PoolTogether Process Quality Review

Score: 91%

This is a Process Quality Review of PoolTogether completed on April 13, 2021. It was performed using the Process Review process (version 0.6.2) and is documented here. The review was performed by Lucas of DeFiSafety. Check out our Telegram.

The final score of the review is 91%, an excellent score. 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**

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.

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 questions;

1. Are the executing code addresses readily available? (Y/N)
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)

Are the executing code addresses readily available? (Y/N)

> Answer: Yes

They are available at https://docs.pooltogether.com/resources-1/networks

/ethereum as indicated in the Appendix.

Is the code actively being used? (%)

> Answer: 100%

Activity is 25 transactions a day on contract *CompoundPrizePool.sol*, as indicated in the Appendix.

Percentage Score 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

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

> Answer: Yes

GitHub: https://github.com/pooltogether

Is there a public software repository with the code at a minimum, but normally test and scripts also (Y/N). Even if the repo was created just to hold the files and has just 1 transaction, it gets a Yes. For teams with private repos, this answer is No.

Is there a development history visible? (%)

> Answer: 100%

With 798 commits and 19 branches, this is clearly a well-maintained repository.

This 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 10 commits

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

> Answer: Yes

Some of the protocol members are clearly listed in their medium articles.

For a yes in this question the real names of some team members must be public on the website or other documentation. If the team is anonymous and then this question is a No.

Documentation

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

Required questions are;

1. Is there a whitepaper? (Y/N)
2. Are the basic software functions documented? (Y/N)
3. Does the software function documentation fully (100%) cover the deployed contracts? (%)
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)
5. Is it possible to trace from software documentation to the implementation in codee (%)

Is there a whitepaper? (Y/N)

> Answer: Yes

Location: https://docs.pooltogether.com/

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

> Answer: Yes

Location: https://docs.pooltogether.com/protocol/overview

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

> Answer: 100%

With Robust and well-organized function documentation, PoolTogther has put together impressive documentation.

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

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

Answer: 31%

Code examples are in the Appendix. As per the SLOC, there is 31% 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.

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

Answer: 100%

There is clear explicit tracability between the code and the documentation at a requirement level for all code.

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 requirements to code such that it is clear where each requirement is coded. 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;

1. Full test suite (Covers all the deployed code) (%)
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)
3. Scripts and instructions to run the tests (Y/N)
4. Packaged with the deployed code (Y/N)
5. Report of the results (%)
6. Formal Verification test done (%)
7. Stress Testing environment (%)

Is there a Full test suite? (%)

Answer: 100%

With a TtC of 170%, there is clearly a robust test suite present.

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

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

Answer: 92%

Location: https://coveralls.io/github/pooltogether/pooltogether-pool-contracts?branch=master

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 improve by adding tests achieving full code coverage. A clear report and scripts in the software repository will guarantee a high score.

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

Answer: Yes

Location: https://github.com/pooltogether/pooltogether-pool-contracts

How to improve this score

Add the scripts to the repository and ensure they work. Ask an outsider to create the environment and run the tests. Improve the scripts and docs based on their feedback.

Packaged with the deployed code (Y/N)

Answer: Yes

How to improve this score

Improving this score requires redeployment of the code, with the tests. This score gives credit to those who test their code before deployment and release them together. If a developer adds tests after deployment they can gain full points for all test elements except this one.

Report of the results (%)

Answer: 0%

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.

Formal Verification test done (%)

Answer: 0%

There is no evidence of formal verification testing done.

Stress Testing environment (%)

Answer: 100%

There is evidence of stress-testing on the Kovan Network.
https://docs.pooltogether.com/resources-1/networks/ethereum#kovan

Audits

Answer: 100%

PoolToghther has been audited by OpenZeppelin twice, with the last time being october 21st.

PoolTogether has also been audited by DitCraft.

PoolTogether was released January 7th.

Guidance:

1. Multiple Audits performed before deployment and results public and implemented or not required (100%)
2. Single audit performed before deployment and results public and implemented or not required (90%)

3. Audit(s) performed after deployment and no changes required. Audit report is public. (70%)
4. No audit performed (20%)
5. Audit Performed after deployment, existence is public, report is not public and no improvements deployed OR smart contract address' not found, question 1 (0%)

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	PoolTogether	
PQ Audit Scoring Matrix (v0.6)	**Points**	**Answer**	**Points**
Total	**240**		217.7
Code and Team			**91%**
1. Are the executing code addresses readily available? (Y/N)	**30**	Y	30
2. Is the code actively being used? (%)	**10**	100%	10
3. Is there a public software repository? (Y/N)	**5**	Y	5
4. Is there a development history visible? (%)	**5**	100%	5

Is the team public (not anonymous)? (Y/N)	20	Y	20
Code Documentation			
1. Is there a whitepaper? (Y/N)	5	Y	5
2. Are the basic software functions documented? (Y/N)	10	Y	10
3. Does the software function documentation fully (100%) cover the deployed contracts? (%)	15	100%	15
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	10	31%	3.1
5 Is it possible to trace from software documentation to the implementation in code (%)	5	100%	5
Testing			
1. Full test suite (Covers all the deployed code) (%)	20	100%	20
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)	5	92%	4.6
3. Scripts and instructions to run the tests? (Y/N)	5	Y	5
4. Packaged with the deployed code (Y/N)	5	Y	5
5. Report of the results (%)	10	0%	0
6. Formal Verification test done (%)	5	0%	0
7. Stress Testing environment (%)	5	100%	5
Audits			
Audit done	70	100%	70
Section Scoring			
Code and Team	70	100%	
Documentation	45	85%	
Testing	55	72%	
Audits	70	100%	

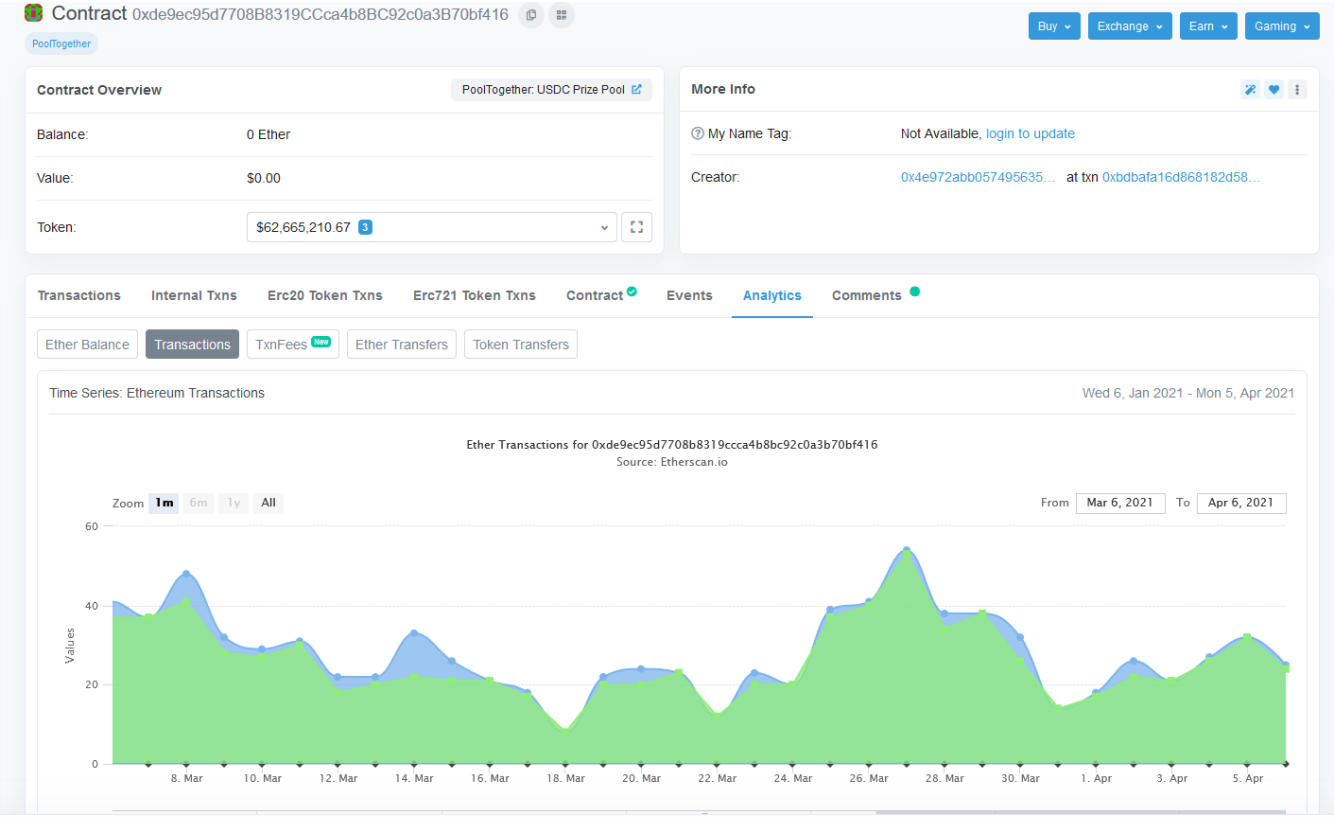
Executing Code Appendix

@pooltogether/current-pool-data ^3.3.2 npm

Contract	Address
Dai Prize Pool	0xEBfb47A7ad0FD6e57323C8A42B2E5A6a4F68fc1a
Dai Prize Strategy	0x178969A87a78597d303C47198c66F68E8be67Dc2
Dai POOL Faucet	0xF362ce295F2A4eaE4348fFC8cDBCe8d729ccb8Eb
UNI Prize Pool	0x0650d780292142835F6ac58dd8E2a336e87b4393
UNI Prize Strategy	0xe8726B85236a489a8E84C56c95790d07a368f913
UNI POOL Faucet	0xa5dddefD30e234Be2Ac6FC1a0364cFD337aa0f61
USDC Prize Pool	0xde9ec95d7708b8319ccca4b8bc92c0a3b70bf416
USDC Prize Strategy	0x3d9946190907ada8b70381b25c71eb9adf5f9b7b
USDC POOL Faucet	0xbd537257fad96e977b9e545be583bbf7028f30b9
COMP Prize Pool	0xBC82221e131c082336cf698F0cA3EBd18aFd4ce7
COMP Prize Strategy	0x3ec4694b65e41f12d6b5d5ba7c2341f4d6859773
COMP POOL Faucet	0x72F06a78bbAac0489067A1973B0Cef61841D58BC
POOL Prize Pool	0x396b4489da692788e327e2e4b2b0459a5ef26791
POOL Prize Strategy	0x21e5e62e0b6b59155110cd36f3f6655fbbcf6424
Loot Box ERC721	0x4d695c615a7AACf2d7b9C481B66045BB2457Dfde
Loot Box Prize Strategy Listener	0xfe7205DF55BA42c8801e44B55BF05F06cCe8565E

Reserve	0xdb8E47BEFe4646fCc62BE61EEE5DF350404c124F
Reserve Registry	0x3e8b9901dBFE766d3FE44B36c180A1bca2B9A295

Code Used Appendix



Example Code Appendix

```
1  // SPDX-License-Identifier: GPL-3.0
2
3  pragma solidity >=0.6.0 <0.7.0;
4
5  import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol
6  import "@openzeppelin/contracts-upgradeable/utils/SafeCastUpgradeable.sol
7  import "@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeal
8  import "@openzeppelin/contracts-upgradeable/token/ERC721/IERC721Upgradeab
9  import "@openzeppelin/contracts-upgradeable/introspection/ERC165CheckerUp
10 import "@openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeal
11 import "@pooltogether/fixed-point/contracts/FixedPoint.sol";
12
13 import "../external/compound/ICompLike.sol";
14 import "../registry/RegistryInterface.sol";
15 import "../reserve/ReserveInterface.sol";
```

```
16 import "../token/TokenListenerInterface.sol";
17 import "../token/TokenListenerLibrary.sol";
18 import "../token/ControlledToken.sol";
19 import "../token/TokenControllerInterface.sol";
20 import "../utils/MappedSinglyLinkedList.sol";
21 import "./PrizePoolInterface.sol";
22
23 /// @title Escrows assets and deposits them into a yield source.  Exposes
24 /// @notice Accounting is managed using Controlled Tokens, whose mint and
25 /// @dev Must be inherited to provide specific yield-bearing asset control
26 abstract contract PrizePool is PrizePoolInterface, OwnableUpgradeable, Re
27   using SafeMathUpgradeable for uint256;
28   using SafeCastUpgradeable for uint256;
29   using SafeERC20Upgradeable for IERC20Upgradeable;
30   using MappedSinglyLinkedList for MappedSinglyLinkedList.Mapping;
31   using ERC165CheckerUpgradeable for address;
32
33   /// @dev Emitted when an instance is initialized
34   event Initialized(
35     address reserveRegistry,
36     uint256 maxExitFeeMantissa,
37     uint256 maxTimelockDuration
38   );
39
40   /// @dev Event emitted when controlled token is added
41   event ControlledTokenAdded(
42     ControlledTokenInterface indexed token
43   );
44
45   /// @dev Emitted when reserve is captured.
46   event ReserveFeeCaptured(
47     uint256 amount
48   );
49
50   event AwardCaptured(
51     uint256 amount
52   );
53
54   /// @dev Event emitted when assets are deposited
55   event Deposited(
56     address indexed operator,
57     address indexed to,
58     address indexed token,
59     uint256 amount,
60     address referrer
61   );
62
63   /// @dev Event emitted when timelocked funds are re-deposited
64   event TimelockDeposited(
65     address indexed operator,
66     address indexed to,
67     address indexed token,
```

```
    uint256 amount
  );

  /// @dev Event emitted when interest is awarded to a winner
  event Awarded(
    address indexed winner,
    address indexed token,
    uint256 amount
  );

  /// @dev Event emitted when external ERC20s are awarded to a winner
  event AwardedExternalERC20(
    address indexed winner,
    address indexed token,
    uint256 amount
  );

  /// @dev Event emitted when external ERC20s are transferred out
  event TransferredExternalERC20(
    address indexed to,
    address indexed token,
    uint256 amount
  );

  /// @dev Event emitted when external ERC721s are awarded to a winner
  event AwardedExternalERC721(
    address indexed winner,
    address indexed token,
    uint256[] tokenIds
  );

  /// @dev Event emitted when assets are withdrawn instantly
  event InstantWithdrawal(
    address indexed operator,
    address indexed from,
    address indexed token,
    uint256 amount,
    uint256 redeemed,
    uint256 exitFee
  );

  /// @dev Event emitted upon a withdrawal with timelock
  event TimelockedWithdrawal(
    address indexed operator,
    address indexed from,
    address indexed token,
    uint256 amount,
    uint256 unlockTimestamp
  );

  event ReserveWithdrawal(
    address indexed to,
```

```
    uint256 amount
  );

  /// @dev Event emitted when timelocked funds are swept back to a user
  event TimelockedWithdrawalSwept(
    address indexed operator,
    address indexed from,
    uint256 amount,
    uint256 redeemed
  );

  /// @dev Event emitted when the Liquidity Cap is set
  event LiquidityCapSet(
    uint256 liquidityCap
  );

  /// @dev Event emitted when the Credit plan is set
  event CreditPlanSet(
    address token,
    uint128 creditLimitMantissa,
    uint128 creditRateMantissa
  );

  /// @dev Event emitted when the Prize Strategy is set
  event PrizeStrategySet(
    address indexed prizeStrategy
  );

  /// @dev Emitted when credit is minted
  event CreditMinted(
    address indexed user,
    address indexed token,
    uint256 amount
  );

  /// @dev Emitted when credit is burned
  event CreditBurned(
    address indexed user,
    address indexed token,
    uint256 amount
  );

  struct CreditPlan {
    uint128 creditLimitMantissa;
    uint128 creditRateMantissa;
  }

  struct CreditBalance {
    uint192 balance;
    uint32 timestamp;
    bool initialized;
  }
```

```
172
173   /// @dev Reserve to which reserve fees are sent
174   RegistryInterface public reserveRegistry;
175
176   /// @dev A linked list of all the controlled tokens
177   MappedSinglyLinkedList.Mapping internal _tokens;
178
179   /// @dev The Prize Strategy that this Prize Pool is bound to.
180   TokenListenerInterface public prizeStrategy;
181
182   /// @dev The maximum possible exit fee fraction as a fixed point 18 numb
183   /// For example, if the maxExitFeeMantissa is "0.1 ether", then the max
184   uint256 public maxExitFeeMantissa;
185
186   /// @dev The maximum possible timelock duration for a timelocked withdr
187   uint256 public maxTimelockDuration;
188
189   /// @dev The total funds that are timelocked.
190   uint256 public timelockTotalSupply;
191
192   /// @dev The total funds that have been allocated to the reserve
193   uint256 public reserveTotalSupply;
194
195   /// @dev The total amount of funds that the prize pool can hold.
196   uint256 public liquidityCap;
197
198   /// @dev the The awardable balance
199   uint256 internal _currentAwardBalance;
200
201   /// @dev The timelocked balances for each user
202   mapping(address => uint256) internal _timelockBalances;
203
204   /// @dev The unlock timestamps for each user
205   mapping(address => uint256) internal _unlockTimestamps;
206
207   /// @dev Stores the credit plan for each token.
208   mapping(address => CreditPlan) internal _tokenCreditPlans;
209
210   /// @dev Stores each users balance of credit per token.
211   mapping(address => mapping(address => CreditBalance)) internal _tokenCr
212
213   /// @notice Initializes the Prize Pool
214   /// @param _controlledTokens Array of ControlledTokens that are control
215   /// @param _maxExitFeeMantissa The maximum exit fee size
216   /// @param _maxTimelockDuration The maximum length of time the withdraw
217   function initialize (
218     RegistryInterface _reserveRegistry,
219     ControlledTokenInterface[] memory _controlledTokens,
220     uint256 _maxExitFeeMantissa,
221     uint256 _maxTimelockDuration
222   )
223     public
```

```
224     initializer
225   {
226     require(address(_reserveRegistry) != address(0), "PrizePool/reserveRe
227     _tokens.initialize();
228     for (uint256 i = 0; i < _controlledTokens.length; i++) {
229       _addControlledToken(_controlledTokens[i]);
230     }
231     __Ownable_init();
232     __ReentrancyGuard_init();
233     _setLiquidityCap(uint256(-1));
234
235     reserveRegistry = _reserveRegistry;
236     maxExitFeeMantissa = _maxExitFeeMantissa;
237     maxTimelockDuration = _maxTimelockDuration;
238
239     emit Initialized(
240       address(_reserveRegistry),
241       maxExitFeeMantissa,
242       maxTimelockDuration
243     );
244   }
245
246   /// @dev Returns the address of the underlying ERC20 asset
247   /// @return The address of the asset
248   function token() external override view returns (address) {
249     return address(_token());
250   }
251
252   /// @dev Returns the total underlying balance of all assets. This inclu
253   /// @return The underlying balance of assets
254   function balance() external returns (uint256) {
255     return _balance();
256   }
257
258   /// @dev Checks with the Prize Pool if a specific token type may be awar
259   /// @param _externalToken The address of the token to check
260   /// @return True if the token may be awarded, false otherwise
261   function canAwardExternal(address _externalToken) external view returns
262     return _canAwardExternal(_externalToken);
263   }
264
265   /// @notice Deposits timelocked tokens for a user back into the Prize P
266   /// @param to The address receiving the tokens
267   /// @param amount The amount of timelocked assets to re-deposit
268   /// @param controlledToken The type of token to be minted in exchange (
269   function timelockDepositTo(
270     address to,
271     uint256 amount,
272     address controlledToken
273   )
274     external
275     onlyControlledToken(controlledToken)
```

```
276     canAddLiquidity(amount)
277     nonReentrant
278   {
279     address operator = _msgSender();
280     _mint(to, amount, controlledToken, address(0));
281     _timelockBalances[operator] = _timelockBalances[operator].sub(amount)
282     timelockTotalSupply = timelockTotalSupply.sub(amount);
283
284     emit TimelockDeposited(operator, to, controlledToken, amount);
285   }
286
287   /// @notice Deposit assets into the Prize Pool in exchange for tokens
288   /// @param to The address receiving the newly minted tokens
289   /// @param amount The amount of assets to deposit
290   /// @param controlledToken The address of the type of token the user is
291   /// @param referrer The referrer of the deposit
292   function depositTo(
293     address to,
294     uint256 amount,
295     address controlledToken,
296     address referrer
297   )
298     external override
299     onlyControlledToken(controlledToken)
300     canAddLiquidity(amount)
301     nonReentrant
302   {
303     address operator = _msgSender();
304
305     _mint(to, amount, controlledToken, referrer);
306
307     _token().safeTransferFrom(operator, address(this), amount);
308     _supply(amount);
309
310     emit Deposited(operator, to, controlledToken, amount, referrer);
311   }
312
313   /// @notice Withdraw assets from the Prize Pool instantly.  A fairness
314   /// @param from The address to redeem tokens from.
315   /// @param amount The amount of tokens to redeem for assets.
316   /// @param controlledToken The address of the token to redeem (i.e. tic
317   /// @param maximumExitFee The maximum exit fee the caller is willing to
318   /// @return The actual exit fee paid
319   function withdrawInstantlyFrom(
320     address from,
321     uint256 amount,
322     address controlledToken,
323     uint256 maximumExitFee
324   )
325     external override
326     nonReentrant
327     onlyControlledToken(controlledToken)
```

```
328       returns (uint256)
329     {
330       (uint256 exitFee, uint256 burnedCredit) = _calculateEarlyExitFeeLessB
331       require(exitFee <= maximumExitFee, "PrizePool/exit-fee-exceeds-user-m
332
333       // burn the credit
334       _burnCredit(from, controlledToken, burnedCredit);
335
336       // burn the tickets
337       ControlledToken(controlledToken).controllerBurnFrom(_msgSender(), fro
338
339       // redeem the tickets less the fee
340       uint256 amountLessFee = amount.sub(exitFee);
341       uint256 redeemed = _redeem(amountLessFee);
342
343       _token().safeTransfer(from, redeemed);
344
345       emit InstantWithdrawal(_msgSender(), from, controlledToken, amount, r
346
347       return exitFee;
348     }
349
350     /// @notice Limits the exit fee to the maximum as hard-coded into the c
351     /// @param withdrawalAmount The amount that is attempting to be withdra
352     /// @param exitFee The exit fee to check against the limit
353     /// @return The passed exit fee if it is less than the maximum, otherwi
354     function _limitExitFee(uint256 withdrawalAmount, uint256 exitFee) inter
355       uint256 maxFee = FixedPoint.multiplyUintByMantissa(withdrawalAmount, 
356       if (exitFee > maxFee) {
357         exitFee = maxFee;
358       }
359       return exitFee;
360     }
361
362     /// @notice Withdraw assets from the Prize Pool by placing them into th
363     /// The timelock is used to ensure that the tickets have contributed th
364     /// @dev Note that if the user has previously timelocked funds then thi
365     /// If the existing timelocked funds are still locked, then the incomin
366     /// balance is added to their existing balance and the new timelock unl
367     /// @param from The address to withdraw from
368     /// @param amount The amount to withdraw
369     /// @param controlledToken The type of token being withdrawn
370     /// @return The timestamp from which the funds can be swept
371     function withdrawWithTimelockFrom(
372       address from,
373       uint256 amount,
374       address controlledToken
375     )
376       external override
377       nonReentrant
378       onlyControlledToken(controlledToken)
379       returns (uint256)
```

```
380   {
381     uint256 blockTime = _currentTime();
382     (uint256 lockDuration, uint256 burnedCredit) = _calculateTimelockDura
383     uint256 unlockTimestamp = blockTime.add(lockDuration);
384     _burnCredit(from, controlledToken, burnedCredit);
385     ControlledToken(controlledToken).controllerBurnFrom(_msgSender(), fro
386     _mintTimelock(from, amount, unlockTimestamp);
387     emit TimelockedWithdrawal(_msgSender(), from, controlledToken, amount
388
389     // return the block at which the funds will be available
390     return unlockTimestamp;
391   }
392
393   /// @notice Adds to a user's timelock balance.  It will attempt to swee
394   /// Note that this will overwrite the previous unlock timestamp.
395   /// @param user The user whose timelock balance should increase
396   /// @param amount The amount to increase by
397   /// @param timestamp The new unlock timestamp
398   function _mintTimelock(address user, uint256 amount, uint256 timestamp)
399     // Sweep the old balance, if any
400     address[] memory users = new address[](1);
401     users[0] = user;
402     _sweepTimelockBalances(users);
403
404     timelockTotalSupply = timelockTotalSupply.add(amount);
405     _timelockBalances[user] = _timelockBalances[user].add(amount);
406     _unlockTimestamps[user] = timestamp;
407
408     // if the funds should already be unlocked
409     if (timestamp <= _currentTime()) {
410       _sweepTimelockBalances(users);
411     }
412   }
413
414   /// @notice Updates the Prize Strategy when tokens are transferred betw
415   /// @param from The address the tokens are being transferred from (0 if
416   /// @param to The address the tokens are being transferred to (0 if bur
417   /// @param amount The amount of tokens being trasferred
418   function beforeTokenTransfer(address from, address to, uint256 amount)
419     if (from != address(0)) {
420       uint256 fromBeforeBalance = IERC20Upgradeable(msg.sender).balanceOf
421       // first accrue credit for their old balance
422       uint256 newCreditBalance = _calculateCreditBalance(from, msg.sender
423
424       if (from != to) {
425         // if they are sending funds to someone else, we need to limit th
426         newCreditBalance = _applyCreditLimit(msg.sender, fromBeforeBalanc
427       }
428
429       _updateCreditBalance(from, msg.sender, newCreditBalance);
430     }
431     if (to != address(0) && to != from) {
```

```
432         _accrueCredit(to, msg.sender, IERC20Upgradeable(msg.sender).balance
433       }
434       // if we aren't minting
435       if (from != address(0) && address(prizeStrategy) != address(0)) {
436         prizeStrategy.beforeTokenTransfer(from, to, amount, msg.sender);
437       }
438     }
439
440     /// @notice Returns the balance that is available to award.
441     /// @dev captureAwardBalance() should be called first
442     /// @return The total amount of assets to be awarded for the current pr
443     function awardBalance() external override view returns (uint256) {
444       return _currentAwardBalance;
445     }
446
447     /// @notice Captures any available interest as award balance.
448     /// @dev This function also captures the reserve fees.
449     /// @return The total amount of assets to be awarded for the current pr
450     function captureAwardBalance() external override nonReentrant returns (
451       uint256 tokenTotalSupply = _tokenTotalSupply();
452
453       // it's possible for the balance to be slightly less due to rounding
454       uint256 currentBalance = _balance();
455       uint256 totalInterest = (currentBalance > tokenTotalSupply) ? current
456       uint256 unaccountedPrizeBalance = (totalInterest > _currentAwardBalan
457
458       if (unaccountedPrizeBalance > 0) {
459         uint256 reserveFee = calculateReserveFee(unaccountedPrizeBalance);
460         if (reserveFee > 0) {
461           reserveTotalSupply = reserveTotalSupply.add(reserveFee);
462           unaccountedPrizeBalance = unaccountedPrizeBalance.sub(reserveFee)
463           emit ReserveFeeCaptured(reserveFee);
464         }
465         _currentAwardBalance = _currentAwardBalance.add(unaccountedPrizeBal
466
467         emit AwardCaptured(unaccountedPrizeBalance);
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	55	5899	901	1189	3809	338

Comments to Code 1189/3809 = 31%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	95	8883	1923	511	6449	127

Tests to Code 6449/ 3809 = 170%