PARAMETERS

Line-98/99:
transferRateInitial/transferRate:

This rate is used for 2 major adjustments;

1- Rate is used to weight PERA holder balances
2- It is also used to weight the transaction amounts for holders. (If the message sender is a holder or if the message receiver is a holder or both, in these cases, transferRate is also used as a multiplier for the transacted amount.)

transferRate is chosen as big as possible since this rate is updated to a smaller number after each transaction. To avoid this rate to go below 1 (since solidity can not handle decimals), its initial value is chosen as 2^240. Details of how transferRate is used will be detailed within the document.

Line-100:
datumIndexLP:

Datum points indicate the points where a liquidity provider stakes or unstakes his UNI-V2 tokens. Whenever a liquidity providers stakes/unstakes, smart contract creates a new datum point.

Line-101:
totalStakedLP:

Total amount of UNI-V2 tokens staked on the PERA smart contract.

Line-102:
dailyRewardForTC:

Daily PERA emission rewards for the trading competition. This amount is added to trading competition reward pool every day for 10 years. After 10 years, smart contract stops releasing emission rewards.

Line-103:
totalTCwinners:

Number of users who are eligible to win the daily trading competition. After each transaction, contract sorts top-10 volume generators of the day and at the end of the day, winners of the daily trading competition will be able to claim their rewards.

Line-104:
decimalLossLP:

Since solidity can not handle decimals, while calculating the LP staker rewards, to avoid any possible reward losses for the LP token stakers, we first multiply the amount of rewards with this number (10^18) and when a user unstakes his LP tokens, calculated claimable reward is divided by the same number.

Line-115:
BlockSizeForTC:

PERA smart contract runs a trading competition each day. Start and end points of days are determined via using number of blocks. In Binance Smart Chain, a day includes about 28,800 blocks.

Line-118:
blockRewardLP:

PERA smart contract releases 0.5 PERA/Block for liquidity provider rewards. To get around the decimal issue, we first specify the reward value as 5 PERA/Block however when the user unstakes his LP tokens, amount of rewards that are earned via PERA emission is divided by 10 to reduce the rate to its actual value, 0.5 PERA/Block.

Line-120/121/122:
tradingCompFee/holderFee/liqproviderFee:

These are the transaction cut rates for each deflation reward. e.g. 50 represents a transaction cut of 0.5%.

Line-124:
lpTokenAddress:

When the PERA token liquidity is supplied to PancakeSwap AMM pool, the pool address should be manually introduced to the PERA smart contract so the contract knows the address of the pool whose UNI-V2 LP tokens will be staked.

Line-128:
struct UserLP:

UserLP stores 2 different information about depositing LP tokens. First one represents the staked UNI-V2 LP token amount for each user and the second one stores in which datum the LP token staker staked his LP tokens.

Line-133:
struct DatumpointlistLP:

Struct involves 3 variables that are related to LP staking. Each variable stores information about a specific datum point.

1- liqsum: Total amount of UNI-V2 LP tokens staked for a specific datum point.
2- prosum: Total amount of transaction cut rewards collected for a specific datum point.
3- block: Block number for each datum point. Variable stores the starting block of each datum point.

These values are used to calculate LP token stakers' pool shares and also how much reward they earned from each datum point.

Line-151/152:
userbalanceOf:

There are 83M PERA tokens in total. 73M is locked in the PERA smart contract and 10M is sent to the contract deployer. PERA smart contract releases 7.3M PERA each year for 10 years.

FUNCTIONS

Line-151/152:
function balanceOf:

There are 2 different ways of returning users balances. If a user is excluded as a holder, then the balance is represented as it is. However, if a user is not excluded, so the user is a holder, then the balance amount is divided by the current value of the transferRate before returning the balance value.

This is due to the fact that holder balances (users who are not excluded as a holder) are stored by multiplying the actual amount with the current value of the transferRate. That's why, in the balanceOf function, holder balances should be divided by the current value of the transferRate.

Line-179:
excludeAccount:

This function is used to exclude a wallet from the holders list. For example, contract deployer's wallet, PERA smart contract or PancakeSwap AMM pool addresses will be excluded from the holders list since we want only the holders to benefit from the transaction cut rewards.

While transferring an account from the holders list to an excluded account, function divides users balance by the current value of the transferRate. This is due to the same reason we have mentioned above; excluded accounts' balances are stored and represented as they are but holder balances are stored by multiplying the actual balance amount with the current value of the transferRate and divided by the same rate within the balanceOf function.

Line-186:
includeAccount:

This function is used to remove an account form the excluded accounts list. Function multiplies the user's balance with the current value of the transferRate to remove the account from the excluded list.

Line-199:
_isExcluded:

Checks whether a given wallet address is excluded or not.

Line-199:
_transfer:

Line-211:
First if case checks whether the message sender is an excluded account or not. Balance requirement for excluded accounts and holders differ from each other. If the message sender is an excluded account, than the following case should be satisfied: userbalanceOf[_from] >= _value, if the sender is a holder, than the following case should be satisfied: userbalanceOf[_from].div(transferRate) >= _value.

Line-220:
_bnum represents how many days have passed since the deployment of the smart contract. _bnum values start from zero and increases consecutively for each day. For example, for the first day _bnum = 0, for the second day _bnum=1.

Line-222:
totalRewardforTC[_bnum]: Variable cumulatively stores daily trading competition rewards coming from the 0.5% transaction cuts.

Line-223:
dlistLP[datumIndexLP].prosum: Variable cumulatively stores liquidity provider transaction cut rewards for each datum point. When a new datum point is created, following transaction cut rewards are stored within the corresponding datum point.

Line-227:
If condition checks whether the message sender and the receiver are excluded accounts or not. The variable transferRate that we used for weighting the holder balances is also used to weight the transacted amounts for holders.

If the message sender is a holder, than the transacted amount is subtracted from the holders balance by multiplying the transacted amount with the current value of the transferRate.

If the receiver is a holder, than the transacted amount is added to the holder’s balance after multiplying the transacted amount with the current value of the transferRate.

Line-248:
Message sender’s address and the transacted amount are sent to tradingComp function.

If the message sender is an excluded account, then the receiver’s address and the transacted amount is sent to tradingComp function.

The reason behind the above rule is as follows:

When a user sells PERA, it means using the transfer function with the input parameters;
msg.sender = User who sells PERA and msg.receiver = PancakeSwap AMM-pool

In this case, since the message sender is a holder, user can participate in the trading competition.

In the case of a buy then the transfer function parameters will be:
msg.sender = PancakeSwap AMM-pool and msg.receiver = User who buys PERA.

Since we want PERA buyers to be also included in the trading competition, if the message sender is an excluded account (as in the case of buying PERA from the PancakeSwap AMM-pool), then the msg.receiver is added to the trading competition.

Line-255:
The calculation below shows how the transferRate is updated after each transaction.

If condition checks if there is any holder or every existing account with PERA token is excluded. If there is no holder to distribute holder transaction cut rewards, then the transferRate value stays the same. In the case where there is a holder, then the update function of the transferRate is as follows.

tR= transferRate

$$tR_{new} = tR_{old} - tR_{old} \times \frac{\text{0.75\% of the transacted amount}}{\text{PERA total supply - Total amount of PERA in the excluded accounts}}$$

PERA total supply = 83 Million. Accounts to be excluded: PERA smart contract, PancakeSwap AMM-pool, and contract depoloyer.

This part is the essence of how holder transaction cut rewards are distributed automatically by weighting holder balances with the transferRate. To understand the logic, we will disintegrate the function into its components.

The denominator of the function (PERA total supply - Total amount of PERA in the excluded accounts) gives the amount of PERA that the holders have. So by dividing the amount of holder reward that is taken out from the current transaction to the total amount of PERA that the holders have gives the rate of how many PERA a user earns from the current transaction if he holds 1 PERA.

For example, if the holders have 1000 PERA in total and 0.75% of the current transaction is 10 PERA (so 10 PERA is taken out from the current transaction to be distributed to the holders) than this division gives us 10/1000 = 0.01 or 1%. This means that each holders balance should increase by 1%. 1% is also the rate between the tR_{new} and tR_{old}. The relation is as follows:

$$tR_{old} = tR_{new} + tR_{new} \times 1\%$$

So the previous value of the transferRate should be 1% bigger than the currently calculated rate of the transferRate. When the user balances are weighted with the current value of the transferRate, it automatically increases each holders balance by 1%.

Line-263:
_removeExcludedAmounts:

Calculates the total amount of PERA that the excluded accounts hold.

Line-273:
activeTraders:

Struct stores the addresses of daily trading competition participants and also stores the information whether the user (if he is a winner of the competition) claimed his daily trading competition reward or not.

Line-299:
nMixAddrandSpBlock:

This function allows us to store users wallet addresses and the day that they joined the trading competition in a gas efficient way. naddrHash function calculates *mod* of the user’s wallet address and only uses 10 digits of the wallet address and then concatenates the result with the number of the day that the user joined the trading competition. By doing so, we combine two different information into a single one by reducing the number of digits to store and the resulting number is also unique to each user.

This function is used for storing each users transaction volumes, wallet addresses, and the day that they joined the trading competition.

Line-303:
tradingComp

If the transaction amount of a holder is greater than 30 PERA than the user is eligible to join the trading competition (if the user is an excluded account, than the user is not included to the competition).

Line-316:
sortTraders:

This is the part where each user is sorted according to their daily generated volume.

Logic is as follows:

First we create a list where each address value is 0x0000 and the volume generated by the 0x000 accounts are also initially set zero. When a user generates volume, the contract compares the user's daily generated volume with the total volume generated by the 10th rank of the competition. If the user is able to make it to the top-10 then the contract compares the user's volume with the 1st rank's volume. If the user's volume is lower than the 1st rank's volume, than it compares it with the 2nd rank's volume. It goes like this until the contract finds the rank that the user has more generated volume. Than the contract replaces the user to the corresponding rank and shifts all the remaining ranks 1 below.

Line-359:
calculateUserTCreward:

Function calculates each daily trading competition winners' claimable rewards by using 3 different parameters.

1- How many transaction cut rewards are collected for that specific day and whether the trading competition emission rewards are still released or not. (After 10 years from the deployment of the contract, it stops releasing new PERA tokens.)

2- User's trading competition ranking (1st place claims 19% of the total trading competition rewards. 2nd place claims 17% and the rate goes down by 2% for each rank reduction.)

3- How many days it passed since the user win the daily trading competition. (If the user claims his rewards right after the competition day then he can claim 51% of his total claimable rewards. Waiting for another day adds 7% and user can claim 58% of his claimable rewards. The rate goes up by 7% after each day and user can claim all of his claimable tokens after waiting for 7 or more days. The amount that the contract cuts from the early-claimed rewards is added to the liquidity provider rewards.)

Line-463:
depositeLPtoken:

When a user stakes UNI-V2 LP tokens, contract creates a new datum point and stores the information of how many tokens the user has staked, in which datum point he started staking and the block number of the last datum point.

Line-481:
LpcutRewards:

Function runs through all the datum points where the user has been in staking. It calculates user's pool share for each datum point and how many rewards have been collected for each datum point. It adds up user's claimable rewards from each datum point and returns user's total rewards that he earned from transaction cuts.

Line-495:
LpemissionRewards:

Function calculates for how many blocks the user has been in staking. Function uses the information of in which datum point the user started staking and the current point. It calculates the difference between the block number when the user staked and the current block. Function runs through each datum point in which the user has been in staking and adds up his emission rewards from each datum point according to his pool share in each datum point.

PERA smart contract stops releasing emission rewards for UNI-V2 LP token stakers after 10 years. Function also checks if a datum point covers both a time interval where there were emission rewards and a time interval where emission reward distribution has been stopped. Emission reward calculations are made by considering such a case.

Line-525:
removeLiqudityLP:

Function is used for unstaking UNI-V2 LP tokens. If a user remains in staking less than a week, than the contract sends 96% of the user's LP tokens and 4% remains in the PERA smart contract and can not be reached by anyone.

Line-554:
addLPToken:

Function can only be used by the contract deployer and is used for introducing the PancakeSwap AMM-pool address to the PERA smart contract.