0.7

Alpaca Finance Process Quality Review

Score: 82%

Overview

This is an Alpaca Finance Process Quality Review completed on 04/10/2021. It was performed using the Process Review process (version 0.7.3) and is documented here. The review was performed by Nick of DeFiSafety. Check out our Telegram.

The final score of the review is **82%**, a **PASS**. The breakdown of the scoring is in Scoring Appendix. For our purposes, a pass is **70%.**

Summary of the Process

Very simply, the review looks for the following declarations from the developer's site. With these declarations, it is reasonable to trust the smart contracts.

- **Here are my smart contracts on the blockchain**
- **Here is the documentation that explains what my smart contracts do**
- **Here are the tests I ran to verify my smart contract**
- **Here are the audit(s) performed on my code by third party experts**
- **Here are the admin controls and strategies**

Disclaimer

This report is for informational purposes only and does not constitute investment advice of any kind, nor does it constitute an offer to provide investment advisory or other services. Nothing in this report shall be considered a solicitation or offer to buy or sell any security, token, future, option or other financial instrument or to offer or provide any investment advice or service to any person in any jurisdiction. Nothing contained in this report constitutes investment advice or offers any opinion with respect to the suitability of any security, and the views expressed in this report should not be taken as advice to buy, sell or hold any security. The information in this report should not be relied upon for the purpose of investing. In preparing the information contained in this report, we have not taken into account the investment needs, objectives and financial circumstances of any particular investor. This information has no regard to the specific investment objectives, financial situation and particular needs of any specific recipient of this information and investments discussed may not be suitable for all investors.

Any views expressed in this report by us were prepared based upon the information available to us at the time such views were written. The views expressed within this report are limited to DeFiSafety and the author and do not reflect those of any additional or third party and are strictly based upon DeFiSafety, its authors, interpretations and evaluation of relevant data. Changed or additional information could cause such

views to change. All information is subject to possible correction. Information may quickly become unreliable for various reasons, including changes in market conditions or economic circumstances.

Chain

This section indicates the blockchain used by this protocol.

> **Chain:** Binance Smart Chain

Guidance:

Ethereum
Binance Smart Chain
Polygon
Avalanche
Terra
Celo
Arbitrum
Solana

Code and Team

This section looks at the code deployed on the Mainnet that gets reviewed and its corresponding software repository. The document explaining these questions is here. This review will answer the following questions:

1) Are the executing code addresses readily available? (%)
2) Is the code actively being used? (%)
3) Is there a public software repository? (Y/N)
4) Is there a development history visible? (%)
5) Is the team public (not anonymous)? (Y/N)

1) Are the executing code addresses readily available? (%)

> **Answer:** 100%

They are available at website https://docs.alpacafinance.org/transparency#contract-documentation, as indicated in the Appendix.

Guidance:

100% Clearly labelled and on website, docs or repo, quick to find
70% Clearly labelled and on website, docs or repo but takes a bit of looking
40% Addresses in mainnet.json, in discord or sub graph, etc
20% Address found but labeling not clear or easy to find
0% Executing addresses could not be found

2) Is the code actively being used? (%)

> **Answer:** 100%

Activity exceeds 300 transactions a day on contract *Grazing Range* as indicated in the Appendix.

Guidance:

100% More than 10 transactions a day
70% More than 10 transactions a week
40% More than 10 transactions a month
10% Less than 10 transactions a month
0% No activity

3) Is there a public software repository? (Y/N)

> **Answer:** Yes

GitHub: https://github.com/alpaca-finance

4) Is there a development history visible? (%)

> **Answer:** 100%

At 133 branches and 884 commits, Alpaca's development history is as moving as it is inspirational.

This metric checks if the software repository demonstrates a strong steady history. This is normally demonstrated by commits, branches and releases in a software repository. A healthy history demonstrates a history of more than a month (at a minimum).

Guidance:

100% Any one of 100+ commits, 10+branches
70% Any one of 70+ commits, 7+branches
50% Any one of 50+ commits, 5+branches

30% Any one of 30+ commits, 3+branches
0% Less than 2 branches or less than 30 commits

5) Is the team public (not anonymous)? (Y/N)

> **Answer:** Yes

A lot of key members of Alpaca Finance are public. Info can be seen at https://www.binance.org/en/blog/bsc-project-spotlight-alpaca/.

Documentation

This section looks at the software documentation. The document explaining these questions is here.

Required questions are;

6) Is there a whitepaper? (Y/N)
7) Are the basic software functions documented? (Y/N)
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)
10) Is it possible to trace from software documentation to the implementation in code (%)

6) Is there a whitepaper? (Y/N)

> **Answer:** Yes

Location: https://docs.alpacafinance.org/

7) Are the basic software functions documented? (Y/N)

> **Answer:** Yes

Basic software functions are covered here.

8) Does the software function documentation fully (100%) cover the deployed contracts? (%)

> **Answer:** 80%

The most important contracts are covered in their documentation, though there are certainly uncovered deployed contracts.

Guidance:

100% All contracts and functions documented
80% Only the major functions documented
79-1% Estimate of the level of software documentation
0% No software documentation

How to improve this score:

This score can be improved by adding content to the software functions document such that it comprehensively covers the requirements. For guidance, refer to the SecurEth System Description Document. Using tools that aid traceability detection will help.

9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)

> **Answer:** 29%

Code examples are in the Appendix. As per the SLOC, there is 29% commenting to code (CtC).

The Comments to Code (CtC) ratio is the primary metric for this score.

Guidance:

100% CtC > 100 Useful comments consistently on all code
90-70% CtC > 70 Useful comment on most code
60-20% CtC > 20 Some useful commenting
0% CtC < 20 No useful commenting

How to improve this score

This score can improve by adding comments to the deployed code such that it comprehensively covers the code. For guidance, refer to the SecurEth Software Requirements.

10) Is it possible to trace from software documentation to the implementation in code (%)

> **Answer:** 90%

There is explicit traceability for all of the most important contracts deployed by Alpaca.

Guidance:

100% Clear explicit traceability between code and documentation at a requirement level for all code

60% Clear association between code and documents via non explicit traceability
40% Documentation lists all the functions and describes their functions
0% No connection between documentation and code

How to improve this score:

This score can improve by adding traceability from documentation to code such that it is clear where each outlined function is coded in the source code. For reference, check the SecurEth guidelines on traceability.

Testing

This section looks at the software testing available. It is explained in this document. This section answers the following questions;

11) Full test suite (Covers all the deployed code) (%)
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)
13) Scripts and instructions to run the tests (Y/N)
14) Report of the results (%)
15) Formal Verification test done (%)
16) Stress Testing environment (%)

11) Is there a Full test suite? (%)

> **Answer:** 100%

Code examples are in the Appendix. As per the SLOC, there is 419% testing to code (TtC).

This score is guided by the Test to Code ratio (TtC). Generally a good test to code ratio is over 100%. However the reviewers best judgement is the final deciding factor.

Guidance:

100% TtC > 120% Both unit and system test visible
80% TtC > 80% Both unit and system test visible
40% TtC < 80% Some tests visible
0% No tests obvious

12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)

> **Answer:** 50%

No code coverage test was found but evidence of extensive testing is clear.

Guidance:

100%	Documented full coverage
99-51%	Value of test coverage from documented results
50%	No indication of code coverage but clearly there is a reasonably complete set of tests
30%	Some tests evident but not complete
0%	No test for coverage seen

How to improve this score:

This score can improved by adding tests that achieve full code coverage. A clear report and scripts in the software repository will guarantee a high score.

13) Scripts and instructions to run the tests (Y/N)

> ✓ **Answer:** Yes

Scripts/Instructions location: https://github.com/alpaca-finance/bsc-alpaca-contract#run-tests

14) Report of the results (%)

> ⚠ **Answer:** 0%

No test report is evident.

Guidance:

100%	Detailed test report as described below
70%	GitHub code coverage report visible
0%	No test report evident

How to improve this score

Add a report with the results. The test scripts should generate the report or elements of it.

15) Formal Verification test done (%)

> ⚠ **Answer:** 0%

No formal verification testing was done.

16) Stress Testing environment (%)

> **Answer:** 100%

Alpaca finance is deployed to the BSC Testnet.

Security

This section looks at the 3rd party software audits done. It is explained in this document. This section answers the following questions;

17) Did 3rd Party audits take place? (%)
18) Is the bounty value acceptably high?

17) Did 3rd Party audits take place? (%)

> **Answer:** 100%

Multiple audits have reviewed Alpaca's code before and after its deployment.

Guidance:

100% Multiple Audits performed before deployment and results public and implemented or not required
90% Single audit performed before deployment and results public and implemented or not required
70% Audit(s) performed after deployment and no changes required. Audit report is public

50% Audit(s) performed after deployment and changes needed but not implemented
20% No audit performed
0% Audit Performed after deployment, existence is public, report is not public and no improvements deployed OR smart contract address' not found, (where question 1 is 0%)

Deduct 25% if code is in a private repo and no note from auditors that audit is applicable to deployed code

18) Is the bounty value acceptably high (%)

> **Answer:** 70%

Alpaca offers an active bug bounty of 100k at the highest tier.

Guidance:

100% Bounty is 10% TVL or at least $1M AND active program (see below)
90% Bounty is 5% TVL or at least 500k AND active program
80% Bounty is 5% TVL or at least 500k
70% Bounty is 100k or over AND active program
60% Bounty is 100k or over
50% Bounty is 50k or over AND active program
40% Bounty is 50k or over
20% Bug bounty program bounty is less than 50k
0% No bug bounty program offered

An active program means that a third party (such as Immunefi) is actively driving hackers to the site. An inactive program would be static mentions on the docs.

Access Controls

This section covers the documentation of special access controls for a DeFi protocol. The admin access controls are the contracts that allow updating contracts or coefficients in the protocol. Since these contracts can allow the protocol admins to "change the rules", complete disclosure of capabilities is vital for user's transparency. It is explained in this document. The questions this section asks are as follow;

19) Can a user clearly and quickly find the status of the admin controls?
20) Is the information clear and complete?
21) Is the information in non-technical terms that pertain to the investments?
22) Is there Pause Control documentation including records of tests?

19) Can a user clearly and quickly find the status of the access controls (%)

> **Answer:** 100%

https://docs.alpacafinance.org/governance/protocol-configurations

Guidance:

100% Clearly labelled and on website, docs or repo, quick to find
70% Clearly labelled and on website, docs or repo but takes a bit of looking
40% Access control docs in multiple places and not well labelled
20% Access control docs in multiple places and not labelled
0% Admin Control information could not be found

20) Is the information clear and complete (%)

> **Answer:** 90%

a) All contracts are clearly labelled as upgradeable (or not) -- 30%.
b) The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) -- 30% -- for now the admin team is clearly in charge of the contracts, though this will change through governance, as outlined in the docs.
c) The capabilities for change in the contracts are described -- 30%

Guidance:

All the contracts are immutable -- 100% OR

a) All contracts are clearly labelled as upgradeable (or not) -- 30% AND
b) The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) -- 30% AND
c) The capabilities for change in the contracts are described -- 30%

How to improve this score:

Create a document that covers the items described above. An example is enclosed.

21) Is the information in non-technical terms that pertain to the investments (%)

> ⓘ **Answer:** 30%

The documentation explains ownership & timelock in clear language, but the contract explanations are in software specific language.

Guidance:

100%	All the contracts are immutable
90%	Description relates to investments safety and updates in clear, complete non-software l language
30%	Description all in software specific language
0%	No admin control information could not be found

How to improve this score:

Create a document that covers the items described above in plain language that investors can understand. An example is enclosed.

22) Is there Pause Control documentation including records of tests (%)

> ⚠ **Answer:** 0%

No pause control documentation could be found.

Guidance:

100% All the contracts are immutable or no pause control needed and this is explained OR
100% Pause control(s) are clearly documented and there is records of at least one test within 3 months

80% Pause control(s) explained clearly but no evidence of regular tests
40% Pause controls mentioned with no detail on capability or tests
0% Pause control not documented or explained

How to improve this score**:**

Create a document that covers the items described above in plain language that investors can understand. An example is enclosed.

Appendices

Author Details

The author of this review is Rex of DeFi Safety.

Email : rex@defisafety.com Twitter : @defisafety

I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

I was coaxed to go to EthDenver 2018 and there I started SecuEth.org with Bryant and Roman. We created guidelines on good processes for blockchain code development. We got EthFoundation funding to assist in their development.

Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

DeFiSafety is my full time gig and we are working on funding vehicles for a permanent staff.

Scoring Appendix

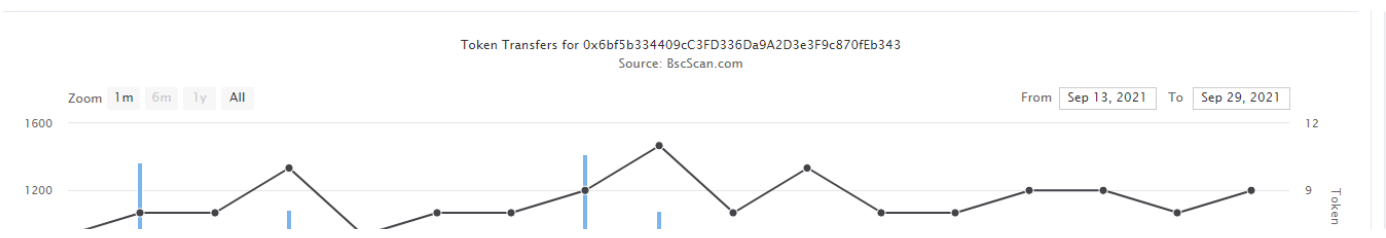
	Total	Alpaca Finance	
PQ Audit Scoring Matrix (v0.7)	**Points**	**Answer**	**Points**
Total	**260**		213.95
Code and Team			**82%**
1) Are the executing code addresses readily available? (%)	**20**	100%	20
2) Is the code actively being used? (%)	**5**	100%	5
3) Is there a public software repository? (Y/N)	**5**	y	5
4) Is there a development history visible? (%)	**5**	100%	5
5) Is the team public (not anonymous)? (Y/N)	**15**	y	15
Code Documentation			

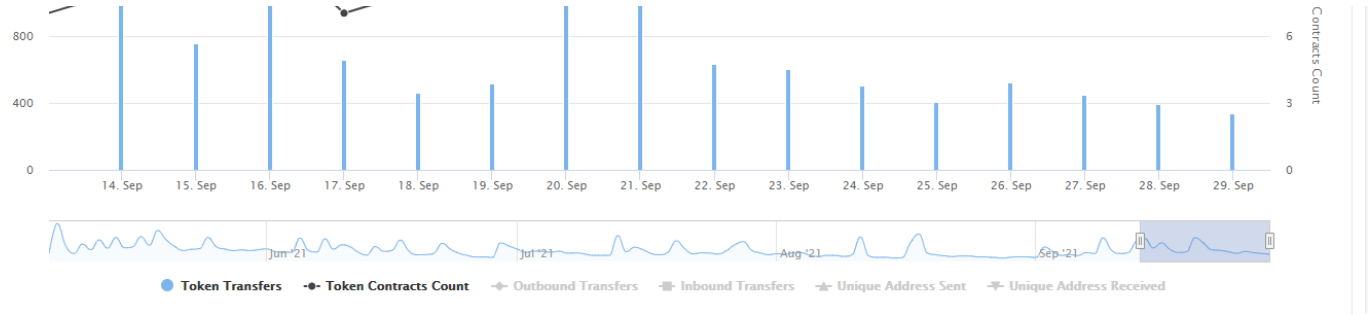
6) Is there a whitepaper? (Y/N)	5	Y	5
7) Are the basic software functions documented? (Y/N)	10	y	10
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)	15	80%	12
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)	5	29%	1.45
10) Is it possible to trace from software documentation to the implementation in code (%)	10	90%	9
Testing			
11) Full test suite (Covers all the deployed code) (%)	20	100%	20
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)	5	50%	2.5
13) Scripts and instructions to run the tests? (Y/N)	5	y	5
14) Report of the results (%)	10	0%	0
15) Formal Verification test done (%)	5	0%	0
16) Stress Testing environment (%)	5	100%	5
Security			
17) Did 3rd Party audits take place? (%)	70	100%	70
18) Is the bug bounty acceptable high? (%)	10	70%	7
Access Controls			
19) Can a user clearly and quickly find the status of the admin controls	5	100%	5
20) Is the information clear and complete	10	90%	9
21) Is the information in non-technical terms	10	30%	3
22) Is there Pause Control documentation including records of tests	10	0%	0
Section Scoring			
Code and Team	50	100%	
Documentation	45	83%	
Testing	50	65%	
Security	80	96%	
Access Controls	35	49%	

Executing Code Appendix


```
"ProxyAdmin": "0x5379F32C8D5F663EACb61eeF63F722950294f452",
"Timelock": "0x2D5408f2287BF9F9B05404794459a846651D0a59",
"Shield": "0x1963f84395C8cf464E5483dE7f2f434c3F1b4656",
"MultiCall": "0x41263cba59eb80dc200f3e2544eda4ed6a90e76c",
"MerkleDistributor": {
  "ITAM-week-1": "0xD193b38C87C8FAF7cB2F6C592d3ae885AE21c4bf",
  "ITAM-week-2": "0x63c60b840933e62E43D8eFc1dfDe31A32CDC4412",
  "ITAM-week-3": "0x083c022046df51F458709E1A5660B87754e58dE7",
  "ITAM-week-4": "0x60bc407144c82CB27Bf0681B457F0e7A75514105"
},
"GrazingRange": {
  "address": "0x6bf5b334409cC3FD336Da9A2D3e3F9c870fEb343",
  "deployedBlock": 7156513,
  "pools": [
```


Code Used Appendix





Example Code Appendix

```
pragma solidity 0.6.6;

import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";

import "./AlpacaToken.sol";
import "./interfaces/IFairLaunch.sol";

// FairLaunch is a smart contract for distributing ALPACA by asking user to stake the ERC2(
contract FairLaunch is IFairLaunch, Ownable, ReentrancyGuard {
  using SafeMath for uint256;
  using SafeERC20 for IERC20;

  // Info of each user.
  struct UserInfo {
    uint256 amount; // How many Staking tokens the user has provided.
    uint256 rewardDebt; // Reward debt. See explanation below.
    uint256 bonusDebt; // Last block that user exec something to the pool.
    address fundedBy; // Funded by who?
    //
    // We do some fancy math here. Basically, any point in time, the amount of ALPACAs
    // entitled to a user but is pending to be distributed is:
    //
    //   pending reward = (user.amount * pool.accAlpacaPerShare) - user.rewardDebt
    //
    // Whenever a user deposits or withdraws Staking tokens to a pool. Here's what happens
    //   1. The pool's `accAlpacaPerShare` (and `lastRewardBlock`) gets updated.
    //   2. User receives the pending reward sent to his/her address.
    //   3. User's `amount` gets updated.
    //   4. User's `rewardDebt` gets updated.
  }

  // Info of each pool.
  struct PoolInfo {
    address stakeToken; // Address of Staking token contract.
    uint256 allocPoint; // How many allocation points assigned to this pool. ALPACAs to dis
    uint256 lastRewardBlock; // Last block number that ALPACAs distribution occurs.
    uint256 accAlpacaPerShare; // Accumulated ALPACAs per share, times 1e12. See below.
    uint256 accAlpacaPerShareTilBonusEnd; // Accumated ALPACAs per share until Bonus End.
  }

```

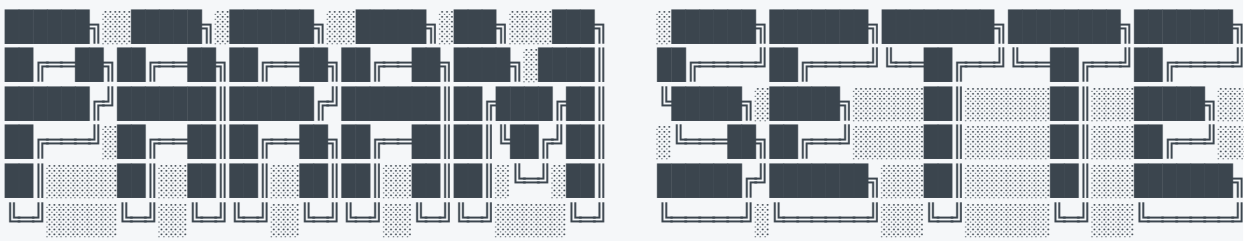
```
  // The Alpaca TOKEN!
  AlpacaToken public alpaca;
  // Dev address.
  address public devaddr;
  // ALPACA tokens created per block.
  uint256 public alpacaPerBlock;
  // Bonus muliplier for early alpaca makers.
  uint256 public bonusMultiplier;
  // Block number when bonus ALPACA period ends.
  uint256 public bonusEndBlock;
  // Bonus lock-up in BPS
  uint256 public bonusLockUpBps;

  // Info of each pool.
  PoolInfo[] public poolInfo;
  // Info of each user that stakes Staking tokens.
  mapping(uint256 => mapping(address => UserInfo)) public userInfo;
  // Total allocation poitns. Must be the sum of all allocation points in all pools.
  uint256 public totalAllocPoint;
  // The block number when ALPACA mining starts.
  uint256 public startBlock;

  event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
  event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
  event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

  constructor(
    AlpacaToken _alpaca,
    address _devaddr,
    uint256 _alpacaPerBlock,
    uint256 _startBlock,
    uint256 _bonusLockupBps,
    uint256 _bonusEndBlock
  ) public {
    bonusMultiplier = 0;
    totalAllocPoint = 0;
    alpaca = _alpaca;
    devaddr = _devaddr;
    alpacaPerBlock = _alpacaPerBlock;
    bonusLockUpBps = _bonusLockupBps;
    bonusEndBlock = _bonusEndBlock;
    startBlock = _startBlock;
  }

  /*
```

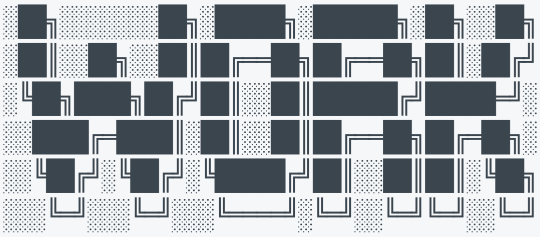


```
  */
```

```
94
95  // Update dev address by the previous dev.
96  function setDev(address _devaddr) public {
97    require(msg.sender == devaddr, "dev: wut?");
98    devaddr = _devaddr;
99  }
100
101 function setAlpacaPerBlock(uint256 _alpacaPerBlock) external onlyOwner {
102   alpacaPerBlock = _alpacaPerBlock;
103 }
104
105 // Set Bonus params. bonus will start to accu on the next block that this function execu
106 // See the calculation and counting in test file.
107 function setBonus(
108   uint256 _bonusMultiplier,
109   uint256 _bonusEndBlock,
110   uint256 _bonusLockUpBps
111 ) external onlyOwner {
112   require(_bonusEndBlock > block.number, "setBonus: bad bonusEndBlock");
113   require(_bonusMultiplier > 1, "setBonus: bad bonusMultiplier");
114   bonusMultiplier = _bonusMultiplier;
115   bonusEndBlock = _bonusEndBlock;
116   bonusLockUpBps = _bonusLockUpBps;
117 }
118
119 // Add a new lp to the pool. Can only be called by the owner.
120 function addPool(
121   uint256 _allocPoint,
122   address _stakeToken,
123   bool _withUpdate
124 ) external override onlyOwner {
125   if (_withUpdate) {
126     massUpdatePools();
127   }
128   require(_stakeToken != address(0), "add: not stakeToken addr");
129   require(!isDuplicatedPool(_stakeToken), "add: stakeToken dup");
130   uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
131   totalAllocPoint = totalAllocPoint.add(_allocPoint);
132   poolInfo.push(
133     PoolInfo({
134       stakeToken: _stakeToken,
135       allocPoint: _allocPoint,
136       lastRewardBlock: lastRewardBlock,
137       accAlpacaPerShare: 0,
138       accAlpacaPerShareTilBonusEnd: 0
139     })
140   );
141 }
142
143 // Update the given pool's ALPACA allocation point. Can only be called by the owner.
144 function setPool(
145   uint256 _pid,
```

```
    uint256 _allocPoint,
    bool /* _withUpdate */
  ) external override onlyOwner {
    massUpdatePools();
    totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
    poolInfo[_pid].allocPoint = _allocPoint;
  }

  /*
```



```
  */

  function isDuplicatedPool(address _stakeToken) public view returns (bool) {
    uint256 length = poolInfo.length;
    for (uint256 _pid = 0; _pid < length; _pid++) {
      if(poolInfo[_pid].stakeToken == _stakeToken) return true;
    }
    return false;
  }

  function poolLength() external override view returns (uint256) {
    return poolInfo.length;
  }

  function manualMint(address _to, uint256 _amount) external onlyOwner {
    alpaca.manualMint(_to, _amount);
  }

  // Return reward multiplier over the given _from to _to block.
  function getMultiplier(uint256 _lastRewardBlock, uint256 _currentBlock) public view retu
    if (_currentBlock <= bonusEndBlock) {
      return _currentBlock.sub(_lastRewardBlock).mul(bonusMultiplier);
    }
    if (_lastRewardBlock >= bonusEndBlock) {
      return _currentBlock.sub(_lastRewardBlock);
    }
    // This is the case where bonusEndBlock is in the middle of _lastRewardBlock and _curr
    return bonusEndBlock.sub(_lastRewardBlock).mul(bonusMultiplier).add(_currentBlock.sub(
  }

  // View function to see pending ALPACAs on frontend.
  function pendingAlpaca(uint256 _pid, address _user) external override view returns (uint
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];
    uint256 accAlpacaPerShare = pool.accAlpacaPerShare;
    uint256 lpSupply = IERC20(pool.stakeToken).balanceOf(address(this));
    if (block.number > pool.lastRewardBlock && lpSupply != 0) {
      uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
```

```
      uint256 alpacaReward = multiplier.mul(alpacaPerBlock).mul(pool.allocPoint).div(totalA
      accAlpacaPerShare = accAlpacaPerShare.add(alpacaReward.mul(1e12).div(lpSupply));
    }
    return user.amount.mul(accAlpacaPerShare).div(1e12).sub(user.rewardDebt);
  }

  // Update reward vairables for all pools. Be careful of gas spending!
  function massUpdatePools() public {
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
      updatePool(pid);
    }
  }

  // Update reward variables of the given pool to be up-to-date.
  function updatePool(uint256 _pid) public override {
    PoolInfo storage pool = poolInfo[_pid];
    if (block.number <= pool.lastRewardBlock) {
      return;
    }
    uint256 lpSupply = IERC20(pool.stakeToken).balanceOf(address(this));
    if (lpSupply == 0) {
      pool.lastRewardBlock = block.number;
      return;
    }
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	12	2222	245	448	1529	169

Comments to Code 448/1529 = 29%

TypeScript Tests

Language	Files	Lines	Blanks	Comments	Code	Complex
TypeScript	17	8160	894	864	6402	110

Tests to Code 6402/1529 = 419%