0.7

Bagels Finance Process Quality Review

Score: 19%

Overview

This is a Bagels Finance Process Quality Review completed on October 5th 2021. It was performed using the Process Review process (version 0.7.3) and is documented here. The review was performed by Nic of DeFiSafety. Check out our Telegram.

The final score of the review is **46%**, a **FAIL**. 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 blockchains used by this protocol. This report covers all of the blockchains upon which the protocol is deployed.

> **Chain:** Ethereum, 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://github.com/bagels-dev/protocol-v1/blob/main/README.md, 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 is over 10 transactions a day on contract *Bank.sol*, 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/bagels-dev/protocol-v1.

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

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

> **Answer:** 0%

With 4 commits and 1 branch, this repository has a lack of development history.

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

How to improve this score:

Continue to test and perform other verification activities after deployment, including routine maintenance updating to new releases of testing and deployment tools. A public development history indicates clearly to the public the level of continued investment and activity by the developers on the application. This gives a level of security and faith in the application.

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

> ⚠ **Answer:** No

For a **"Yes"** in this question, the real names of some team members must be public on the website or other documentation (LinkedIn, etc). If the team is anonymous, then this question is a **"No"**.

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://bagelsofficial.gitbook.io/bagels-finance/.

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

> ⚠ **Answer:** No

There are no software functions documented in the Bagels Finance documentation.

How to improve this score:

Write the document based on the deployed code. For guidance, refer to the SecurEth System Description Document.

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

> ⚠ **Answer:** 0%

There are no software functions documented in the Bagels Finance 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

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:** 0%

As there are no software functions documented in the Bagels Finance documentation, we cannot evaluate their traceability as to their implementation in the protocol's source 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 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:** 0%

There is no testing suite in the Bagels Finance GitHub.

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

How to improve this score:

This score can improved by adding tests to fully cover the code. Document what is covered by traceability or test results in the software repository.

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

> ⚠ **Answer:** 0%

There is no evidence of any Bagels Finance code coverage.

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:** No

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.

14) Report of the results (%)

> ⚠ **Answer:** 0%

No test report was found.

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 Bagels Finance Formal Verification test was found.

16) Stress Testing environment (%)

⚠ **Answer:** 0%

There is no evidence of any Bagels Finance testnet smart contract usage.

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%

The existence of two audits were previously mentioned at https://bagelsofficial.gitbook.io/bagels-finance/faqs#have-your-contracts-been-audited, but the reports were not public.

The audits have since been published (after this report was released) - see the timestamp at the bottom of the GitBook. As such, we have updated the score given this new information.

Both audits were completed before deployment, and the issues have been resolved.

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%

The Bagels Finance Bug Bounty program rewards participating users with up to 200k.

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:** 70%

Governance information was found at https://bagelsofficial.gitbook.io/bagels-finance/bagels-finance-white-paper-as-of-sep-27-2021#10-0-dao-governance. It took a bit of looking with the search function. There is also additional DAO information at https://bagelsofficial.gitbook.io/bagels-finance/how-it-works/dao.

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:** 15%

a) No specific contracts are labelled as upgradeable or not.

b) Contract ownership is not specified.

c) Capabilities for change are vaguely described at https://bagelsofficial.gitbook.io/bagels-finance/how-it-works/dao - 15%

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:** 0%

There is a good amount of investment safety, written in user-friendly language, in the Bagels Finance documentation at https://bagelsofficial.gitbook.io/bagels-finance/risks. However, none of it relates to access controls.

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%

There is no information regarding Pause Control or any similar function in the Bagels Finance documentation.

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	Bagels Finance	
PQ Audit Scoring Matrix (v0.7)	**Points**	**Answer**	**Points**
Total	**260**		118.45
Code and Team			**46%**
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**	0%	0
5) Is the team public (not anonymous)? (Y/N)	**15**	n	0
Code Documentation			
6) Is there a whitepaper? (Y/N)	**5**	y	5
7) Are the basic software functions documented? (Y/N)	**10**	n	0
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)	**15**	0%	0
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**	0%	0
Testing			
11) Full test suite (Covers all the deployed code) (%)	**20**	0%	0
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)	**5**	0%	0
13) Scripts and instructions to run the tests? (Y/N)	**5**	0%	0
14) Report of the results (%)	**10**	0%	0
15) Formal Verification test done (%)	**5**	0%	0
16) Stress Testing environment (%)	**5**	0%	0
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**	70%	3.5
20) Is the information clear and complete	**10**	15%	1.5
21) Is the information in non-technical terms	**10**	0%	0
22) Is there Pause Control documentation including records of tests	**10**	0%	0
Section Scoring			
Code and Team	50	60%	

Documentation	45	14%	
Testing	50	0%	
Security	80	96%	
Access Controls	35	14%	

Executing Code Appendix

Bagel Protocol Ver 1.0

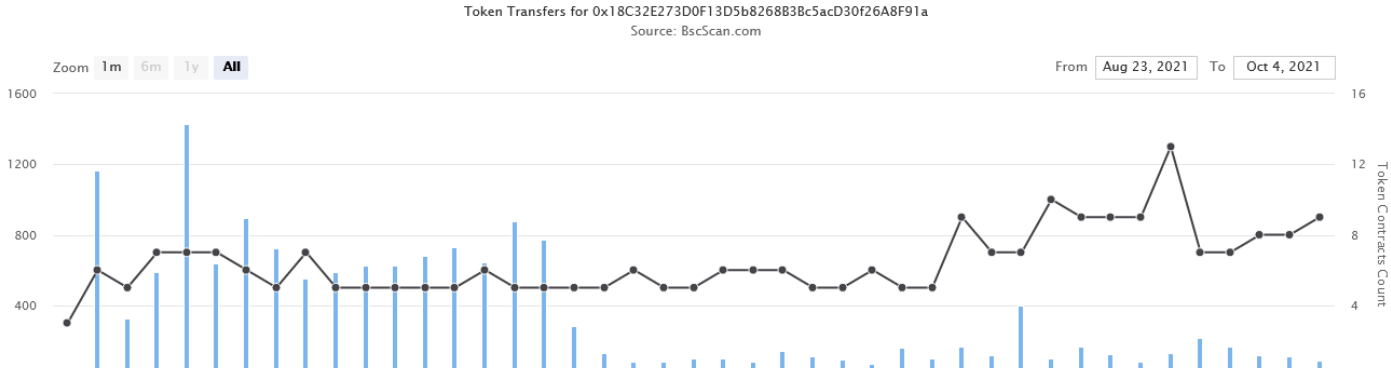
Token

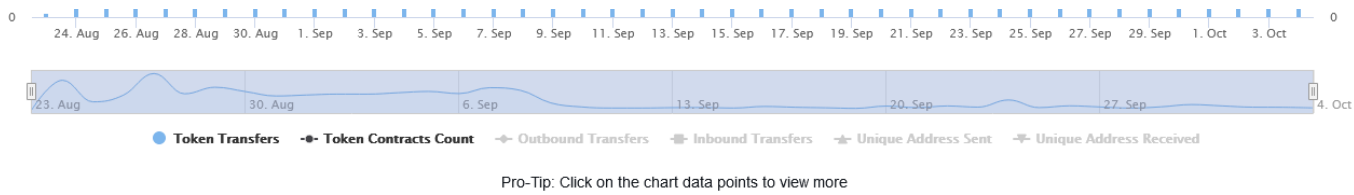
Name	Address
Bagel	0xBb238FcE6E2eE90781cD160C9C6eAf3a4CfAD801

Lend

Name	Address
Bank	0x18C32E273D0F13D5b8268B3Bc5acD30f26A8F91a
gMDX	0x36C61780246aE62Fb1F2E6Bc2bf20eC4b871a4aa
gUSDT	0x32999E0590A043aB75E3DEB298817ea8030407A5
gBUSD	0xe8D1Df138AC3118Ab3eaF5D4a2406e311F4c6253
gBNB	0xd373d80bF66738572F169385321ca388a21B779e
gBTCB	0x26c433AfD699Bd774C827be21Db2c0EBD33dcEC5
gETH	0x41cC1637d93246574bC745371Eaa2CAe2F3d98eF
gCAKE	0x85CF44015E02A919b65E7936be7df917F843Bd36
gUSDC	0x40eD7226b07ab46E64Ca775A498981E9C8270400

Code Used Appendix





Pro-Tip: Click on the chart data points to view more

Example Code Appendix

```
contract Bank is GTokenFactory, Ownable, ReentrancyGuard {
    using SafeToken for address;
    using SafeMath for uint256;

    event OpPosition(uint256 indexed id, uint256 debt, uint back);
    event Liquidate(uint256 indexed id, address indexed killer, uint256 prize, uint256 lef

    struct TokenBank {
        address tokenAddr;
        address gTokenAddr;
        bool isOpen;
        bool canDeposit;
        bool canWithdraw;
        uint256 totalVal;
        uint256 totalDebt;
        uint256 totalDebtShare;
        uint256 totalReserve;
        uint256 lastInterestTime;
        uint256 tokenIdx;
    }

    struct Production {
        address coinToken;
        address currencyToken;
        address borrowToken;
        bool isOpen;
        bool canBorrow;
        address goblin;
        uint256 minDebt;
        uint256 openFactor;
        uint256 liquidateFactor;
    }

    struct Position {
        address owner;
        uint256 productionId;
        uint256 debtShare;
    }

    IBankConfig config;

    mapping(address => TokenBank) public banks;
    uint256 public currentToken = 0;
```

```
44
45      mapping(uint256 => Production) public productions;
46      uint256 public currentPid = 1;
47
48      mapping(uint256 => Position) public positions;
49      uint256 public currentPos = 1;
50
51      modifier onlyEOA() {
52          require(msg.sender == tx.origin, "not eoa");
53          _;
54      }
55
56      constructor() public {}
57
58      /// read
59      function positionInfo(uint256 posId) public view returns (uint256, uint256, uint256, ad
60          Position storage pos = positions[posId];
61          Production storage prod = productions[pos.productionId];
62
63          return (pos.productionId, Goblin(prod.goblin).health(posId, prod.borrowToken),
64              debtShareToVal(prod.borrowToken, pos.debtShare), pos.owner);
65      }
66
67      function totalToken(address token) public view returns (uint256) {
68          TokenBank storage bank = banks[token];
69          require(bank.isOpen, 'token not exists');
70
71          uint balance = token == address(0)? address(this).balance: SafeToken.myBalance(toke
72          balance = bank.totalVal < balance? bank.totalVal: balance;
73
74          return balance.add(bank.totalDebt).sub(bank.totalReserve);
75      }
76
77      function debtShareToVal(address token, uint256 debtShare) public view returns (uint256)
78          TokenBank storage bank = banks[token];
79          require(bank.isOpen, 'token not exists');
80
81          if (bank.totalDebtShare == 0) return debtShare;
82          return debtShare.mul(bank.totalDebt).div(bank.totalDebtShare);
83      }
84
85      function debtValToShare(address token, uint256 debtVal) public view returns (uint256) {
86          TokenBank storage bank = banks[token];
87          require(bank.isOpen, 'token not exists');
88
89          if (bank.totalDebt == 0) return debtVal;
90          return debtVal.mul(bank.totalDebtShare).div(bank.totalDebt);
91      }
92
93
94      /// write
95      function deposit(address token, uint256 amount) external payable nonReentrant {
96          TokenBank storage bank = banks[token];
```

```
97          require(bank.isOpen && bank.canDeposit, 'Token not exist or cannot deposit');
98
99          calInterest(token);
100
101         if (token == address(0)) {
102             amount = msg.value;
103         } else {
104             SafeToken.safeTransferFrom(token, msg.sender, address(this), amount);
105         }
106
107         bank.totalVal = bank.totalVal.add(amount);
108         uint256 total = totalToken(token).sub(amount);
109         uint256 pTotal = GToken(bank.gTokenAddr).totalSupply();
110
111         uint256 pAmount = (total == 0 || pTotal == 0) ? amount: amount.mul(pTotal).div(tota
112         GToken(bank.gTokenAddr).mint(msg.sender, pAmount);
113     }
114
115     function withdraw(address token, uint256 pAmount) external nonReentrant {
116         TokenBank storage bank = banks[token];
117         require(bank.isOpen && bank.canWithdraw, 'Token not exist or cannot withdraw');
118
119         calInterest(token);
120
121         uint256 amount = pAmount.mul(totalToken(token)).div(GToken(bank.gTokenAddr).totalSu
122         bank.totalVal = bank.totalVal.sub(amount);
123
124         GToken(bank.gTokenAddr).burn(msg.sender, pAmount);
125
126         if (token == address(0)) {
127             SafeToken.safeTransferETH(msg.sender, amount);
128         } else {
129             SafeToken.safeTransfer(token, msg.sender, amount);
130         }
131     }
132
133     function opPosition(uint256 posId, uint256 pid, uint256 borrow, bytes calldata data)
134     external payable onlyEOA nonReentrant {
135
136         if (posId == 0) {
137             posId = currentPos;
138             currentPos ++;
139             positions[posId].owner = msg.sender;
140             positions[posId].productionId = pid;
141
142         } else {
143             require(posId < currentPos, "bad position id");
144             require(positions[posId].owner == msg.sender, "not position owner");
145
146             pid = positions[posId].productionId;
147         }
148
149         Production storage production = productions[pid];
```

```
150
        require(production.isOpen, 'Production not exists');
151
152         require(borrow == 0 || production.canBorrow, "Production can not borrow");
153         calInterest(production.borrowToken);
154
155         uint256 debt = _removeDebt(positions[posId], production).add(borrow);
156         bool isBorrowBNB = production.borrowToken == address(0);
157
158         uint256 sendBNB = msg.value;
159         uint256 beforeToken = 0;
160         if (isBorrowBNB) {
161             sendBNB = sendBNB.add(borrow);
162             require(sendBNB <= address(this).balance && debt <= banks[production.borrowToke
163             beforeToken = address(this).balance.sub(sendBNB);
164
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	8	3210	589	595	2026	300

Comments to Code 595/2026 = 29%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complex
JavaScript	N/A	N/A	N/A	N/A	N/A	N/A

Tests to Code = N/A