YHT: Gambling Ecology of Block chain

Abstract

YHT is a token of the gambling ecosystem on the Blockchain, similar to a warrant, the holder of which can enjoy the benefits of the entire YHT gambling ecosystem. YHT uses innovative gambling methods to distribute the tokens, and the distribution system is a lottery contract on Ethereum. During each issue the total amount of tokens is fixed, and the amount of tokens each player can gain is proportional to his/her betting amount. At present, the full-open-source token contract and lottery contract have been deployed to Ethereum[1], and open interface is reserved to enable more games to access the YHT ecosystem later. In the future, these will be deployed to other public chains[2] to extend the application boundary of YHT ecosystem.

1. Introduction

Gambling games have a long history, from Mesopotamia[3] at the third century BC to the present, there are a large number of people who are happy with it[4]. However, it often requires a reliable arbitration center to determine the profit and loss. Due to the involvement of interests, opaque actions, black box operations, and even roll-and-roll runs often occur. Using a decentralized blockchain system can effectively circumvent these risks, and open-source smart contracts can provide maximum transparency.

Accordingly, the YHT gambling ecology is emerging, which not only solves the long-running drawbacks through the blockchain smart contract, but also activate and connect the entire ecology via Tokenomics. When participating in the gambling, the player can obtain tokens via mining-game, and enjoy the benefits of the entire ecosystem by holding tokens.

2. Distribution of tokens

2.1 Lottery Contract

The YHT distribution system is a lottery contract with the following rules:

(1) Three draws per week, on Tuesday, Thursday and Saturday (UTC+0)[①];

(2) The winning percentage is proportional to the bet amount, and the only winner is randomly selected;

(3) The winner will win 41.77% of the prize pool;

(4) 18% of the prize pool will be left to the next prize pool;

(5) 39%[②]of the prize pool will be allocated to the token holder as dividends;

(6) 1.23% of the prize pool will be charged to maintain the contract operation[③] and the community development;

Random numbers are generated by the oraclize service[5]to ensure a high degree of security[6]。

2.2 YHT Contract

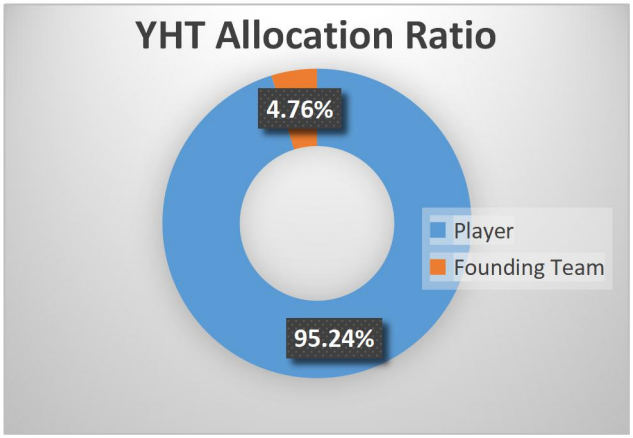
By betting each period of the lottery, you can get YHT, the rules are:

(1) A total of 271,828[④] YHT per period;

(2) At the end of each period, the total amount of YHT is reallocated according to the current bet ratio[⑤];

(3) Cutting the amount of YHT by half per 314 period[⑥];

(4) During each period, an additional 5% of the token is given to the founding team to support the future development of the YHT gambling ecosystem.



① Depending on the operation of Ethereum, there may be an advance or a delay.
② The referee's 5% return bonus will be deducted from this section, so it actually may be less than 39% and the reward will last up to 15 periods.
③ Using the oraclize timing trigger and random number generation service, the cost of core function to determine the jackpot winner is proportional to the number of bettors, so the gas consumed may be more.
④ The decimal part is omitted, and the actual integrity is 271828.182845904523536028.
⑤ Distribute the YHT first and then calculate the dividends.
⑥ When the amount of tokens produced during one period is less than 100, the YHT distribution will be closed.

Figure 1 YHT allocation strategy

3. Contract Open Interface

The YHT contract open interface is declared as follows.

```
Contract YHTOpenInterface {
   function transferExtraEarnings(address addr) external payable;
   function transferBonusEarnings() external payable returns(uint256);
}
```

transferExtraEarnings function: An external contract can save revenue to a specified user address by calling this function. For example, a gameplay can use this function to assign a "Grand Prize" award to the final winner.

transferBonusEarnings function: An external contract can assign revenue to all YHT holders by calling this function. For example, a gameplay method can distribute its cumulative rate income by calling this function.

Note: The lottery contract distributes the grand prize and dividends through these two interfaces.

4. Snapshot System

The number of addresses of the token holders will become huge, and finally the dividends distribution will become impossible due to the tremendous cost, so we developed a snapshot system that records the tokens information of each YHT holder at every award segment. Once the number of tokens changes, a snapshot will be taken. Through the snapshot, we can quickly calculate the total accumulated dividend income of the player. When the player withdraws the proceeds, the bonus will be calculated and combined. This allows the cost of dividends distribution to maintain at a constant level, so that a large amount and frequent dividends distribution is possible, and more games can be accessed in the future.

The core data structure and some of the algorithms are as follows:

```
struct BalanceSnapshot {
   uint256 balance;                    // balance of snapshot
   uint256 bonusIdBegin;               // begin of bonusId
   uint256 bonusIdEnd;                 // end of bonusId
```

```
}

struct User {
    ....
    ....
    BalanceSnapshot[] snapshots;    // the balance snapshot array
    uint256 snapshotsLength;            // the length of balance snapshot array
}

/**
  * @dev record a snapshot of balance
  * with the bonuses information can accurately calculate the earnings
  */
function balanceSnapshot(address addr, uint256 bonusRoundId) private {
    uint256 currentBalance = balances[addr];
    User storage user = users_[addr];
    if (user.snapshotsLength == 0) {
        user.snapshotsLength = 1;
        user.snapshots.push(BalanceSnapshot(currentBalance, bonusRoundId, 0));
    } else {
        BalanceSnapshot storage lastSnapshot = user.snapshots[user.snapshotsLength - 1];
        assert(lastSnapshot.bonusIdEnd == 0);

        // same as last record point just updated balance
        if (lastSnapshot.bonusIdBegin == bonusRoundId) {
            lastSnapshot.balance = currentBalance;
        } else {
            assert(lastSnapshot.bonusIdBegin < bonusRoundId);

            // if this snapshot is not the same as the last time, automatically merges part of the earnings
            if (bonusRoundId - lastSnapshot.bonusIdBegin < kAutoCombineBonusesCount) {
                uint256 amount = computeRoundBonuses(lastSnapshot.bonusIdBegin, bonusRoundId,
lastSnapshot.balance);
                user.bonusEarnings = user.bonusEarnings.add(amount);

                lastSnapshot.balance = currentBalance;
```

```
            lastSnapshot.bonusIdBegin = bonusRoundId;
            lastSnapshot.bonusIdEnd = 0;
        } else {
            lastSnapshot.bonusIdEnd = bonusRoundId;
            if (user.snapshotsLength == user.snapshots.length) {
                user.snapshots.length += 1;
            }
            user.snapshots[user.snapshotsLength++] = BalanceSnapshot(currentBalance, bonusRo
    undId, 0);
        }
    }
}


/**
 * @dev get bonuses
 * @param begin begin bonusId
 * @param end end bonusId
 * @param balance the balance in the round
 * Not use SafeMath, it is core loop, not use SafeMath will be saved 20% gas
 */
function computeRoundBonuses(uint256 begin, uint256 end, uint256 balance) view private
returns(uint256) {
    require(begin != 0);
    require(end != 0);

    uint256 amount = 0;
    while (begin < end) {
        uint256 value = balance * bonuses_[begin].payment / bonuses_[begin].currentTotalSupply;
        amount += value;
        ++begin;
    }
    return amount;
```

```
}

/**
* @dev compute snapshot bonuses
*/
function computeSnapshotBonuses(User storage user, uint256 lastBonusRoundId) view private returns(uint256) {
    uint256 amount = 0;
    uint256 length = user.snapshotsLength;
    for (uint256 i = 0; i < length; ++i) {
        uint256 value = computeRoundBonuses(
        user.snapshots[i].bonusIdBegin,
        i < length - 1 ? user.snapshots[i].bonusIdEnd : lastBonusRoundId,
        user.snapshots[i].balance);
        amount = amount.add(value);
    }
    return amount;
}
```

5. Development & Operation

The YHT community will use open source software to conduct iterative development and future operation by a decentralized collaborative method. It will not create a corporate entity and will not have a centralized office, therefore it can maintain extremely low development and operating costs. People who are interested in the program are welcome to participate us to build and improve the YHT community.

6. Conclusion

We build a decentralized gambling system on the blockchain and distribute YHT tokens during the betting to motivate and connect the game participants. Open interface of the contract is designed and reserved to enable more games to access the YHT gambling ecosystem. A snapshot

system is developed to solve the huge cost of dividends distribution brought by a large number of games in the future. The decentralized collaborative open source software is used for future development and operation of the community, and finally to build a continuous and stable blockchain gambling ecology.

Appendix: Contract Open Source Address

https://kovan.etherscan.io/address/0x31233391c9a776de54ca06ecb322f41dc5003db4#code

https://kovan.etherscan.io/address/0xce4b68cdec372fb998d15923282772ae0ba5fc57#code

Reference

[1] Vitalik Buterin. Ethereum: a next generation smart contract and decentralized application platform (2013). URL {https://github.com/ethereum/wiki/wiki/White-Paper}, 2018.

[2] EOS.IO Technical White Paper v2. URL {https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md},2018.

[3] Schwartz, David (January 7, 2013). Roll The Bones: The History of Gambling. Winchester Books. ISBN 978-0615847788.

[4] Wikipedia contributors. "Problem gambling." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 12 Jul. 2018. Web. 25 Aug. 2018.

[5] Oraclize. The leading oracle service for smart contracts and blockchain applications. URL {https://docs.oraclize.it/#home}, 2018.

[6] Oraclize. A Scalable Architecture for On-Demand, Untrusted Delivery of Entropy. URL {http://www.oraclize.it/papers/random_datasource-rev1.pdf}, 2018.