Smart Contract

Security Assessment

For DeGen

31 Aug 2022

Ascendant

@ascendantproj

www.ascendant.finance

Table of Contents

DISCLAIMER

Ascendant Finance ("Ascendant") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Ascendant.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Ascendant is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Ascendant or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Ascendant retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Ascendant is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Ascendant may, at its discretion, claim bug bounties from third-parties while doing so.

Executive Summary

Severity	Found
High	1
Medium	2
Low	6
Informational	152
Total	159

We performed an independent technical audit to identify Smart Contracts uncertainties. This shall protect the code from illegitimate authorization attempts or external & internal threats of any type. This also ensures end-to-end proofing of the contract from frauds. The audit was performed semi-manually. We analyzed the Smart Contracts code line-by-line and used an automation tool to report any suspicious code.

The following tools were used:

- Truffle
- Remix IDE
- Slither

Overview

This report has been prepared for DeGen on the Binance network. Ascendant provides a user-centered examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

Summary

Project Name	DeGen
Platform	Binance
Language	Solidity

Contracts Assessed

Name	Location
Avatar.sol	Not deployed
AvatarPresale.sol	Not deployed
GameERC721.sol	Not deployed
GameMaster.sol	Not deployed
GameState.sol	Not deployed
Point.sol	Not deployed

Name	Location
PointSwap.sol	Not deployed
ShieldNFT.sol	Not deployed
SpecialNFT.sol	Not deployed
IAvatar.sol	Not deployed
IGameState.sol	Not deployed
IPoint.sol	Not deployed
IPointSwap.sol	Not deployed
OwnableUpgradeable.sol	Imported
AddressUpgradeable.sol	Imported
ContextUpgradeable.sol	Imported
StringsUpgradeable.sol	Imported
IERC20Upgradeable.sol	Imported
PausableUpgradeable.sol	Imported
ReentrancyGuardUpgradeable.sol	Imported
SafeERC20Upgradeable.sol	Imported

Findings Summary

Severity	Found
High	1
Medium	2
Low	6
Informational	152
Total	159

Classification of Issues

High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices, Generally pose a negligible level of risk, if any.

Manual Review

Issues Checking Status

Issue Description	Checking Status
Compiler errors	PASS
Race conditions and Reentrancy. Cross-function race conditions.	PASS
Possible delays in data delivery.	PASS
Oracle calls.	PASS
Front running.	PASS
Timestamp dependence.	FAIL
Integer Overflow and Underflow.	PASS
DoS with Revert.	PASS
DoS with block gas limit.	PASS
Methods execution permissions.	PASS
Economy model of the contract.	PASS
The impact of the exchange rate on the logic.	PASS
Private user data leaks.	PASS
Malicious Event log.	PASS
Scoping and Declarations.	PASS
Uninitialized storage pointers.	PASS

Arithmetic accuracy.	PASS
Design Logic.	PASS
Cross-function race conditions.	PASS
Safe Open Zeppelin contracts implementation and usage.	PASS
Fallback function security.	PASS

Severity	**High**
Contract	GameMaster.sol
Description	Block.timestamp Dependence for Pseudorandomness

Code Snippet	function buySpecialNFT(uint256 nftId) external notBlacklisted notBought(nftId) whenActive nonReentrant {
	uint256 price = specialNFTPrices[nftId];
	require(price > 0, "Price not set");
	busd.safeTransferFrom(msg.sender, treasury, price);
	alreadyBought[nftLevelId][msg.sender][nftId] = true;
	uint256 specialNftId = nftId;
	if (nftId == 9) {
	uint256 randomNum = uint256(keccak256(abi.encodePacked(block.difficulty, block.timestamp)));
	specialNftId = (randomNum % 6) + 1;
	}
	specialNFTs[specialNftId].mint(msg.sender);
	emit NFTBought(msg.sender, nftId, specialNftId);
	}

Recommendation	Block.timestamp can be manipulated by miners. Consider implementing a randomization function off-chain instead.
Status	

Audit Findings

Severity	**Medium**
Contract	ShieldNFT.sol
Description	Checks-Effects-Interactions

Code Snippet	function mint(address user)
	external onlyAuthorizedContracts {
	tokenCount++;
	_mint(user, tokenCount);
	usesLeft[tokenCount] = uses;
	}

Recommendation	Potential reentrancy issue: state should be updated before minting. Switch the order of _mint and usesLeft
Status	

Audit Findings

Severity	**Medium**
Contract	ShieldNFT.sol
Description	Checks-Effects-Interactions
Code Snippet	function consume(address user) external onlyAuthorizedContracts { uint256 userBalance = balanceOf(user); require(userBalance > 0, "Not present"); uint256 tokenId = getLatestUserToken(user); uint256 userUsesLeft = usesLeft[tokenId]; if (userUsesLeft == 1) { _burn(user); } usesLeft[tokenId]--; }
Recommendation	Potential reentrancy issue: Where possible state changes should occur before interactions. Issue can also be mitigated with ReentrancyGuard
Status	

Functional Test Status

Function Name	Type/Return Type	Score
Contract IERC20Upgradeable		
allowance	external	PASS
approve	external	PASS
balanceOf	external	PASS
totalSupply	external	PASS
transfer	external	PASS
transferFrom	external	PASS
Contract AddressUpgradeable		
functionalCall	internal	PASS
functionCallWithValue	internal	PASS
functionStaticCall	internal	PASS
isContract	internal	PASS
sendValue	internal	PASS
verifyCallResult	internal	PASS
SafeERC20Upgradeable		
_callOptionalReturn	private	PASS
safeApprove	internal	PASS
safeDecreaseAllowance	internal	PASS

safeIncreaseAllowance	internal	PASS
safePermit	internal	PASS
safeTransfer	internal	PASS
safeTransferFrom	internal	PASS
ContextUpgradeable		
_msgData	internal	PASS
_msgSender	internal	PASS
PausableUpgradeable		
_pause	internal	PASS
_unpause	internal	PASS
paused	read/public	PASS
OwnableUpgradeable		
owner	read/public	PASS
renounceOwnership	write/public	PASS
transferOwnership	write/public	PASS
ISpecialNFT		
balanceOf	read/external	PASS
consume	write/public	PASS
mint	write/external	PASS
IPointSwap		
gameReset	write/external	PASS

setActiveTime	write/external	PASS
IPoint		
balanceOf	read/external	PASS
burnFrom	write/external	PASS
mint	write/external	PASS
multiBalanceOf	read/external	PASS
transfer	write/external	PASS
transferFrom	write/external	PASS
IGameState		
gameId	read/external	PASS
incrementGameId	write/external	PASS
IAvatar		
balanceOf	read/external	PASS
claimPoints	write/external	PASS
freezeRewards	write/external	PASS
gameReset	write/external	PASS
mint	write/external	PASS
setActiveTime	write/external	PASS
setBoosterTime	write/external	PASS
unFreezeRewards	write/external	PASS
GameERC721		

_afterTokenTransfer	internal	PASS
_beforeTokenTransfer	internal	PASS
_burn	internal	PASS
_exists	internal	PASS
_mint	internal	PASS
balanceOf	read/public	PASS
getGameId	internal	PASS
getLatestUserToken	read/public	PASS
getUserTokens	read/public	PASS
name	read/public	PASS
ownerOf	read/public	PASS
symbol	read/public	PASS
tokenURI	read/public	PASS
SpecialNFT		
consume	write/external	PASS
mint	write/external	PASS
setAuthorizedContract	write/external	PASS
setGameState	write/external	PASS
setTokenURI	write/external	PASS
ShieldNFT		
consume	write/external	PASS

getUsesLeft	read/external	PASS
mint	read/external	PASS
setAuthorizedContract	write/external	PASS
setGameState	write/external	PASS
setTokenURI	write/external	PASS
PointSwap		
_updateK	internal	PASS
gameReset	write/external	PASS
pause	write/external	PASS
removeLiquidity	write/external	PASS
setActiveTime	write/external	PASS
setAuthorizedContract	write/external	PASS
setBuyLimit	write/external	PASS
setBuyRefreshTime	write/external	PASS
setSellLimit	write/external	PASS
setSellRefreshTime	write/external	PASS
swapBusdForPoint	write/external	PASS
swapPointForBusd	write/external	PASS
unpause	write/external	PASS
Point		
balanceOf	read/public	PASS

burnFrom	write/external	PASS
decimals	read/public	PASS
getGameId	internal	PASS
mint	write/external	PASS
multiBalanceOf	read/external	PASS
name	read/public	PASS
setAuthorizedContract	write/public	PASS
setGameState	write/external	PASS
symbol	read/public	PASS
totalSupply	read/public	PASS
transfer	write/external	PASS
transferFrom	write/external	PASS
GameState		
incrementGameId	write/external	PASS
setAuthorizedContract	write/external	PASS
GameMaster		
attackPlayer	write/external	PASS
buySpecialNFT	write/external	PASS
claimPoints	write/external	PASS
getUserShield	internal	PASS
resetNFTBuys	write/external	PASS

setActiveTime	write/external	PASS
setBlacklisted	write/external	PASS
setBoosterParameters	write/external	PASS
setFreezeDuration	write/external	PASS
setPointRemoveAmount	write/external	PASS
setPointStealAmount	write/external	PASS
setSpecialNFTPrices	write/external	PASS
setSpecialNFTs	write/external	PASS
setTreasury	write/external	PASS
startNewGame	write/external	PASS
unfreeze	write/external	PASS
usePointBooster	write/external	PASS
AvatarPresale		
buy	write/external	PASS
setActiveTime	write/external	PASS
setPrice	write/external	PASS
setTreasury	write/external	PASS
setWhitelist	write/external	PASS
Avatar		
_claim	internal	PASS
claimPoints	write/external	PASS

freezeRewards	write/external	PASS
gameReset	write/external	PASS
getClaimAmount	read/public	PASS
getPointsByTokenIds	read/external	PASS
mint	write/external	PASS
setActiveTime	write/external	PASS
setAuthorizedContract	write/external	PASS
setBoosterTime	write/external	PASS
setGameState	write/external	PASS
setMaxSupply	write/external	PASS
setMinter	write/external	PASS
setPoint	write/external	PASS
setPointsPerSecond	write/external	PASS
setTokenURI	write/external	PASS
unfreezeRewards	write/external	PASS

Automated Review

Solidity Static Analysis

Issue	Severity
Check-effects-interaction: NOTE: All flags for checks-effects-interactions have been downgraded from MEDIUM to LOW due to the utilization of Reentrancy Guard. Potential violation of Checks-Effects-Interaction pattern in PointSwap.addLiquidity(uint256,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis. Pos: 454:4:	Low
Check-effects-interaction: **NOTE:** *All flags for checks-effects-interactions have been downgraded from MEDIUM to LOW due to the utilization of Reentrancy Guard.* Potential violation of Checks-Effects-Interaction pattern in PointSwap.removeLiquidity(uint256,uint256): Could potentially lead to re-entrancy vulnerability. Pos: 464.6	Low

Check-effects-interaction: **NOTE:** *All flags for checks-effects-interactions have been downgraded from MEDIUM to LOW due to the utilization of Reentrancy Guard.* Potential violation of Checks-Effects-Interaction pattern in PointSwap.swapPointForBusd(uint256,uint256,uint256) : Could potentially lead to re-entrancy vulnerability. Pos: 519.8	Low
Check-effects-interaction: **NOTE:** *All flags for checks-effects-interactions have been downgraded from MEDIUM to LOW due to the utilization of Reentrancy Guard.* Potential violation of Checks-Effects-Interaction pattern in GameMaster.setActiveTime(uint256,uint256): Could potentially lead to re-entrancy vulnerability. Pos: 796.4	Low
Check-effects-interaction: **NOTE:** *All flags for checks-effects-interactions have been downgraded from MEDIUM to LOW due to the utilization of Reentrancy Guard.* Potential violation of Checks-Effects-Interaction pattern in GameMaster.startNewGame(uint256,uint256): Could potentially lead to re-entrancy vulnerability. Pos: 835.4	Low

Check-effects-interaction: **NOTE:** *All flags for checks-effects-interactions have been downgraded from MEDIUM to LOW due to the utilization of Reentrancy Guard.* Potential violation of Checks-Effects-Interaction pattern in Avatar._claim(address): Could potentially lead to re-entrancy vulnerability. Pos: 1108.4	Low
ShieldNFT.getUsesLeft(address) For loop over dynamic array: Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.	Informational
GameMaster.setSpecialNFTs(...) For loop over dynamic array: Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.	Informational

GameMaster.setSpecialNFTPrices(...) For loop over dynamic array: Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.	Informational
AvatarPresale.setWhitelist...) For loop over dynamic array: Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.	Informational
Data Truncated x3: Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants. Functions: PointSwap.getPointAmount PointSwap.getBusdAmount Avatar.getClaimAmount	Informational

Omitted Results

Note: Any issues that have been omitted from this report have been deemed by the reviewing team as irrelevant, inapplicable, and/or negligible to the proper functioning of this contract. Thus, any omitted issues can be safely ignored.

Conclusion

The smart contracts reviewed in this audit contain no critical severity issues and all High to Medium issues have either been corrected or acknowledged.

Please check the disclaimer above and note, the audit makes no statements or warranties on business model, investment attractiveness or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by Owner.

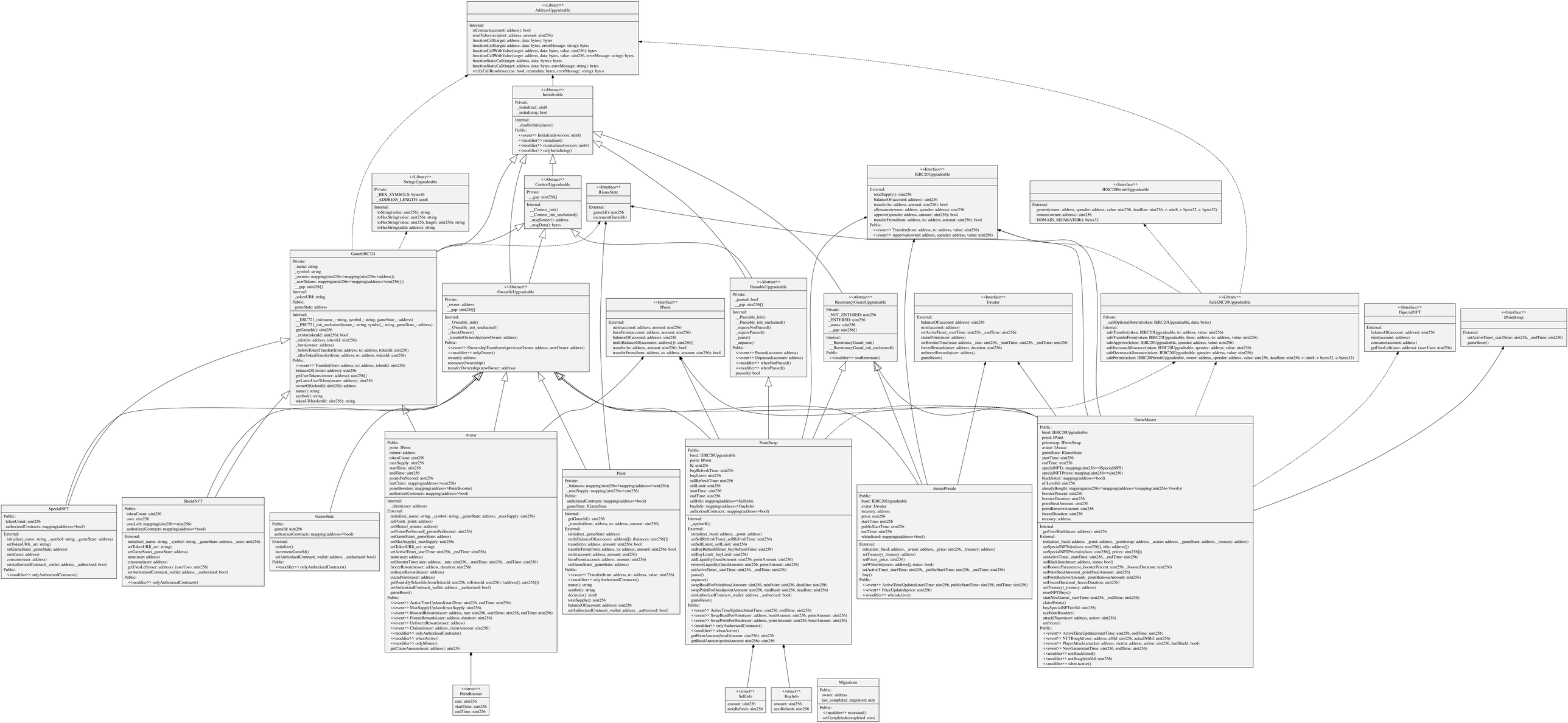
Ascendant

Ascendant

@ascendantproj

www.ascendant.finance

Ascendant


<<Library>>
AddressUpgradeable
Internal:
isContract(account: address): bool
sendValue(recipient: address, amount: uint256)
functionCall(target: address, data: bytes): bytes
functionCall(target: address, data: bytes, errorMessage: string): bytes
functionCallWithValue(target: address, data: bytes, value: uint256): bytes
functionCallWithValue(target: address, data: bytes, value: uint256, errorMessage: string): bytes
functionStaticCall(target: address, data: bytes): bytes
functionStaticCall(target: address, data: bytes, errorMessage: string): bytes
verifyCallResult(success: bool, returndata: bytes, errorMessage: string): bytes
<<Abstract>>
Initializable
Private:
_initialized: uint8
_initializing: bool
Internal:
__disableInitializers()
Public:
<<event>> Initialized(version: uint8)
<<modifier>> initializer()
<<modifier>> reinitializer(version: uint8)
<<modifier>> onlyInitializing()
<<Library>>
StringsUpgradeable
Private:
_HEX_SYMBOLS: bytes16
_ADDRESS_LENGTH: uint8
Internal:
toString(value: uint256): string
toHexString(value: uint256): string
toHexString(value: uint256, length: uint256): string
toHexString(addr: address): string
<<Abstract>>
ContextUpgradeable
Private:
__gap: uint256[]
Internal:
__Context_init()
__Context_init_unchained()
_msgSender(): address
_msgData(): bytes
<<Interface>>
IGameState
External:
gameId(): uint256
incrementGameId()
<<Interface>>
IERC20Upgradeable
External:
totalSupply(): uint256
balanceOf(account: address): uint256
transfer(to: address, amount: uint256): bool
allowance(owner: address, spender: address): uint256
approve(spender: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<event>> Approval(owner: address, spender: address, value: uint256)
<<Interface>>
IERC20PermitUpgradeable
External:
permit(owner: address, spender: address, value: uint256, deadline: uint256, v: uint8, r: bytes32, s: bytes32)
nonces(owner: address): uint256
DOMAIN_SEPARATOR(): bytes32
GameERC721
Private:
_name: string
_symbol: string
_owners: mapping(uint256=>mapping(uint256=>address))
_userTokens: mapping(uint256=>mapping(address=>uint256[]))
__gap: uint256[]
Internal:
_tokenURI: string
Public:
gameState: address
Internal:
__ERC721_init(name_: string, symbol_: string, gameState_: address)
__ERC721_init_unchained(name_: string, symbol_: string, gameState_: address)
getGameId(): uint256
_exists(tokenId: uint256): bool
_mint(to: address, tokenId: uint256)
_burn(owner: address)
_beforeTokenTransfer(from: address, to: address, tokenId: uint256)
_afterTokenTransfer(from: address, to: address, tokenId: uint256)
Public:
<<event>> Transfer(from: address, to: address, tokenId: uint256)
balanceOf(owner: address): uint256
getUserTokens(owner: address): uint256[]
getLatestUserToken(owner: address): uint256
ownerOf(tokenId: uint256): address
name(): string
symbol(): string
tokenURI(tokenId: uint256): string
<<Abstract>>
OwnableUpgradeable
Private:
_owner: address
__gap: uint256[]
Internal:
__Ownable_init()
__Ownable_init_unchained()
_checkOwner()
_transferOwnership(newOwner: address)
Public:
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
owner(): address
renounceOwnership()
transferOwnership(newOwner: address)
<<Interface>>
IPoint
External:
mint(account: address, amount: uint256)
burnFrom(account: address, amount: uint256)
balanceOf(account: address): uint256
multiBalanceOf(accounts: address[]): uint256[]
transfer(to: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
<<Abstract>>
PausableUpgradeable
Private:
_paused: bool
__gap: uint256[]
Internal:
__Pausable_init()
__Pausable_init_unchained()
_requireNotPaused()
_requirePaused()
_pause()
_unpause()
Public:
<<event>> Paused(account: address)
<<event>> Unpaused(account: address)
<<modifier>> whenNotPaused()
<<modifier>> whenPaused()
paused(): bool
<<Abstract>>
ReentrancyGuardUpgradeable
Private:
_NOT_ENTERED: uint256
_ENTERED: uint256
_status: uint256
__gap: uint256[]
Internal:
__ReentrancyGuard_init()
__ReentrancyGuard_init_unchained()
Public:
<<modifier>> nonReentrant()
<<Interface>>
IAvatar
External:
balanceOf(account: address): uint256
mint(account: address)
setActiveTime(_startTime: uint256, _endTime: uint256)
claimPoints(user: address)
setBoosterTime(user: address, _rate: uint256, _startTime: uint256, _endTime: uint256)
freezeRewards(user: address, duration: uint256)
unfreezeRewards(user: address)
gameReset()
<<Library>>
SafeERC20Upgradeable
Private:
_callOptionalReturn(token: IERC20Upgradeable, data: bytes)
Internal:
safeTransfer(token: IERC20Upgradeable, to: address, value: uint256)
safeTransferFrom(token: IERC20Upgradeable, from: address, to: address, value: uint256)
safeApprove(token: IERC20Upgradeable, spender: address, value: uint256)
safeIncreaseAllowance(token: IERC20Upgradeable, spender: address, value: uint256)
safeDecreaseAllowance(token: IERC20Upgradeable, spender: address, value: uint256)
safePermit(token: IERC20PermitUpgradeable, owner: address, spender: address, value: uint256, deadline: uint256, v: uint8, r: bytes32, s: bytes32)
<<Interface>>
ISpecialNFT
External:
balanceOf(account: address): uint256
mint(account: address)
consume(account: address)
getUsesLeft(user: address): (userUses: uint256)
<<Interface>>
IPointSwap
External:
setActiveTime(_startTime: uint256, _endTime: uint256)
gameReset()
SpecialNFT
Public:
tokenCount: uint256
authorizedContracts: mapping(address=>bool)
External:
initialize(_name: string, _symbol: string, _gameState: address)
setTokenURI(_uri: string)
setGameState(_gameState: address)
mint(user: address)
consume(user: address)
setAuthorizedContract(_wallet: address, _authorized: bool)
Public:
<<modifier>> onlyAuthorizedContracts()
ShieldNFT
Public:
tokenCount: uint256
uses: uint256
usesLeft: mapping(uint256=>uint256)
authorizedContracts: mapping(address=>bool)
External:
initialize(_name: string, _symbol: string, _gameState: address, _uses: uint256)
setTokenURI(_uri: string)
setGameState(_gameState: address)
mint(user: address)
consume(user: address)
getUsesLeft(user: address): (userUses: uint256)
setAuthorizedContract(_wallet: address, _authorized: bool)
Public:
<<modifier>> onlyAuthorizedContracts()
GameState
Public:
gameId: uint256
authorizedContracts: mapping(address=>bool)
External:
initialize()
incrementGameId()
setAuthorizedContract(_wallet: address, _authorized: bool)
Public:
<<modifier>> onlyAuthorizedContracts()
Avatar
Public:
point: IPoint
minter: address
tokenCount: uint256
maxSupply: uint256
startTime: uint256
endTime: uint256
pointsPerSecond: uint256
lastClaim: mapping(address=>uint256)
pointBoosters: mapping(address=>PointBooster)
authorizedContracts: mapping(address=>bool)
Internal:
__claim(user: address)
External:
initialize(_name: string, _symbol: string, _gameState: address, _maxSupply: uint256)
setPoint(_point: address)
setMinter(_minter: address)
setPointsPerSecond(_pointsPerSecond: uint256)
setGameState(_gameState: address)
setMaxSupply(_maxSupply: uint256)
setTokenURI(_uri: string)
setActiveTime(_startTime: uint256, _endTime: uint256)
mint(user: address)
setBoosterTime(user: address, _rate: uint256, _startTime: uint256, _endTime: uint256)
freezeRewards(user: address, duration: uint256)
unfreezeRewards(user: address)
claimPoints(user: address)
getPointsByTokenIds(fromTokenId: uint256, toTokenId: uint256): (address[], uint256[])
setAuthorizedContract(_wallet: address, _authorized: bool)
gameReset()
Public:
<<event>> ActiveTimeUpdated(startTime: uint256, endTime: uint256)
<<event>> MaxSupplyUpdated(maxSupply: uint256)
<<event>> BoosterRewards(user: address, rate: uint256, startTime: uint256, endTime: uint256)
<<event>> FrozenRewards(user: address, duration: uint256)
<<event>> UnfrozenRewards(user: address)
<<event>> Claimed(user: address, claimAmount: uint256)
<<modifier>> onlyAuthorizedContracts()
<<modifier>> whenActive()
<<modifier>> onlyMinter()
getClaimAmount(user: address): uint256
Point
Private:
_balances: mapping(uint256=>mapping(address=>uint256))
_totalSupply: mapping(uint256=>uint256)
Public:
authorizedContracts: mapping(address=>bool)
gameState: IGameState
Internal:
getGameId(): uint256
_transfer(from: address, to: address, amount: uint256)
External:
initialize(_gameState: address)
multiBalanceOf(accounts: address[]): (balances: uint256[])
transfer(to: address, amount: uint256): bool
transferFrom(from: address, to: address, amount: uint256): bool
mint(account: address, amount: uint256)
burnFrom(account: address, amount: uint256)
setGameState(_gameState: address)
Public:
<<event>> Transfer(from: address, to: address, value: uint256)
<<modifier>> onlyAuthorizedContracts()
name(): string
symbol(): string
decimals(): uint8
totalSupply(): uint256
balanceOf(account: address): uint256
setAuthorizedContract(_wallet: address, _authorized: bool)
PointSwap
Public:
busd: IERC20Upgradeable
point: IPoint
K: uint256
buyRefreshTime: uint256
buyLimit: uint256
sellRefreshTime: uint256
sellLimit: uint256
startTime: uint256
endTime: uint256
sellInfo: mapping(address=>SellInfo)
buyInfo: mapping(address=>BuyInfo)
authorizedContracts: mapping(address=>bool)
Internal:
_updateK()
External:
initialize(_busd: address, _point: address)
setSellRefreshTime(_sellRefreshTime: uint256)
setSellLimit(_sellLimit: uint256)
setBuyRefreshTime(_buyRefreshTime: uint256)
setBuyLimit(_buyLimit: uint256)
addLiquidity(busdAmount: uint256, pointAmount: uint256)
removeLiquidity(busdAmount: uint256, pointAmount: uint256)
setActiveTime(_startTime: uint256, _endTime: uint256)
pause()
unpause()
swapBusdForPoint(busdAmount: uint256, minPoint: uint256, deadline: uint256)
swapPointForBusd(pointAmount: uint256, minBusd: uint256, deadline: uint256)
setAuthorizedContract(_wallet: address, _authorized: bool)
gameReset()
Public:
<<event>> ActiveTimeUpdated(startTime: uint256, endTime: uint256)
<<event>> SwapBusdForPoint(user: address, busdAmount: uint256, pointAmount: uint256)
<<event>> SwapPointForBusd(user: address, pointAmount: uint256, busdAmount: uint256)
<<modifier>> onlyAuthorizedContracts()
<<modifier>> whenActive()
getPointAmount(busdAmount: uint256): uint256
getBusdAmount(pointAmount: uint256): uint256
AvatarPresale
Public:
busd: IERC20Upgradeable
avatar: IAvatar
treasury: address
price: uint256
startTime: uint256
publicStartTime: uint256
endTime: uint256
whitelisted: mapping(address=>bool)
External:
initialize(_busd: address, _avatar: address, _price: uint256, _treasury: address)
setTreasury(_treasury: address)
setPrice(_price: uint256)
setWhitelist(users: address[], status: bool)
setActiveTime(_startTime: uint256, _publicStartTime: uint256, _endTime: uint256)
buy()
Public:
<<event>> ActiveTimeUpdated(startTime: uint256, publicStartTime: uint256, endTime: uint256)
<<event>> PriceUpdated(price: uint256)
<<modifier>> whenActive()
GameMaster
Public:
busd: IERC20Upgradeable
point: IPoint
pointswap: IPointSwap
avatar: IAvatar
gameState: IGameState
startTime: uint256
endTime: uint256
specialNFTs: mapping(uint256=>ISpecialNFT)
specialNFTPrices: mapping(uint256=>uint256)
blacklisted: mapping(address=>bool)
nftLevelId: uint256
alreadyBought: mapping(uint256=>mapping(address=>mapping(uint256=>bool)))
boosterPercent: uint256
boosterDuration: uint256
pointStealAmount: uint256
pointRemoveAmount: uint256
freezeDuration: uint256
treasury: address
Internal:
getUserShield(user: address): uint256
External:
initialize(_busd: address, _point: address, _pointswap: address, _avatar: address, _gameState: address, _treasury: address)
setSpecialNFTs(indices: uint256[], nfts: address[])
setSpecialNFTPrices(indices: uint256[], prices: uint256[])
setActiveTime(_startTime: uint256, _endTime: uint256)
setBlacklisted(user: address, status: bool)
setBoosterParameters(_boosterPercent: uint256, _boosterDuration: uint256)
setPointStealAmount(_pointStealAmount: uint256)
setPointRemoveAmount(_pointRemoveAmount: uint256)
setFreezeDuration(_freezeDuration: uint256)
setTreasury(_treasury: address)
resetNFTBuys()
startNewGame(_startTime: uint256, _endTime: uint256)
claimPointe()
buySpecialNFT(nftId: uint256)
usePointBooster()
attackPlayer(user: address, action: uint256)
unfreeze()
Public:
<<event>> ActiveTimeUpdated(startTime: uint256, endTime: uint256)
<<event>> NFTBought(user: address, nftId: uint256, actualNftId: uint256)
<<event>> PlayerAttack(attacker: address, victim: address, action: uint256, hadShield: bool)
<<event>> NewGame(startTime: uint256, endTime: uint256)
<<modifier>> notBlacklisted()
<<modifier>> notBought(nftId: uint256)
<<modifier>> whenActive()
<<struct>>
PointBooster
rate: uint256
startTime: uint256
endTime: uint256
<<struct>>
SellInfo
amount: uint256
nextRefresh: uint256
<<struct>>
BuyInfo
amount: uint256
nextRefresh: uint256
Migrations
Public:
owner: address
last_completed_migration: uint
Public:
<<modifier>> restricted()
setCompleted(completed: uint)