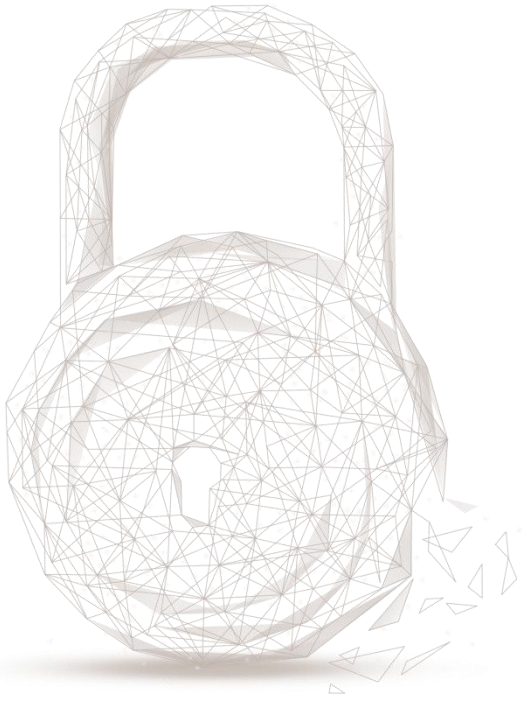
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number：202009042020

Smart Contract Name：

Token contract: DEGO

Sale contract: DegoOpenSale

Player book: PlayerBook

Smart Contract Address：

None

Smart Contract Address Link：

None

Start Date：2020.09.02

Completion Date：2020.09.04

Overall Result：Pass（Distinction）

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	ERC20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass
		Low-level Function (call/delegatecall) Security	Pass

		Returned Value Security	Pass
		selfdestruct Function Security	Pass
3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass
		Business Implementations	Pass
4	Integer Overflow/Underflow	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	Not Audit
12	Fake Deposit	-	Pass
13	event security	-	Pass

Note: Audit results and suggestions in code comments

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contracts DEGO&DegoOpenSale&PlayerBook, including Coding Standards, Security, and Business Logic. **DEGO&DegoOpenSale&PlayerBook contracts passed all audit items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1、Basic Token Information

Token name	dego.finance
Token symbol	DEGO
decimals	18
totalSupply	Initial token supply is 0 (Mintable with the cap of 21 million; Burnable)
Token type	ERC20

Table 1 - Basic Token Information

2、Token Vesting Information

This project has the relevant contract to implement token vesting (time lock), this part of the code is not contained in this audit contents. So the relevant audit result is not listed.

3、Custom Function Audit

- token burn fee & token reward fee

The token contract implements the internal function *_transfer* to do the token transfer. When a certain amount of tokens is sent through this function, the burn fee and the reward fee are calculated according to the specified rate and separately sent to the *_burnPool* and *_rewardPool.* The corresponding fee is deducted from the from address, the left amount is sent to the target address.

- token purchase

The DegoOpenSale contract implements the functions *buyDeGo* and *doBuy* to do the token purchase. When the contract is not paused and the current sale stage has started, the users can send ETH to the DegoOpenSale contract, the *Fallback* function will call the *buyDeGo* function to perform corresponding token purchase.

The users can be added to the whitelist, only the whitelist users can buy tokens when the sale stage (total 5 stages) is performed to 1 or 5.

- Player registration & Referral reward

The PlayerBook contract implements the functions to do the player registration and the referral reward settlement. The users can register the player id and referral, claim the corresponding referral reward.

Audited Source Code with Comments:

```
// Beosin (Chengdu LianAn) // File: DegoToken.sol
pragma solidity ^0.5.5; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

import '@openzeppelin/contracts/math/SafeMath.sol';
import '@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol';

import "../library/Governance.sol";

/// @title DegoToken Contract

contract DegoToken is Governance, ERC20Detailed{

    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.

    //events
    event eveSetRate(uint256 burn  rate, uint256 reward  rate);
    event eveRewardPool(address rewardPool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Mint(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    // for minters
    mapping (address => bool) public _minters; // Beosin (Chengdu LianAn) // Declare the mapping variable '_minters' for storing whether the corresponding address has minter permission.

    //token base data
    uint256 internal _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' for storing the total token supply.
    mapping(address => uint256) public _balances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_balances' for storing the token balance of corresponding address.
    mapping (address => mapping (address => uint256)) public _allowances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_allowances' for storing the allowance between two addresses.

    /// Constant token specific fields
    uint8 internal constant _decimals = 18; // Beosin (Chengdu LianAn) // Declare the variable '_decimals' for recording the token decimals. There are 18 decimals.
    uint256 public  _maxSupply = 0; // Beosin (Chengdu LianAn) // Declare the variable '_maxSupply' for storing the maximum token supply.

    ///
    bool public _openTransfer = false; // Beosin (Chengdu LianAn) // Declare the variable '_openTransfer' for storing the transfer function open flag.

    // hardcode limit rate
    uint256 public constant _maxGovernValueRate = 2000;//2000/10000
    uint256 public constant _minGovernValueRate = 10;  //10/10000
```

```
    uint256 public constant _rateBase = 10000;

    // additional variables for use if transaction fees ever became necessary
    uint256 public  _burnRate = 250;
    uint256 public  _rewardRate = 250;

    uint256 public _totalBurnToken = 0;
    uint256 public  totalRewardToken = 0;

    //todo reward pool!
    address public  rewardPool = 0x6666666666666666666666666666666666666666;
    //todo burn pool!
    address public _burnPool = 0x6666666666666666666666666666666666666666;

    /**
    * @dev set the token transfer switch
    */
    function enableOpenTransfer() public onlyGovernance
    {
        openTransfer = true; // Beosin (Chengdu LianAn) // Change the transfer function open flag to true.
    }


    /**
     * CONSTRUCTOR
     *
     * @dev Initialize the Token
     */

    constructor () public ERC20Detailed("dego.finance", "DEGO",  decimals) {
       uint256  exp =  decimals;
         maxSupply = 21000000 * (10**  exp); // Beosin (Chengdu LianAn) // Initialize the mint cap.
    }



    /**
    * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
    * @param spender The address which will spend the funds.
    * @param amount The amount of tokens to be spent.
    */
    function approve(address spender, uint256 amount) public
    returns (bool)
    {
        require(msg.sender != address(0), "ERC20: approve from the zero address"); // Beosin (Chengdu LianAn) //
The non-zero address check for caller.
        require(spender != address(0), "ERC20: approve to the zero address"); // Beosin (Chengdu LianAn) // The
non-zero address check for 'spender'.
```

```
        _allowances[msg.sender][spender] = amount; // Beosin (Chengdu LianAn) // The allowance which caller allowed to 'spender' is set to 'amount'.
        emit Approval(msg.sender, spender, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.

        return true;
    }

    /**
    * @dev Function to check the amount of tokens than an owner _allowed to a spender.
    * @param owner address The address which owns the funds.
    * @param spender address The address which will spend the funds.
    * @return A uint256 specifying the amount of tokens still available for the spender.
    */
    function allowance(address owner, address spender) public view
    returns (uint256)
    {
        return _allowances[owner][spender];
    }

    /**
    * @dev Gets the balance of the specified address.
    * @param owner The address to query the the balance of.
    * @return An uint256 representing the amount owned by the passed address.
    */
    function balanceOf(address owner) public  view
    returns (uint256)
    {
        return   balances[owner];
    }

    /**
    * @dev return the token total supply
    */
    function totalSupply() public view
    returns (uint256)
    {
        return _totalSupply;
    }

    /**
    * @dev for mint function
    */
    function mint(address account, uint256 amount) public
    {
        require(account != address(0), "ERC20: mint to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.
        require(_minters[msg.sender], "!minter"); // Beosin (Chengdu LianAn) // Require that the caller has minter permission.
```

```
        uint256 curMintSupply = _totalSupply.add(_totalBurnToken);
        uint256 newMintSupply = curMintSupply.add(amount);
        require( newMintSupply <= _maxSupply,"supply is max!"); // Beosin (Chengdu LianAn) // Require that the total token supply after this mint cannot exceed the mint cap.
        // Beosin (Chengdu LianAn) // Update the total token supply and the token balance of the specified address.
         totalSupply =   totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        // Beosin (Chengdu LianAn) // Trigger the events 'Mint' and 'Transfer'.
        emit Mint(address(0), account, amount);
        emit Transfer(address(0), account, amount);
    }
    // Beosin (Chengdu LianAn) // The function 'addMinter' is defined to grant the minter permission to the specified address '_minter'.
    function addMinter(address   minter) public onlyGovernance
    {
        minters[  minter] = true;
    }
    // Beosin (Chengdu LianAn) // The function 'removeMinter' is defined to remove the minter permission of specified address '_minter'.
    function removeMinter(address   minter) public onlyGovernance
    {
        minters[  minter] = false;
    }

    // Beosin (Chengdu LianAn) // Receiving ETH is not allowed in this contract.
    function() external payable {
        revert();
    }

    /**
    * @dev for govern value
    */
    function setRate(uint256 burn  rate, uint256 reward  rate) public
        onlyGovernance
    {
        // Beosin (Chengdu LianAn) // Validate the specified rate.
        require(_maxGovernValueRate >= burn_rate && burn_rate >= _minGovernValueRate,"invalid burn rate");
        require(_maxGovernValueRate >= reward_rate && reward_rate >= _minGovernValueRate,"invalid reward rate");
        // Beosin (Chengdu LianAn) // Update the specified rate.
        _burnRate = burn_rate;
        _rewardRate = reward_rate;

        emit eveSetRate(burn_rate, reward_rate); // Beosin (Chengdu LianAn) // Trigger the event 'eveSetRate'.
    }
```

```
    /**
     * @dev for set reward
     */
    function setRewardPool(address rewardPool) public
        onlyGovernance
    {
        require(rewardPool != address(0x0)); // Beosin (Chengdu LianAn) // Require that the specified address 'rewardPool' should not be the zero address.
        // Beosin (Chengdu LianAn) // Update the address of rewardPool.
        _rewardPool = rewardPool;

        emit eveRewardPool(  rewardPool); // Beosin (Chengdu LianAn) // Trigger the event 'eveRewardPool'.
    }
    /**
     * @dev transfer token for a specified address
     * @param to The address to transfer to.
     * @param value The amount to be transferred.
     */
    function transfer(address to, uint256 value) public
    returns (bool)
    {
        return   transfer(msg.sender,to,value); // Beosin (Chengdu LianAn) // Call the function '_transfer' to transfer tokens.
    }

    /**
     * @dev Transfer tokens from one address to another
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
     * @param value uint256 the amount of tokens to be transferred
     */
    function transferFrom(address from, address to, uint256 value) public
    returns (bool)
    {
        uint256 allow =   allowances[from][msg.sender];
          allowances[from][msg.sender] = allow.sub(value); // Beosin (Chengdu LianAn) // Update the allowance between 'from' and caller.

        return _transfer(from,to,value); // Beosin (Chengdu LianAn) // Call the function '_transfer' to transfer tokens.
    }



    /**
     * @dev Transfer tokens with fee
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
```

```
    * @param value uint256s the amount of tokens to be transferred
    */
    function _transfer(address from, address to, uint256 value) internal
    returns (bool)
    {
        // :)
        require(_openTransfer || from == governance, "transfer closed"); // Beosin (Chengdu LianAn) // Require that the transfer is open or the from address is 'governance'.

        require(from != address(0), "ERC20: transfer from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'from'.
        require(to != address(0), "ERC20: transfer to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'to'. Avoid losing transferred tokens.

        uint256 sendAmount = value; // Beosin (Chengdu LianAn) // Record the initial send amount.
        uint256 burnFee = (value.mul(  burnRate)).div(  rateBase); // Beosin (Chengdu LianAn) // Calculate the burn fee according to the specified burn rate.
        if (burnFee > 0) {
            //to burn
            balances[  burnPool] =   balances[  burnPool].add(burnFee); // Beosin (Chengdu LianAn) // Send the corresponding 'burnFee' to the '_burnPool'.
            totalSupply =   totalSupply.sub(burnFee); // Beosin (Chengdu LianAn) // Update the total token supply.
            sendAmount = sendAmount.sub(burnFee); // Beosin (Chengdu LianAn) // Update the actual send amount.

            totalBurnToken =   totalBurnToken.add(burnFee); // Beosin (Chengdu LianAn) // Update the total destroyed token amount.

            emit Transfer(from,   burnPool, burnFee); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer' to log the token destruction.
        }

        uint256 rewardFee = (value.mul(  rewardRate)).div(  rateBase); // Beosin (Chengdu LianAn) // Calculate the reward fee according to the specified reward rate.
        if (rewardFee > 0) {
            //to reward
            _balances[_rewardPool] = _balances[_rewardPool].add(rewardFee); // Beosin (Chengdu LianAn) // Send the corresponding 'rewardFee' to the '_rewardPool'.
            sendAmount = sendAmount.sub(rewardFee); // Beosin (Chengdu LianAn) // Update the actual send amount.

            _totalRewardToken = _totalRewardToken.add(rewardFee); // Beosin (Chengdu LianAn) // Update the total reward token amount.

            emit Transfer(from, _rewardPool, rewardFee); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer' to log the token reward.
        }
        // Beosin (Chengdu LianAn) // Alter the token balance of 'from' and 'to'.
```

```
        _balances[from] = _balances[from].sub(value);
        _balances[to] = _balances[to].add(sendAmount);

        emit Transfer(from, to, sendAmount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer' to log the actual token transfer.

        return true;
    }
}


// Beosin (Chengdu LianAn) // File: DegoOpenSale.sol
pragma solidity ^0.5.5; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

import '@openzeppelin/contracts/lifecycle/Pausable.sol';
import '@openzeppelin/contracts/ownership/Ownable.sol';
import "@openzeppelin/contracts/utils/Address.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";

import "../library/SafeERC20.sol";
import "../interface/IERC20.sol";

/// @title DegoOpenSale Contract

contract DegoOpenSale is Pausable,Ownable {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.
    using Address for address;
    // Beosin (Chengdu LianAn) // Declare the structure 'condition' for storing the basic information of sale stage.
    struct condition {
        uint256 price;          //  dego per eth
        uint256 limitFund;      //a quota
        uint256 startTime;      //the stage start time
        uint256 maxSoldAmount;  //the stage max sold amount
    }

    //
    uint8 public constant _whiteListStage1 = 1;
    uint8 public constant _whiteListStage5 = 5;
    //
    /// All deposited ETH will be instantly forwarded to this address.
    address payable public _teamWallet = 0x6666666666666666666666666666666666666666;

    /// IERC20 compilant _dego token contact instance
    IERC20 public _dego = IERC20(0x6666666666666666666666666666666666666666);

    /// tags show address can join in open sale
```

```
    mapping (uint8 =>  mapping (address => bool)) public _fullWhiteList;

    //the stage condition map
    mapping (uint8 => condition) public _stageCondition;

    //the user get fund per stage
    mapping (uint8 =>  mapping (address => uint256) ) public _stageFund;


    //the stage had sold amount
    mapping (uint8 => uint256) public _stageSoldAmount;

    /*
     * EVENTS
     */
    event eveNewSale(address indexed destAddress, uint256 ethCost, uint256 gotTokens);
    event eveClaim(address indexed destAddress, uint256 gotTokens);
    event eveTeamWallet(address wallet);



    /// @dev valid the address
    modifier validAddress( address addr ) {
        require(addr != address(0x0)); // Beosin (Chengdu LianAn) // Require that the specified address 'addr' should not be the zero address.
        require(addr != address(this)); // Beosin (Chengdu LianAn) // Require that the specified address 'addr' should not be this contract address.
         ;
    }

    constructor()
        public
    {
        pause(); // Beosin (Chengdu LianAn) // Call the function 'pause', making the contract is paused. Make the token purchase cannot be successful.

        // uint256 testRate = 100;
        // uint256 testRate2 = 1000;

        // setCondition(1,3500 * testRate2,10*1e18/testRate, now,          525000*1e18);
        // setCondition(2,2500 * testRate2,5 *1e18/testRate, now + 1 days, 375000*1e18);
        // setCondition(3,2000 * testRate2,5 *1e18/testRate, now + 1 days, 600000*1e18);
        // setCondition(4,1500 * testRate2,5 *1e18/testRate, now + 1 days, 450000*1e18);
        // setCondition(5,1500 * testRate2,2 *1e18/testRate, now + 3 days, 150000*1e18);

        setCondition(1,3500 ,10*1e18, now,          525000*1e18);
        setCondition(2,2500 ,5 *1e18, now + 1 days, 375000*1e18);
        setCondition(3,2000 ,5 *1e18, now + 1 days, 600000*1e18);
        setCondition(4,1500 ,5 *1e18, now + 1 days, 450000*1e18);
```

```
        setCondition(5,1500 ,2 *1e18, now + 3 days, 150000*1e18);

    }


    /**
    * @dev for set team wallet
    */
    function setTeamWallet(address payable wallet) public
        onlyOwner
    {
        require(wallet != address(0x0)); // Beosin (Chengdu LianAn) // Require that the specified address 'wallet' should not be the zero address.

        _teamWallet = wallet; // Beosin (Chengdu LianAn) // Update the address of team wallet.

        emit eveTeamWallet(wallet);
    }

    /// @dev set the sale condition for every stage;
    function setCondition(
    uint8 stage,
    uint256 price,
    uint256 limitFund,
    uint256 startTime,
    uint256 maxSoldAmount )
        internal
        onlyOwner // Beosin (Chengdu LianAn) // Require that only the contract owner can call this function.
    {

          stageCondition[stage].price = price;
          stageCondition[stage].limitFund =limitFund;
          stageCondition[stage].startTime= startTime;
          stageCondition[stage].maxSoldAmount=maxSoldAmount;
    }



    /// @dev set the sale start time for every stage;
    function setStartTime(uint8 stage,uint256 startTime )
        public
        onlyOwner
    {
        _stageCondition[stage].startTime = startTime; // Beosin (Chengdu LianAn) // Update the start time of specified sale stage.
    }

    /// @dev batch set quota for user admin
    /// if openTag <=0, removed
```

```
    function setWhiteList(uint8 stage, address[] calldata users, bool openTag)
        external
        onlyOwner
    {
        // Beosin (Chengdu LianAn) // Batch set the whitelist status of specified 'stage' and 'users'.
        for (uint256 i = 0; i < users.length; i++) {
            _fullWhiteList[stage][users[i]] = openTag;
        }
    }

    /// @dev batch set quota for early user quota
    /// if openTag <=0, removed
    function addWhiteList(uint8 stage, address user, bool openTag)
        external
        onlyOwner
    {
        _fullWhiteList[stage][user] = openTag; // Beosin (Chengdu LianAn) // Set the whitelist status of specified
'stage' and 'user'.
    }

    /**
     * @dev If anybody sends Ether directly to this  contract, consider he is getting DeGo token
     */
    function () external payable {
        buyDeGo(msg.sender); // Beosin (Chengdu LianAn) // Call the function 'buyDeGo' to buy tokens.
    }

    //
    function getStage( ) view public returns(uint8) {
        // Beosin (Chengdu LianAn) // Traversal the specified 5 stages, return the current stage.
        for(uint8 i=1; i<6; i++){
            uint256 startTime =   stageCondition[i].startTime;
            if(now >= startTime &&   stageSoldAmount[i] <   stageCondition[i].maxSoldAmount ){
                return i;
            }
        }

        return 0;
    }

    //
    function conditionCheck( address addr ) view internal  returns(uint8) {

        uint8 stage = getStage(); // Beosin (Chengdu LianAn) // Get the current stage.
        require(stage!=0,"stage not begin"); // Beosin (Chengdu LianAn) // Require that the purchase stage has
started.

        uint256 fund = _stageFund[stage][addr]; // Beosin (Chengdu LianAn) // Get the costed ETH fund of 'addr'.
```

```
        require(fund < _stageCondition[stage].limitFund,"stage fund is full "); // Beosin (Chengdu LianAn) //
Require that the current stage fund is not full.

        return stage;
    }

    /// @dev Exchange msg.value ether to Dego for account recepient
    /// @param receipient Dego tokens receiver
    function buyDeGo(address receipient)
        internal
        whenNotPaused // Beosin (Chengdu LianAn) // Require that the function is callable (the contract is not
paused).
        validAddress(receipient) // Beosin (Chengdu LianAn) // Require the specified address 'receipient' is valid.
        returns (bool)
    {
        // Do not allow contracts to game the system

        require(tx.gasprice <= 500000000000 wei);

        uint8 stage = conditionCheck(receipient); // Beosin (Chengdu LianAn) // Call the function
'conditionCheck' to verify the stage.
        if(stage==  whiteListStage1 || stage==  whiteListStage5 ){
            require(  fullWhiteList[stage][receipient],"your are not in the whitelist "); // Beosin (Chengdu LianAn) //
Require that only the whitelist users can buy tokens at stage 1 or 5.
        }

        doBuy(receipient, stage); // Beosin (Chengdu LianAn) // Call the function 'doBuy' to buy tokens.

        return true;
    }


    /// @dev Buy DeGo token normally
    function doBuy(address receipient, uint8 stage) internal {
        // protect partner quota in stage one
        uint256 value = msg.value; // Beosin (Chengdu LianAn) // Record the ETH amount carried in this
transaction.
        uint256 fund = _stageFund[stage][receipient]; // Beosin (Chengdu LianAn) // Get the costed ETH fund of
'receipient'.
        fund = fund.add(value); // Beosin (Chengdu LianAn) // Record the costed ETH fund after this purchase.
        if (fund > _stageCondition[stage].limitFund ) { // Beosin (Chengdu LianAn) // If the costed ETH fund
exceeds the limit, refund the excess.
            uint256 refund = fund.sub(_stageCondition[stage].limitFund);
            value = value.sub(refund); // Beosin (Chengdu LianAn) // Update the actual consumption ETH amount.
            msg.sender.transfer(refund); // Beosin (Chengdu LianAn) // Refund the corresponding ETHs to the
caller.
        }

        uint256 soldAmount = _stageSoldAmount[stage]; // Beosin (Chengdu LianAn) // Record the sold amount
```

```
of the specified stage at the current time.
        uint256 tokenAvailable = _stageCondition[stage].maxSoldAmount.sub(soldAmount); // Beosin (Chengdu LianAn) // Get the available token amount that can be sold.
        require(tokenAvailable > 0);

        uint256 costValue = 0; // Beosin (Chengdu LianAn) // The local variable 'costValue' for recording the ETH amount to be costed.
        uint256 getTokens = 0; // Beosin (Chengdu LianAn) // The local variable 'getTokens' for recording the token amount that could actually get.

        // all conditions has checked in the caller functions
        uint256 price =   stageCondition[stage].price;
        getTokens = price * value;
        // Beosin (Chengdu LianAn) // Accurate the amount of actually gettable tokens and costed ETH.
        if (tokenAvailable >= getTokens) {
            costValue = value;
        } else {
            costValue = tokenAvailable.div(price);
            getTokens = tokenAvailable;
        }

        if (costValue > 0) {

            stageSoldAmount[stage] =   stageSoldAmount[stage].add(getTokens); // Beosin (Chengdu LianAn) // Update the sold amount of the specified stage at the current time.
            stageFund[stage][receipient]=  stageFund[stage][receipient].add(costValue); // Beosin (Chengdu LianAn) // Update the costed ETH fund.

            dego.mint(msg.sender, getTokens); // Beosin (Chengdu LianAn) // Call the function 'mint' to issue tokens to the caller.

            emit eveNewSale(receipient, costValue, getTokens); // Beosin (Chengdu LianAn) // Trigger the event 'eveNewSale'.
        }

        // not enough token sale, just return eth
        uint256 toReturn = value.sub(costValue);
        if (toReturn > 0) {
            msg.sender.transfer(toReturn); // Beosin (Chengdu LianAn) // Return corresponding ETHs to the caller.
        }

    }

    // get sale eth
    function seizeEth() external  {
        uint256 _currentBalance =  address(this).balance; // Beosin (Chengdu LianAn) // Get the ETH balance of this contract.
        _teamWallet.transfer(_currentBalance); // Beosin (Chengdu LianAn) // Send all ETHs of this contract to
```

the '_teamWallet'.
}

}

// Beosin (Chengdu LianAn) // File: PlayerBook.sol
pragma solidity ^0.5.0; **// Beosin (Chengdu LianAn) // Fixing compiler version is recommended.**

import '@openzeppelin/contracts/math/SafeMath.sol';
import '@openzeppelin/contracts/ownership/Ownable.sol';
import "../library/NameFilter.sol";
import "../library/SafeERC20.sol";
import "../library/Governance.sol";
import "../interface/IPlayerBook.sol";

contract PlayerBook is Governance, IPlayerBook {
using NameFilter for string; **// Beosin (Chengdu LianAn) // Use the NameFilter library for name filter.**
using SafeMath for uint256; **// Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.**
using SafeERC20 for IERC20; **// Beosin (Chengdu LianAn) // Use the SafeERC20 library for safe external ERC20 contract calling.**

// register pools
mapping (address => bool) public pools;

// (addr => pID) returns player id by address
mapping (address => uint256) public pIDxAddr;
// (name => pID) returns player id by name
mapping (bytes32 => uint256) public pIDxName;
// (pID => data) player data
mapping (uint256 => Player) public plyr;
// (pID => name => bool) list of names a player owns. (used so you can change your display name amoungst any name you own)
mapping (uint256 => mapping (bytes32 => bool)) public plyrNames;

// the of refrerrals
uint256 public _totalReferReward;
// total number of players
uint256 public _pID;
// total register name count
uint256 public _totalRegisterCount = 0;

// the direct refer's reward rate
uint256 public _refer1RewardRate = 700; //7%
// the second direct refer's reward rate
uint256 public _refer2RewardRate = 300; //3%
// base rate

```
    uint256 public _baseRate = 10000;

    // base price to register a name
    uint256 public _registrationBaseFee = 100 finney;
    // register fee count step
    uint256 public _registrationStep = 100;
    // add base price for one step
    uint256 public   stepFee = 100 finney;

    bytes32 public _defaulRefer ="dego";

    address payable public   teamWallet = 0x6666666666666666666666666666666666666666; // Beosin (Chengdu LianAn) // Declare the variable '_teamWallet' for storing the team address.

    IERC20 public   dego = IERC20(0x0);
    // Beosin (Chengdu LianAn) // Declare the structure 'Player' for storing the player information.
    struct Player {
        address addr;
        bytes32 name;
        uint8 nameCount;
        uint256 laff;
        uint256 amount;
        uint256 rreward;
        uint256 allReward;
        uint256 lv1Count;
        uint256 lv2Count;
    }
    // Beosin (Chengdu LianAn) // Declare the relevant events.
    event eveClaim(uint256 pID, address addr, uint256 reward, uint256 balance );
    event eveBindRefer(uint256 pID, address addr, bytes32 name, uint256 affID, address affAddr, bytes32 affName);
    event eveDefaultPlayer(uint256 pID, address addr, bytes32 name);
    event eveNewName(uint256 pID, address addr, bytes32 name, uint256 affID, address affAddr, bytes32 affName, uint256 balance  );
    event eveSettle(uint256 pID, uint256 affID, uint256 aff  affID, uint256 affReward, uint256 aff  affReward, uint256 amount);
    event eveAddPool(address addr);
    event eveRemovePool(address addr);

    // Beosin (Chengdu LianAn) // Constructor, initialize the default player.
    constructor()
        public
    {
        _pID = 0;
        _totalReferReward = 0;
        addDefaultPlayer(_teamWallet,_defaulRefer); // Beosin (Chengdu LianAn) // Call the function 'addDefaultPlayer' to add the default player.
    }
```

```
    /**
     * check address
     */
    modifier validAddress( address addr ) {
        require(addr != address(0x0)); // Beosin (Chengdu LianAn) // Require that the specified address should not be the zero address.
        ;
    }

    /**
     * check pool
     */
    modifier isRegisteredPool(){
        require(  pools[msg.sender],"invalid pool address!"); // Beosin (Chengdu LianAn) // Require that the pool of caller is valid.
        _;
    }

    /**
     * contract dego balances
     */
    function balances()
        public
        view
        returns(uint256)
    {
        return (  dego.balanceOf(address(this))); // Beosin (Chengdu LianAn) // Return the DEGO token balance of this contract.
    }

    // only function for creating additional rewards from dust
    function seize(IERC20 asset) external returns (uint256 balance) {
        require(address(  dego) != address(asset), "forbbiden dego");

        balance = asset.balanceOf(address(this));
        asset.safeTransfer(_teamWallet, balance); // Beosin (Chengdu LianAn) // Withdraw the specified ERC20 tokens of this contract to the '_teamWallet'.
    }

    // get register fee
    function seizeEth() external  {
        uint256 _currentBalance =  address(this).balance;
        _teamWallet.transfer(_currentBalance); // Beosin (Chengdu LianAn) // Withdraw the ETHs of this contract to the '_teamWallet'.
    }

    /**
```

```
    * revert invalid transfer action
    */
    function() external payable {
        revert(); // Beosin (Chengdu LianAn) // Receiving ETH is not allowed in this contract.
    }


    /**
     * registe a pool
     */
    function addPool(address poolAddr)
        onlyGovernance
        public
    {
        require( !_pools[poolAddr], "derp, that pool already been registered"); // Beosin (Chengdu LianAn) //
Require that the pool of 'poolAddr' should not be registered.

        _pools[poolAddr] = true;

        emit eveAddPool(poolAddr); // Beosin (Chengdu LianAn) // Trigger the event 'eveAddPool'.
    }

    /**
     * remove a pool
     */
    function removePool(address poolAddr)
        onlyGovernance
        public
    {
        require(   pools[poolAddr], "derp, that pool must be registered"); // Beosin (Chengdu LianAn) // Require
that the pool of 'poolAddr' should be registered.

          pools[poolAddr] = false;

        emit eveRemovePool(poolAddr); // Beosin (Chengdu LianAn) // Trigger the event 'eveRemovePool'.
    }

    /**
     * resolve the refer's reward from a player
     */
    function settleReward(address from, uint256 amount)
        isRegisteredPool() // Beosin (Chengdu LianAn) // Require that the pool of caller is valid.
        validAddress(from) // Beosin (Chengdu LianAn) // Check the validity of the 'from'.
        external
        returns (uint256)
    {
         // set up our tx event data and determine if player is new or not
        _determinePID(from); // Beosin (Chengdu LianAn) // Call the function '_determinePID' to check whether
the specified player 'from' is new.
```

```
        uint256 pID = _pIDxAddr[from];
        uint256 affID = _plyr[pID].laff;
        // Beosin (Chengdu LianAn) // Check the referral of specified player.
        if(affID <= 0 ){
            affID = _pIDxName[_defaulRefer];
            _plyr[pID].laff = affID;
        }
        // Beosin (Chengdu LianAn) // Check validity of 'amount'.
        if(amount <= 0){
            return 0;
        }

        uint256 fee = 0;
        // Beosin (Chengdu LianAn) // Settle and store the corresponding reward of referral.
        // father
        uint256 affReward = (amount.mul(_refer1RewardRate)).div(_baseRate);
          plyr[affID].rreward =   plyr[affID].rreward.add(affReward);
          totalReferReward =   totalReferReward.add(affReward);
        fee = fee.add(affReward);


        // grandfather
        uint256 aff  affID =   plyr[affID].laff;
        uint256 aff  affReward = amount.mul(  refer2RewardRate).div(  baseRate);
        if(aff  affID <= 0){
            aff  affID =   pIDxName[  defaulRefer];
        }
          plyr[aff  affID].rreward =   plyr[aff  affID].rreward.add(aff  affReward);
          totalReferReward=   totalReferReward.add(aff  affReward);

          plyr[pID].amount =   plyr[pID].amount.add( amount);

        fee = fee.add(aff  affReward);

        emit eveSettle( pID,affID,aff  affID,affReward,aff  affReward,amount); // Beosin (Chengdu LianAn) //
Trigger the event 'eveSettle'.

        return fee;
    }

    /**
     * claim all of the refer reward.
     */
    function claim()
        public
    {
        address addr = msg.sender;
        uint256 pid = _pIDxAddr[addr];
```

```
        uint256 reward = _plyr[pid].rreward;

        require(reward > 0,"only have reward");

        //reset
        _plyr[pid].allReward = _plyr[pid].allReward.add(reward);
        _plyr[pid].rreward = 0;

        //get reward
        _dego.safeTransfer(addr, reward); // Beosin (Chengdu LianAn) // Send the reward to the caller.

        // fire event
        emit eveClaim(_pIDxAddr[addr], addr, reward, balances()); // Beosin (Chengdu LianAn) // Trigger the event 'eveClaim'.
    }


    /**
     * check name string
     */
    function checkIfNameValid(string memory nameStr)
        public
        view
        returns(bool)
    {
        bytes32 name = nameStr.nameFilter();
        if (  pIDxName[name] == 0)
            return (true);
        else
            return (false);
    }

    /**
     * @dev add a default player
     */
    function addDefaultPlayer(address addr, bytes32 name)
        private
    {
        _pID++;

        _plyr[_pID].addr = addr;
        _plyr[_pID].name = name;
        _plyr[_pID].nameCount = 1;
        _pIDxAddr[addr] = _pID;
        _pIDxName[name] = _pID;
        _plyrNames[_pID][name] = true;

        //fire event
        emit eveDefaultPlayer(_pID,addr,name); // Beosin (Chengdu LianAn) // Trigger the event
```

'eveDefaultPlayer'.

```
    }

    /**
     * @dev set refer reward rate
     */
    function setReferRewardRate(uint256 refer1Rate, uint256 refer2Rate ) public
        onlyGovernance
    {
        // Beosin (Chengdu LianAn) // Update the referral reward rate.
         refer1RewardRate = refer1Rate;
         refer2RewardRate = refer2Rate;
    }

    /**
     * @dev set registration step count
     */
    function setRegistrationStep(uint256 registrationStep) public
        onlyGovernance
    {
         registrationStep = registrationStep; // Beosin (Chengdu LianAn) // Update the registration step.
    }

    /**
     * @dev set dego contract address
     */
    function setDegoContract(address dego)  public
        onlyGovernance{
         dego = IERC20(dego); // Beosin (Chengdu LianAn) // Set the DEGO contract address.
    }


    /**
     * @dev registers a name.  UI will always display the last name you registered.
     * but you will still own all previously registered names to use as affiliate
     * links.
     * - must pay a registration fee.
     * - name must be unique
     * - names will be converted to lowercase
     * - cannot be only numbers
     * - cannot start with 0x
     * - name must be at least 1 char
     * - max length of 32 characters long
     * - allowed characters: a-z, 0-9
     * -functionhash- 0x921dec21 (using ID for affiliate)
     * -functionhash- 0x3ddd4698 (using address for affiliate)
     * -functionhash- 0x685ffd83 (using name for affiliate)
     * @param nameString players desired name
     * @param affCode affiliate name of who refered you
```

```
     * (this might cost a lot of gas)
     */

    function registerNameXName(string memory nameString, string memory affCode)
        public
        payable
    {

        // make sure name fees paid
        require (msg.value >= this.getRegistrationFee(), "umm.....  you have to pay the name fee");

        // filter name + condition checks
        bytes32 name = NameFilter.nameFilter(nameString);
        // if names already has been used
        require(  pIDxName[name] == 0, "sorry that names already taken");

        // set up address
        address addr = msg.sender;
         // set up our tx event data and determine if player is new or not
          determinePID(addr); // Beosin (Chengdu LianAn) // Call the function '_determinePID' to check whether the specified player 'addr' is new.
        // fetch player id
        uint256 pID =   pIDxAddr[addr];
        // if names already has been used
        require(  plyrNames[pID][name] == false, "sorry that names already taken");

        // add name to player profile, registry, and name book
          plyrNames[pID][name] = true;
          pIDxName[name] = pID;
          plyr[pID].name = name;
          plyr[pID].nameCount++;

          totalRegisterCount++;


        //try bind a refer
        if(_plyr[pID].laff == 0){

            bytes memory tempCode = bytes(affCode);
            bytes32 affName = 0x0;
            if (tempCode.length >= 0) {
                assembly {
                    affName := mload(add(tempCode, 32))
                }
            }

            _bindRefer(addr,affName);
        }
        uint256 affID = _plyr[pID].laff;
```

```
        // fire event
        emit eveNewName(pID, addr, name, affID, _plyr[affID].addr, _plyr[affID].name, _registrationBaseFee ); // Beosin (Chengdu LianAn) // Trigger the event 'eveNewName'.
    }

    /**
     * @dev bind a refer,if affcode invalid, use default refer
     */
    function bindRefer( address from, string calldata  affCode )
        isRegisteredPool()
        external
        returns (bool)
    {

        bytes memory tempCode = bytes(affCode);
        bytes32 affName = 0x0;
        if (tempCode.length >= 0) {
            assembly {
                affName := mload(add(tempCode, 32))
            }
        }

        return   bindRefer(from, affName);
    }


    /**
     * @dev bind a refer,if affcode invalid, use default refer
     */
    function   bindRefer( address from, bytes32  name )
        // Beosin (Chengdu LianAn) // Check the validity of the corresponding address.
        validAddress(msg.sender)
        validAddress(from)
        private
        returns (bool)
    {
        // set up our tx event data and determine if player is new or not
        _determinePID(from); // Beosin (Chengdu LianAn) // Call the function '_determinePID' to check whether the specified player 'from' is new.
        // fetch player id
        uint256 pID = _pIDxAddr[from];
        if( _plyr[pID].laff != 0){
            return false;
        }
        if (_pIDxName[name] == 0){
            //unregister name
            name = _defaulRefer;
        }
```

```
        uint256 affID = _pIDxName[name];
        if( affID == pID){
            affID = _pIDxName[_defaulRefer];
        }

        _plyr[pID].laff = affID;

        //lvcount
        _plyr[affID].lv1Count++;
        uint256 aff_affID = _plyr[affID].laff;
        if(aff  affID != 0 ){
            plyr[aff  affID].lv2Count++;
        }

        // fire event
        emit eveBindRefer(pID, from, name, affID,   plyr[affID].addr,   plyr[affID].name); // Beosin (Chengdu LianAn) // Trigger the event 'eveBindRefer'.
        return true;
    }

    //
    // Beosin (Chengdu LianAn) // The function '_determinePID' is defined to check whether the specified player 'addr' is new.
    function   determinePID(address addr)
        private
        returns (bool)
    {
        if (  pIDxAddr[addr] == 0) // Beosin (Chengdu LianAn) // If the player id of the 'addr' is empty, register it as a new player.
        {
            pID++;
            pIDxAddr[addr] =   pID;
            plyr[  pID].addr = addr;

            // set the new player bool to true
            return (true);
        } else {
            return (false);
        }
    }
    // Beosin (Chengdu LianAn) // The function 'hasRefer' is defined to check whether the specified address has referral.
    function hasRefer(address from)
        isRegisteredPool()
        external
        returns(bool)
    {
        _determinePID(from); // Beosin (Chengdu LianAn) // Call the function '_determinePID' to check whether
```

```
the specified player 'from' is new.
        uint256 pID =  _pIDxAddr[from];
        return (_plyr[pID].laff > 0);
    }

    // Beosin (Chengdu LianAn) // The function 'getPlayerName' is defined to query the player name of the specified address.
    function getPlayerName(address from)
        external
        view
        returns (bytes32)
    {
        uint256 pID =  _pIDxAddr[from];
        if(_pID==0){
            return "";
        }
        return (_plyr[pID].name);
    }
    // Beosin (Chengdu LianAn) // The function 'getPlayerLaffName' is defined to query the referral name of the specified player.
    function getPlayerLaffName(address from)
        external
        view
        returns (bytes32)
    {
        uint256 pID =    pIDxAddr[from];
        if(  pID==0){
            return "";
        }

        uint256 aID=  plyr[pID].laff;
        if( aID== 0){
            return "";
        }

        return (  plyr[aID].name);
    }
    // Beosin (Chengdu LianAn) // The function 'getPlayerInfo' is defined to query the player information of specified address 'from'.
    function getPlayerInfo(address from)
        external
        view
        returns (uint256,uint256,uint256,uint256)
    {
        uint256 pID = _pIDxAddr[from]; // Beosin (Chengdu LianAn) // Get the player ID of 'from'.
        if(_pID==0){
            return (0,0,0,0);
        }
```

```
        return (_plyr[pID].rreward,_plyr[pID].allReward,_plyr[pID].lv1Count,_plyr[pID].lv2Count);
    }
    // Beosin (Chengdu LianAn) // The function 'getTotalReferReward' is defined to get the total refer reward.
    function getTotalReferReward()
        external
        view
        returns (uint256)
    {
        return(_totalReferReward);
    }
    // Beosin (Chengdu LianAn) // The function 'getRegistrationFee' is defined to get the registration fee.
    function getRegistrationFee()
        external
        view
        returns (uint256)
    {
        if( _totalRegisterCount <_registrationStep || _registrationStep == 0){
            return   registrationBaseFee;
        }
        else{
            uint256 step =   totalRegisterCount.div(  registrationStep);
            return   registrationBaseFee.add(step.mul(  stepFee));
        }
    }
}
```

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com