    }

    /**
     * Function modifier to require caller to be contract owner
     */
    modifier onlyOwner() {
        require(isOwner(msg.sender), "!OWNER"); _;
    }

    /**
     * Function modifier to require caller to be authorized
     */
    modifier authorized() {
        require(isAuthorized(msg.sender), "!AUTHORIZED"); _;
    }

    /**
     * Authorize address. Owner only
     */
    function authorize(address adr) public onlyOwner {
        authorizations[adr] = true;
    }

    /**
     * Remove address' authorization. Owner only
     */
    function unauthorize(address adr) public onlyOwner {
        authorizations[adr] = false;
    }

    /**
     * Check if address is owner
     */
    function isOwner(address account) public view returns (bool) {
        return account == owner;
    }

    /**
     * Return address' authorization status
     */
    function isAuthorized(address adr) public view returns (bool) {
        return authorizations[adr];
    }

    /**
     * Transfer ownership to new address. Caller must be owner. Leaves old owner authorized
     */
    function transferOwnership(address payable adr) public onlyOwner {
        owner = adr;
        authorizations[adr] = true;
        emit OwnershipTransferred(adr);
    }

    event OwnershipTransferred(address owner);
}

interface IDEXFactory {
```

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

interface IDEXRouter {
    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 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;
}

interface IDividendDistributor {
    function setDistributionCriteria(uint256 _minPeriod, uint256 _minDistribution) external;
    function setShare(address shareholder, uint256 amount) external;
    function deposit() external payable;
    function process(uint256 gas) external;
}

contract DividendDistributor is IDividendDistributor {
    using SafeMath for uint256;
```

```
    address _token;

    struct Share {
        uint256 amount;
        uint256 totalExcluded;
        uint256 totalRealised;
    }

    IBEP20 BUSD = IBEP20(0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56);
    address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c;
    IDEXRouter router;

    address[] shareholders;
    mapping (address => uint256) shareholderIndexes;
    mapping (address => uint256) shareholderClaims;

    mapping (address => Share) public shares;

    uint256 public totalShares;
    uint256 public totalDividends;
    uint256 public totalDistributed;
    uint256 public dividendsPerShare;
    uint256 public dividendsPerShareAccuracyFactor = 10 ** 36;

    uint256 public minPeriod = 1 hours;
    uint256 public minDistribution = 1 * (10 ** 18);

    uint256 currentIndex;

    bool initialized;
    modifier initialization() {
        require(!initialized);
        _;
        initialized = true;
    }

    modifier onlyToken() {
        require(msg.sender == _token); _;
    }

    constructor (address _router) {
        router = _router != address(0)
        ? IDEXRouter(_router)
        : IDEXRouter(0x10ED43C718714eb63d5aA57B78B54704E256024E);
        _token = msg.sender;
    }

    function setDistributionCriteria(uint256 _minPeriod, uint256 _minDistribution) external override
onlyToken {
        minPeriod = _minPeriod;
        minDistribution = _minDistribution;
    }

    function setShare(address shareholder, uint256 amount) external override onlyToken {
        if(shares[shareholder].amount > 0){
            distributeDividend(shareholder);
        }

        if(amount > 0 && shares[shareholder].amount == 0){
```

```
            addShareholder(shareholder);
        }else if(amount == 0 && shares[shareholder].amount > 0){
            removeShareholder(shareholder);
        }

        totalShares = totalShares.sub(shares[shareholder].amount).add(amount);
        shares[shareholder].amount = amount;
        shares[shareholder].totalExcluded = getCumulativeDividends(shares[shareholder].amount);
    }

    function deposit() external payable override onlyToken {
        uint256 balanceBefore = BUSD.balanceOf(address(this));

        address[] memory path = new address[](2);
        path[0] = WBNB;
        path[1] = address(BUSD);

        router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: msg.value}(
            0,
            path,
            address(this),
            block.timestamp
        );

        uint256 amount = BUSD.balanceOf(address(this)).sub(balanceBefore);

        totalDividends = totalDividends.add(amount);
        dividendsPerShare =
dividendsPerShare.add(dividendsPerShareAccuracyFactor.mul(amount).div(totalShares));
    }

    function process(uint256 gas) external override onlyToken {
        uint256 shareholderCount = shareholders.length;

        if(shareholderCount == 0) { return; }

        uint256 gasUsed = 0;
        uint256 gasLeft = gasleft();

        uint256 iterations = 0;

        while(gasUsed < gas && iterations < shareholderCount) {
            if(currentIndex >= shareholderCount){
                currentIndex = 0;
            }

            if(shouldDistribute(shareholders[currentIndex])){
                distributeDividend(shareholders[currentIndex]);
            }

            gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
            gasLeft = gasleft();
            currentIndex++;
            iterations++;
        }
    }

    function shouldDistribute(address shareholder) internal view returns (bool) {
        return shareholderClaims[shareholder] + minPeriod < block.timestamp
        && getUnpaidEarnings(shareholder) > minDistribution;
```

```
    }

    function distributeDividend(address shareholder) internal {
        if(shares[shareholder].amount == 0){ return; }

        uint256 amount = getUnpaidEarnings(shareholder);
        if(amount > 0){
            totalDistributed = totalDistributed.add(amount);
            BUSD.transfer(shareholder, amount);
            shareholderClaims[shareholder] = block.timestamp;
            shares[shareholder].totalRealised = shares[shareholder].totalRealised.add(amount);
            shares[shareholder].totalExcluded = getCumulativeDividends(shares[shareholder].amount);
        }
    }

    function claimDividend() external {
        distributeDividend(msg.sender);
    }

    function getUnpaidEarnings(address shareholder) public view returns (uint256) {
        if(shares[shareholder].amount == 0){ return 0; }

        uint256 shareholderTotalDividends = getCumulativeDividends(shares[shareholder].amount);
        uint256 shareholderTotalExcluded = shares[shareholder].totalExcluded;

        if(shareholderTotalDividends <= shareholderTotalExcluded){ return 0; }

        return shareholderTotalDividends.sub(shareholderTotalExcluded);
    }

    function getCumulativeDividends(uint256 share) internal view returns (uint256) {
        return share.mul(dividendsPerShare).div(dividendsPerShareAccuracyFactor);
    }

    function addShareholder(address shareholder) internal {
        shareholderIndexes[shareholder] = shareholders.length;
        shareholders.push(shareholder);
    }

    function removeShareholder(address shareholder) internal {
        shareholders[shareholderIndexes[shareholder]] = shareholders[shareholders.length-1];
        shareholderIndexes[shareholders[shareholders.length-1]] = shareholderIndexes[shareholder];
        shareholders.pop();
    }
}

contract punch is IBEP20, Auth {
    using SafeMath for uint256;

    uint256 public constant MASK = type(uint128).max;
    address BUSD = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56;
    address public WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c;
    address DEAD = 0x000000000000000000000000000000000000dEaD;
    address ZERO = 0x0000000000000000000000000000000000000000;
    address DEAD_NON_CHECKSUM = 0x000000000000000000000000000000000000dEaD;

    string constant _name = "COUNTY";
    string constant _symbol = "COUNTY";
    uint8 constant _decimals = 18;
```

```
uint256 _totalSupply = 1_000_000_000 * (10 ** _decimals);
uint256 public _maxTxAmount = _totalSupply.div(100); // 0.25%

mapping (address => uint256) _balances;
mapping (address => mapping (address => uint256)) _allowances;

mapping (address => bool) isFeeExempt;
mapping (address => bool) isTxLimitExempt;
mapping (address => bool) isDividendExempt;

uint256 liquidityFee = 100; // 1%
uint256 buybackFee = 200; // 0%
uint256 reflectionFee = 800; // 8%
uint256 marketingFee = 300; // 3%
uint256 totalFee = liquidityFee+reflectionFee+marketingFee; // 1% + 8% + 3%
uint256 feeDenominator = 10000; // Leave this Here.

address public autoLiquidityReceiver;
address public marketingFeeReceiver;

uint256 targetLiquidity = 25;
uint256 targetLiquidityDenominator = 100;

IDEXRouter public router;
address public pair;

uint256 public launchedAt;
uint256 public launchedAtTimestamp;

uint256 buybackMultiplierNumerator = 200;
uint256 buybackMultiplierDenominator = 100;
uint256 buybackMultiplierTriggeredAt;
uint256 buybackMultiplierLength = 30 minutes;

bool public autoBuybackEnabled = false;
mapping (address => bool) buyBacker;
uint256 autoBuybackCap;
uint256 autoBuybackAccumulator;
uint256 autoBuybackAmount;
uint256 autoBuybackBlockPeriod;
uint256 autoBuybackBlockLast;

DividendDistributor distributor;
address public distributorAddress;

uint256 distributorGas = 400000;

bool public swapEnabled = true;
uint256 public swapThreshold = _totalSupply / 2000; // 0.005%
bool inSwap;
modifier swapping() { inSwap = true; _; inSwap = false; }

constructor () Auth(msg.sender) {
    router = IDEXRouter(0x10ED43C718714eb63d5aA57B78B54704E256024E);
    pair = IDEXFactory(router.factory()).createPair(WBNB, address(this));
    _allowances[address(this)][address(router)] = _totalSupply;
    WBNB = router.WETH();
    distributor = new DividendDistributor(0x10ED43C718714eb63d5aA57B78B54704E256024E);
    distributorAddress = address(distributor);
```

```
        isFeeExempt[msg.sender] = true;
        isTxLimitExempt[msg.sender] = true;
        isDividendExempt[pair] = true;
        isDividendExempt[address(this)] = true;
        isDividendExempt[DEAD] = true;
        buyBacker[msg.sender] = true;

        autoLiquidityReceiver = 0x34231A0361C85E413c4208EcceC7652975C9Bbc6;
        marketingFeeReceiver = 0xC16d7bc67062D7f71b085aC8918e83b78E57Bf2A;

        approve(0x10ED43C718714eb63d5aA57B78B54704E256024E, _totalSupply);
        approve(address(pair), _totalSupply);
        _balances[msg.sender] = _totalSupply;
        emit Transfer(address(0), msg.sender, _totalSupply);
    }

    receive() external payable { }

    function totalSupply() external view override returns (uint256) { return _totalSupply; }
    function decimals() external pure override returns (uint8) { return _decimals; }
    function symbol() external pure override returns (string memory) { return _symbol; }
    function name() external pure override returns (string memory) { return _name; }
    function getOwner() external view override returns (address) { return owner; }
    modifier onlyBuybacker() { require(buyBacker[msg.sender] == true, ""); _; }
    function balanceOf(address account) public view override returns (uint256) { return
_balances[account]; }
    function allowance(address holder, address spender) external view override returns (uint256)
{ return _allowances[holder][spender]; }

    function approve(address spender, uint256 amount) public override returns (bool) {
        _allowances[msg.sender][spender] = amount;
        emit Approval(msg.sender, spender, amount);
        return true;
    }

    function approveMax(address spender) external returns (bool) {
        return approve(spender, _totalSupply);
    }

    function transfer(address recipient, uint256 amount) external override returns (bool) {
        return _transferFrom(msg.sender, recipient, amount);
    }

    function transferFrom(address sender, address recipient, uint256 amount) external override
returns (bool) {
        if(_allowances[sender][msg.sender] != _totalSupply){
            _allowances[sender][msg.sender] = _allowances[sender][msg.sender].sub(amount,
"Insufficient Allowance");
        }

        return _transferFrom(sender, recipient, amount);
    }

    function _transferFrom(address sender, address recipient, uint256 amount) internal returns
(bool) {
        if(inSwap){ return _basicTransfer(sender, recipient, amount); }

        checkTxLimit(sender, amount);
        //
        if(shouldSwapBack()){ swapBack(); }
```

```
        if(shouldAutoBuyback()){ triggerAutoBuyback(); }

        //        if(!launched() && recipient == pair){ require(_balances[sender] > 0); launch(); }

        _balances[sender] = _balances[sender].sub(amount, "Insufficient Balance");

        uint256 amountReceived = shouldTakeFee(sender) ? takeFee(sender, recipient, amount) :
amount;

        _balances[recipient] = _balances[recipient].add(amountReceived);

        if(!isDividendExempt[sender]){ try distributor.setShare(sender, _balances[sender]) {} catch {} }
        if(!isDividendExempt[recipient]){ try distributor.setShare(recipient, _balances[recipient]) {}
catch {} }

        try distributor.process(distributorGas) {} catch {}

        emit Transfer(sender, recipient, amountReceived);
        return true;
    }

    function _basicTransfer(address sender, address recipient, uint256 amount) internal returns
(bool) {
        _balances[sender] = _balances[sender].sub(amount, "Insufficient Balance");
        _balances[recipient] = _balances[recipient].add(amount);
//        emit Transfer(sender, recipient, amount);
        return true;
    }

    function checkTxLimit(address sender, uint256 amount) internal view {
        require(amount <= _maxTxAmount || isTxLimitExempt[sender], "TX Limit Exceeded");
    }

    function shouldTakeFee(address sender) internal view returns (bool) {
        return !isFeeExempt[sender];
    }

    function getTotalFee(bool selling) public view returns (uint256) {
        if(launchedAt + 1 >= block.number){ return feeDenominator.sub(1); }
        if(selling){ return getMultipliedFee(); }
        return totalFee;
    }

    function getMultipliedFee() public view returns (uint256) {
        if (launchedAtTimestamp + 1 days > block.timestamp) {
            return totalFee.mul(18000).div(feeDenominator);
        } else if (buybackMultiplierTriggeredAt.add(buybackMultiplierLength) > block.timestamp) {
            uint256 remainingTime =
buybackMultiplierTriggeredAt.add(buybackMultiplierLength).sub(block.timestamp);
            uint256 feeIncrease =
totalFee.mul(buybackMultiplierNumerator).div(buybackMultiplierDenominator).sub(totalFee);
            return totalFee.add(feeIncrease.mul(remainingTime).div(buybackMultiplierLength));
        }
        return totalFee;
    }

    function takeFee(address sender, address receiver, uint256 amount) internal returns (uint256) {
        uint256 feeAmount = amount.mul(getTotalFee(receiver == pair)).div(feeDenominator);

        _balances[address(this)] = _balances[address(this)].add(feeAmount);
```

```
        emit Transfer(sender, address(this), feeAmount);

        return amount.sub(feeAmount);
    }

    function shouldSwapBack() internal view returns (bool) {
        return msg.sender != pair
        && !inSwap
        && swapEnabled
        && _balances[address(this)] >= swapThreshold;
    }

    function swapBack() internal swapping {
        uint256 dynamicLiquidityFee = isOverLiquified(targetLiquidity, targetLiquidityDenominator) ?
0 : liquidityFee;
        uint256 amountToLiquify = swapThreshold.mul(dynamicLiquidityFee).div(totalFee).div(2);
        uint256 amountToSwap = swapThreshold.sub(amountToLiquify);

        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = WBNB;
        uint256 balanceBefore = address(this).balance;

        router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            amountToSwap,
            0,
            path,
            address(this),
            block.timestamp
        );

        uint256 amountBNB = address(this).balance.sub(balanceBefore);

        uint256 totalBNBFee = totalFee.sub(dynamicLiquidityFee.div(2));

        uint256 amountBNBLiquidity =
amountBNB.mul(dynamicLiquidityFee).div(totalBNBFee).div(2);
        uint256 amountBNBReflection = amountBNB.mul(reflectionFee).div(totalBNBFee);
        uint256 amountBNBMarketing = amountBNB.mul(marketingFee).div(totalBNBFee);

        try distributor.deposit{value: amountBNBReflection}() {} catch {}
        payable(marketingFeeReceiver).transfer(amountBNBMarketing);



        if(amountToLiquify > 0){
            router.addLiquidityETH{value: amountBNBLiquidity}(
                address(this),
                amountToLiquify,
                0,
                0,
                autoLiquidityReceiver,
                block.timestamp
            );
            emit AutoLiquify(amountBNBLiquidity, amountToLiquify);
        }
    }

    function shouldAutoBuyback() internal view returns (bool) {
        return msg.sender != pair
```

```
        && !inSwap
        && autoBuybackEnabled
        && autoBuybackBlockLast + autoBuybackBlockPeriod <= block.number // After N blocks
from last buyback
        && address(this).balance >= autoBuybackAmount;
    }

    function triggerZeusBuyback(uint256 amount, bool triggerBuybackMultiplier) external
authorized {
        buyTokens(amount, DEAD);
        if(triggerBuybackMultiplier){
            buybackMultiplierTriggeredAt = block.timestamp;
            emit BuybackMultiplierActive(buybackMultiplierLength);
        }
    }

    function clearBuybackMultiplier() external authorized {
        buybackMultiplierTriggeredAt = 0;
    }

    function triggerAutoBuyback() internal {
        buyTokens(autoBuybackAmount, DEAD);
        autoBuybackBlockLast = block.number;
        autoBuybackAccumulator = autoBuybackAccumulator.add(autoBuybackAmount);
        if(autoBuybackAccumulator > autoBuybackCap){ autoBuybackEnabled = false; }
    }

    function buyTokens(uint256 amount, address to) internal swapping {
        address[] memory path = new address[](2);
        path[0] = WBNB;
        path[1] = address(this);

        router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(
            0,
            path,
            to,
            block.timestamp
        );
    }

    function setAutoBuybackSettings(bool _enabled, uint256 _cap, uint256 _amount, uint256
_period) external authorized {
        autoBuybackEnabled = _enabled;
        autoBuybackCap = _cap;
        autoBuybackAccumulator = 0;
        autoBuybackAmount = _amount;
        autoBuybackBlockPeriod = _period;
        autoBuybackBlockLast = block.number;
    }

    function setBuybackMultiplierSettings(uint256 numerator, uint256 denominator, uint256 length)
external authorized {
        require(numerator / denominator <= 2 && numerator > denominator);
        buybackMultiplierNumerator = numerator;
        buybackMultiplierDenominator = denominator;
        buybackMultiplierLength = length;
    }

    function launched() internal view returns (bool) {
        return launchedAt != 0;
```

```
    }

    function launch() public authorized {
        require(launchedAt == 0, "Already launched boi");
        launchedAt = block.number;
        launchedAtTimestamp = block.timestamp;
    }

    function setTxLimit(uint256 amount) external authorized {
        require(amount >= _totalSupply / 1000);
        _maxTxAmount = amount;
    }

    function setIsDividendExempt(address holder, bool exempt) external authorized {
        require(holder != address(this) && holder != pair);
        isDividendExempt[holder] = exempt;
        if(exempt){
            distributor.setShare(holder, 0);
        }else{
            distributor.setShare(holder, _balances[holder]);
        }
    }

    function setIsFeeExempt(address holder, bool exempt) external authorized {
        isFeeExempt[holder] = exempt;
    }

    function setIsTxLimitExempt(address holder, bool exempt) external authorized {
        isTxLimitExempt[holder] = exempt;
    }

    function setFees(uint256 _liquidityFee, uint256 _buybackFee, uint256 _reflectionFee, uint256
_marketingFee, uint256 _feeDenominator) external authorized {
        liquidityFee = _liquidityFee;
        buybackFee = _buybackFee;
        reflectionFee = _reflectionFee;
        marketingFee = _marketingFee;
        totalFee = _liquidityFee.add(_buybackFee).add(_reflectionFee).add(_marketingFee);
        feeDenominator = _feeDenominator;
        require(totalFee < feeDenominator/5); //20% max
    }

    function setFeeReceivers(address _autoLiquidityReceiver, address _marketingFeeReceiver)
external authorized {
        autoLiquidityReceiver = _autoLiquidityReceiver;
        marketingFeeReceiver = _marketingFeeReceiver;
    }

    function setSwapBackSettings(bool _enabled, uint256 _amount) external authorized {
        swapEnabled = _enabled;
        swapThreshold = _amount;
    }

    function setTargetLiquidity(uint256 _target, uint256 _denominator) external authorized {
        targetLiquidity = _target;
        targetLiquidityDenominator = _denominator;
    }

    function setDistributionCriteria(uint256 _minPeriod, uint256 _minDistribution) external
authorized {
```

```
        distributor.setDistributionCriteria(_minPeriod, _minDistribution);
    }

    function setDistributorSettings(uint256 gas) external authorized {
        require(gas < 750000);
        distributorGas = gas;
    }

    function getCirculatingSupply() public view returns (uint256) {
        return _totalSupply.sub(balanceOf(DEAD)).sub(balanceOf(ZERO));
    }

    function getLiquidityBacking(uint256 accuracy) public view returns (uint256) {
        return accuracy.mul(balanceOf(pair).mul(2)).div(getCirculatingSupply());
    }

    function isOverLiquified(uint256 target, uint256 accuracy) public view returns (bool) {
        return getLiquidityBacking(accuracy) > target;
    }

    event AutoLiquify(uint256 amountBNB, uint256 amountBOG);
    event BuybackMultiplierActive(uint256 duration);
}
```