Security Review Report
NM-0236 Munchables

(May 30, 2024)

Contents

1 Executive Summary

This document outlines the security review conducted by Nethermind for the Munchables protocol implementation. Munchables is a GameFi protocol where users aim to accumulate as many points as they can though a variety of actions such as feeding, petting, hatching primordials and managing their Snuggery. This Munchables implementation supports migrating NFTs from a previous version of the game, allowing existing users to keep their original Munchable NFTs.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract, and (d) creation of test cases. Along with this document, we report 37 points of attention, where 1 is classified as `Critical`, 7 are classified as `High`, 6 are classified as `Medium`, 6 are classified as `Low`, 10 are classified as `Info` and 7 are classified as `Best Practices`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope of this audit. Section 3 summarizes the issues. Section 4 describes the centralization risks. Section 5 describes the system overview. Section 6 discusses the risk rating methodology adopted for this audit. Section 7 details the issues. Section 8 discusses the documentation provided by the client for this audit. Section 9 presents the compilation, tests, and automated tests. Section 10 concludes the document.

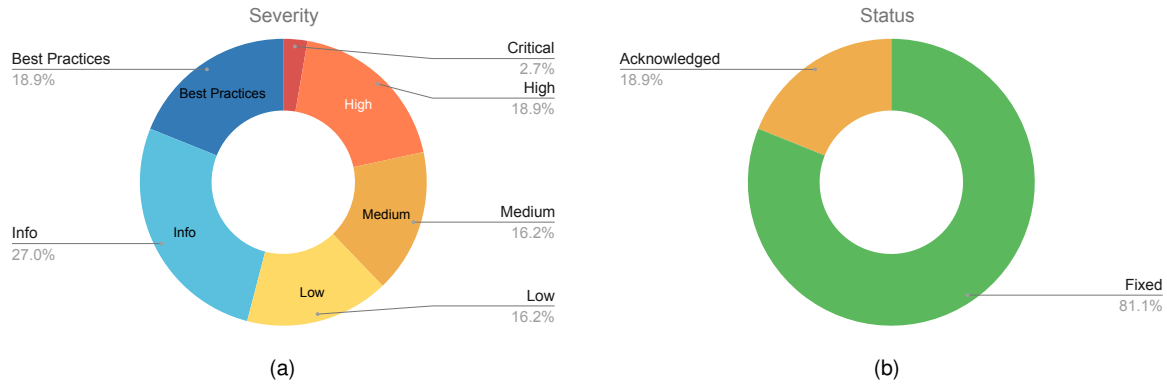

Fig. 1: Distribution of issues: Critical (1), **High** (7), **Medium** (6), **Low** (6), **Undetermined** (0), **Informational** (10), **Best Practices** (7). **Distribution of status: Fixed** (30), **Acknowledged** (7), **Mitigated** (0), **Unresolved** (0), **Partially Fixed** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	May 27, 2024
Response from Client	May 28, 2024
Final Report	May 30, 2024
Repository	munchablesorg/munchables-common-core
Commit (Audit)	`5f3087a78efff1c3c1ef4c2f248dd8a231127749`
Commit (Final)	`2056e1c58d8427d18f7a6c0be293d4602e8f27c6`
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/proxy/ProxyFactory.sol	9	1	11.1%	2	12
2	src/config/BaseConfigStorage.sol	73	1	1.4%	14	88
3	src/config/ConfigStorage.sol	295	19	6.4%	48	362
4	src/config/BaseConfigStorageUpgradeable.sol	18	1	5.6%	4	23
5	src/managers/BaseBlastManager.sol	83	6	7.2%	15	104
6	src/managers/LockManager.sol	382	42	11.0%	66	490
7	src/managers/ClaimManager.sol	204	5	2.5%	35	244
8	src/managers/SnuggeryManager.sol	267	22	8.2%	55	344
9	src/managers/MigrationManager.sol	331	3	0.9%	44	378
10	src/managers/BonusManager.sol	227	7	3.1%	26	260
11	src/managers/RewardsManager.sol	170	4	2.4%	26	200
12	src/managers/MunchadexManager.sol	115	1	0.9%	15	131
13	src/managers/AccountManager.sol	312	19	6.1%	46	377
14	src/managers/BaseBlastManagerUpgradeable.sol	22	1	4.5%	3	26
15	src/managers/PrimordialManager.sol	125	8	6.4%	33	166
16	src/managers/NFTAttributeManagerV1.sol	163	5	3.1%	29	197
17	src/distributors/FundTreasuryDistributor.sol	39	1	2.6%	8	48
18	src/libraries/MunchablesCommonLib.sol	147	3	2.0%	23	173
19	src/overlords/NFTOverlord.sol	373	31	8.3%	71	475
20	src/tokens/OldMunchNFT.sol	169	81	47.9%	31	281
21	src/tokens/MunchToken.sol	25	4	16.0%	5	34
22	src/tokens/MunchNFT.sol	118	2	1.7%	15	135
23	src/tokens/TestERC20Token.sol	24	4	16.7%	6	34
24	src/rng/RNGProxySelfHosted.sol	14	1	7.1%	3	18
25	src/rng/BaseRNGProxy.sol	40	1	2.5%	14	55
26	src/rng/RNGProxyAPI3.sol	48	3	6.2%	8	59
27	src/interfaces/IBlast.sol	97	11	11.3%	11	119
28	src/interfaces/ILockManager.sol	117	125	106.8%	57	299
29	src/interfaces/IMunchNFT.sol	6	8	133.3%	3	17
30	src/interfaces/IMigrationManager.sol	80	57	71.2%	25	162
31	src/interfaces/INFTAttributesManager.sol	52	34	65.4%	17	103
32	src/interfaces/IConfigStorage.sol	169	7	4.1%	34	210
33	src/interfaces/IHoldsGovernorship.sol	4	3	75.0%	1	8
34	src/interfaces/IClaimManager.sol	92	25	27.2%	12	129
35	src/interfaces/ISnuggeryManager.sol	61	71	116.4%	28	160
36	src/interfaces/IConfigNotifiable.sol	5	1	20.0%	2	8
37	src/interfaces/IMunchadexManager.sol	28	8	28.6%	3	39
38	src/interfaces/IPrimordialManager.sol	27	16	59.3%	15	58
39	src/interfaces/IBaseBlastManager.sol	4	1	25.0%	1	6
40	src/interfaces/IRNGProxy.sol	20	5	25.0%	3	28
41	src/interfaces/IMunchToken.sol	4	2	50.0%	1	7
42	src/interfaces/IBonusManager.sol	16	3	18.8%	5	24
43	src/interfaces/IDistributor.sol	9	3	33.3%	3	15
44	src/interfaces/INFTOverlord.sol	74	65	87.8%	26	165
45	src/interfaces/IAccountManager.sol	83	93	112.0%	36	212
46	src/interfaces/IRNGProxySelfHosted.sol	4	1	25.0%	1	6
47	src/interfaces/IRewardsManager.sol	11	1	9.1%	5	17
	Total	**4756**	**816**	**17.2%**	**934**	**6506**

3 Summary of Issues

	Finding	Severity	Update
1	Malicious caller can migrate other player's NFTs to their address	Critical	Fixed
2	Account registration can be blocked by adding future user as sub-account	High	Fixed
3	Downcast can reduce number of unrevealed NFTs to be minted	High	Fixed
4	Function `_calculateLevelBonus(...)` reverts with a Snuggery length more than 10	High	Fixed
5	Loss of precision in `_claimPoints`	High	Fixed
6	Players cannot spend points	High	Fixed
7	USDPrice cannot be updated in the lockManager	High	Fixed
8	User cannot export non-zero-chonk Munchables from their Snuggery	High	Fixed
9	A max reveal queue greater than one will overwrite existing reveal data	Medium	Fixed
10	Player can pet their own Munchables	Medium	Fixed
11	Player lock durations can be less than the default minimum	Medium	Fixed
12	Players can mint multiple munchables for the same primordial	Medium	Fixed
13	Players can reduce their asset unlock time	Medium	Fixed
14	The function `getSnuggery` reverts with the index out of bounds error	Medium	Fixed
15	Any user can access the full migration bonus	Low	Fixed
16	Function `getFullPlayerData` may not return accurate Snuggery data	Low	Fixed
17	MunchNFT contract's pause mechanism is unreachable	Low	Fixed
18	The last entry in a user's sub-account array cannot be removed	Low	Fixed
19	Thresholds can be updated while a proposal is in progress	Low	Fixed
20	User can set their own address as a referrer	Low	Fixed
21	A price proposal can be both approved and disapproved	Info	Fixed
22	Double approval account check when approving USD price	Info	Acknowledged
23	Duplicate notifiable addresses and handled differently in add/removal	Info	Acknowledged
24	Non configured but active tokens result in division by zero	Info	Fixed
25	Primordials can be fed after hatching	Info	Fixed
26	Redundant loop in function `addSubAccount`	Info	Acknowledged
27	The same spray proposal can be executed multiple times	Info	Fixed
28	Unnecessary storage write when player chonks haven't changed	Info	Acknowledged
29	Wrong error message in `burnUnrevealedForPoints(...)`	Info	Fixed
30	tokenLocked can be overwritten during migration snapshot loading	Info	Acknowledged
31	Common storage does not have provision for additional features in future	Best Practices	Acknowledged
32	Lock pragmas to specific compiler version	Best Practices	Fixed
33	Migration data token ID should not be zero	Best Practices	Fixed
34	Missing `_disableInitializers(...)` function in upgradeable contracts	Best Practices	Fixed
35	Redundant inheritance	Best Practices	Fixed
36	Unused imports	Best Practices	Acknowledged
37	Use of legacy call to transfer native tokens	Best Practices	Fixed

4 Centralization risks

This section of the report will highlight centralization risks that are present within the protocol. It should be noted that the Munchables team has indicated that they will be using a multisig wallet to ensure that privileged access is controlled and used correctly to mitigate risks.

The USD price of the lockable assets are set by privileged roles: The price of lockable assets which are used to mint Munchables are set by addresses with the PriceFeed role instead of using a more common asset pricing mechanism such as an oracle. The PriceFeed addresses will submit a price proposal which can be voted on. Once the threshold of "for" votes is met, the new price will be accepted, or in the case of "against" votes, the proposal will be discarded. Although a voting process exists, the pricing mechanism for assets is more centralized than relying on an oracle. The admin can also directly overwrite the price using the `configureToken` function, bypassing any proposals. Additionally, the price updates will be less common, so during the time windows between price updates, the USD price of assets may be stale and when users are locking assets, meaning the users may not be paying the correct amount.

All protocol storage is tracked in one contract: The Munchables protocol consists of many contracts, but they all refer to a single contract `ConfigStorage` to query storage. This contract uses enum inputs as keys to store fixed size types and dynamic types like arrays. All storage can be written or overwritten directly by the admin affecting the protocol as a whole. In traditional protocol structures where storage is managed on the same contract, allowing for input validation when setting parameters, it is not possible to add input validation using the central storage approach in `ConfigStorage`. This allows the admin to place the protocol in invalid states by setting storage data such that depending contracts may fail once data has been queried. For example setting an important address to zero, causing contract calls to fail.

Infinite Schnibbles spraying: Schnibbles can be sent to any address through a proposal and execute mechanism, requiring the role "Social" to propose and "SocialApproval" to execute. There are no limits on the amount of Schnibbles that can be sprayed, nor the frequency at which the proposals are executed. The addresses assigned to the "Social" and "SocialApproval" roles are able to create and distribute Schnibbles to any address.

RNG oracle can fallback to self hosted with manual inputs: By default the RNG source will be an API3 RNG oracle, which is a decentralized source of randomness. As a backup feature, the Munchables protocol has a self hosted RNG oracle where a privileged role "NFTOracle" can set the random output to be any fixed value. There is potential for preferential treatment or ensuring specific outputs to lead to certain attributes in a minted Munchable.

5 System Overview

This section presents basic concepts on the game Munchables and a system overview of the audited contracts.

5.1 Munchables

The game involves earning MUNCH points by owning and managing Munchables (NFTs). Players feed their own and pet others' Munchables and refer more players to earn points. Each Munchable is a different creature and has multiple distinct attributes that impact the balance of the game. Players can obtain a Munchable by migrating from season 1, locking up funds, buying one on the open market, or minting and leveling up a primordial to hatch a new Munchable. Players use their Snuggery, a customizable home, to manage up to 6 Munchables, but they can convert points to increase the number of slots. Players import their Munchables in their Snuggery to feed them Schnibbles. The allocated number of Schnibbles is based on how much a player has locked and if the player has a migration bonus. The impact of Schnibbles varies when fed to Munchables, converting into Chonks (XP). As Munchables reach new Chonk thresholds, they level up, enhancing their stats and evolving into new forms.

5.2 High Level Visual Overview of Munchables

The component diagram in Fig. 2 presents the structural design of the protocol. This diagram shows the relationship between the different components of the system. A component diagram allows developers to group contracts based on a common purpose so that others can examine a software project at a high level. This audit consists of 22 smart contracts and 1 (one) library. The package `config` relies on the contracts responsible for storing all information throughout the Munchables protocol.

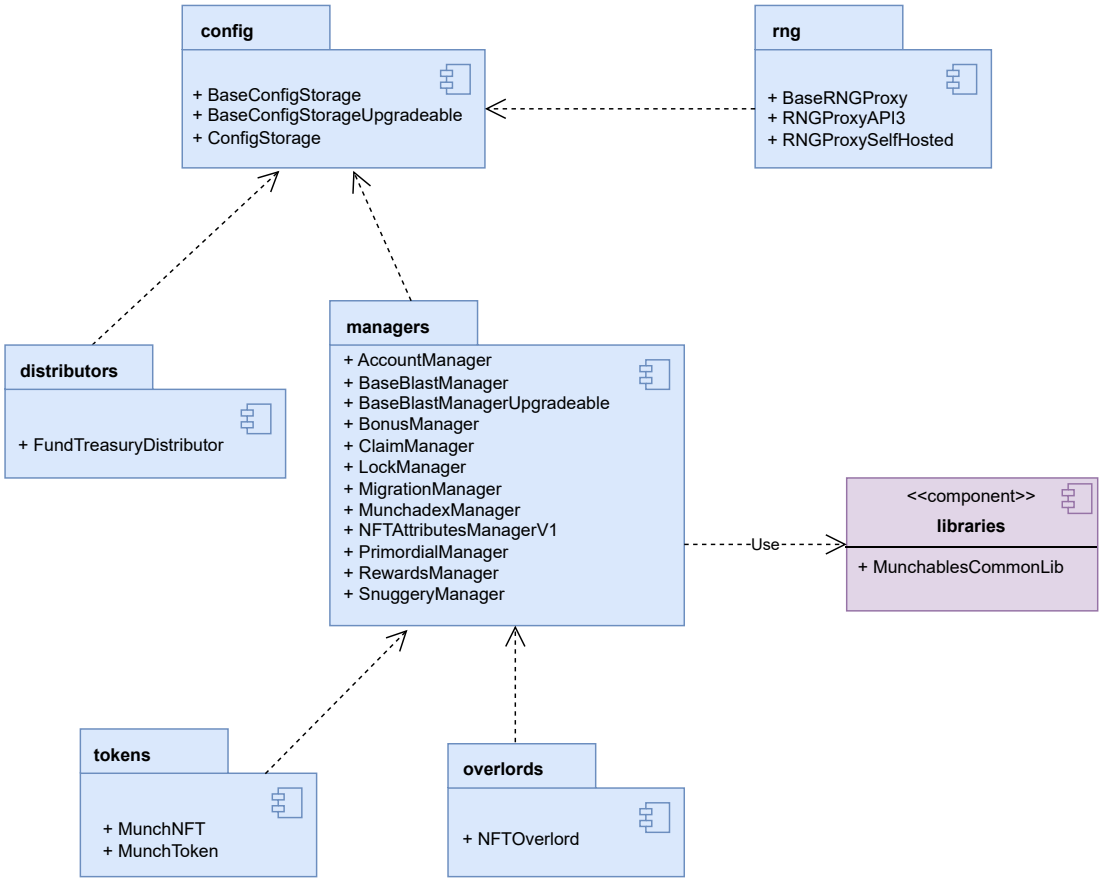

Fig. 2: Component diagram of the audited contracts

1. Package **config** - this package relies on the contracts responsible for managing central functionalities throughout the Munchables protocol.
   a. **ConfigStorage**: The central storage location for the Munchables protocol. All other contracts inherit an abstract contract that allows them to interact with the config storage contract to query protocol data such as important addresses, RNG data for attribute generation, point multipliers and much more. Storage data can be modified on the config storage contract, and the changes will be reflected across the protocol.
   b. **BaseConfigStorage**: Every other contract in the system inherits this contract that handles the initialization and connection to the **ConfigStorage**.

2. Package **managers** - this package contains most of the system's contracts. These contracts depend on the **config** package since they inherit **BaseConfigStorage**. Particularly, provide the game functionalities that interface with external-facing actions, such as locking up funds, importing Munchables into the Snuggery, feeding Munchables, etc. The contracts are presented below:
   a. **AccountManager**: Handles new player registration for both main-accounts and sub-accounts. Other contracts within the Munchables protocol query the account manager to get the main account associated with a given caller address. The schnibble spray proposal and execute mechanism is also implemented in this contract.
   b. **BonusManager**: Contains all functions necessary for bonus calculations, including feed bonuses, harvest bonuses and pet bonuses. These bonuses can be affected by migrations, lock duration, Munchable levels, referrals and the Munchadex.
   c. **ClaimManager**: Handles all logic related to claiming points, as well changing to the next period. Points can be manually claimed or force claimed when other interactions in the protocol are done. Points can be converted into munch tokens.
   d. **LockManager**: Users lock their funds on this contract in exchange for Munchable NFTs. Three different assets are accepted: WETH, USDB, and native Ether. The USD price proposal system is managed on this contract as well, using a voting system controlled by privileged addresses.
   e. **MigrationManager**: Logic for supporting the migration of Season 1 Munchables over to the new implementation. The admin will upload a snapshot containing all previous player data, and users can migrate their NFTs over, burning their old and minting a new replacement in the process.
   f. **MunchadexManager**: Tracks the circulation on Munchables based on attributes and stats such as by realm and rarity, also tracking the unique counts.
   g. **NFTAttributesManagerV1**: Stores the attributes associated with each unique Munchable. Keeping the attributes logic separate from the actual Munchable ERC721 contract was a design choice by the Munchables team to allow for modularity, in case changes need to be made in the future.
   h. **PrimordialManager**: Handles the claiming, feeding, levelling and hatching of Primordials, an alternative pathway to a Munchable without having to lock assets.
   i. **RewardsManager**: Handles blast yield claiming and gas fee claiming.
   j. **SnuggeryManager**: Logic for placing Munchables in the Snuggery including importing, exporting, feeding, petting and increasing the snuggery size.

3. Package **tokens** - consists of contracts that implement ERC-20 and ERC-721.
   a. **MunchNFT**: The Munchable NFT contract, an ERC-721 implementation with token IDs that increment starting at ID of 1.

4. Package **overlords** - this contract contains a single contract described below:
   a. **NFTOverlord**: Used for creating and revealing Munchables both through assets locks and primordials. Will use RNG data to generate attributes which are passed and stored on the NFT attributes manager contract.

5. Package **rng** - contains two smart contracts to request random numbers:
   a. **RNGProxyAPI3**: The primary RNG contract, where the source of the randomness is API3. This is the default RNG contract, with a backup RNG source in case the randomness source needs to be changed.
   b. **RNGProxySelfHosted**: Exists as a backup RNG source, in case the API3 RNG source stops behaving correctly. This source of randomness is centralized, but can be used in emergency situations.

6 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

7 Issues

7.1 [Critical] Malicious caller can migrate other player's NFTs to their address

File(s): MigrationManager.sol

Description: When users want to migrate all their locked NFTs, they call the function `migrateAllNFTs`. This will migrate a user's NFTs in batches of five per call, and importantly, this function can be called by anyone. A snippet from the function is shown below:

```
function migrateAllNFTs(address _user, uint32 _skip) external override {
    if (_userLockedAction[_user] != UserLockedChoice.LOCKED_FULL_MIGRATION)
    // ...
    _migrateNFTs(_user, _userLockedAmounts[_user].tokenLocked, tokenIds);
}
```

When `migrateAllNFTs` is called, the internal function `_migrateNFTs` will be invoked, which loops through the array of `tokenIds` to verify migration snapshot data, and mint the new token. As shown below, the mint is done through `NFTOverlord.mintForMigration`, with the `msg.sender` passed as the owner of the new token.

```
function _migrateNFTs(...) internal {
    // ...
    for (; i < tokenIds.length; i++) {
        // Snapshot validations done here
        // ...

        // @audit Caller will be owner, callable by anyone
        //        Caller isn't guaranteed to be original owner
        newTokenIds[i] = _nftOverlord.mintForMigration(
            msg.sender,
            snapshot.attributes,
            snapshot.immutableAttributes,
            snapshot.gameAttributes
        );

        migratedTokenIds[i] = snapshot.tokenId;
        _oldNFTContract.burn(tokenId);
    }
    // ....
}
```

Since `msg.sender` is used as the argument for the new owner, and the function is callable by anyone, it is possible to steal migration tokens from other users.

Recommendation(s): Consider changing the input in `mintForMigration` from `msg.sender` to the `_user` argument to prevent migration tokens from being stolen by other users.

Status: Fixed

Update from the client: Fix in pull request #179, commit hash `cd1f4b0`.

7.2 [High] Account registration can be blocked by adding future user as sub-account

File(s): AccountManager.sol

Description: The function `register` is used to add a new main-account to the protocol and initialize necessary storage such as registration date, unfed Schnibbles and referrer. There is a check which will revert execution if the caller is already a sub-account, to prevent an address from being both a sub-account and main-account at the same time. This check is shown below:

```
function register(...) external /* ... */ {
    if (mainAccounts[msg.sender] != address(0))
        revert SubAccountCannotRegisterError();
    // ...
}
```

When adding a new sub-account, there are no guarantees that the caller has any control over the sub-account they are assigning to their main-account. A malicious user can add victim addresses as a sub-account before the victim is able to call `register` themselves, and they will no longer be able to register or interact with the protocol. The `addSubAccount` function is shown below:

```
function addSubAccount(address _subAccount) external /* ... */ {
    if (mainAccounts[_subAccount] != address(0))
        revert SubAccountAlreadyRegisteredError();

    // @audit The `_subAccount` address may not belong to the caller
    //        and could be any arbitrary address, including future users

    mainAccounts[_subAccount] = msg.sender;

    for (uint256 i; i < subAccounts[msg.sender].length; i++) {
        if (subAccounts[msg.sender][i] == _subAccount)
            revert SubAccountAlreadyRegisteredError();
    }

    subAccounts[msg.sender].push(_subAccount);
    // ...
}
```

Given that these contracts will be deployed on the Blast Network which is based on the OP Stack, there is no public mempool to watch for registration activity to frontrun. However, historical interactions with the previous Munchables implementation can be analyzed to find all existing users. This data can then be used to prevent registration from all original Munchables users.

Recommendation(s): Consider implementing a mechanism to ensure that a given address permits being assigned as a sub-account to some main-account address. This will prevent arbitrary addresses from being added as a sub-account without permission.

Status: Fixed

Update from the client: PR #179, commit hash 5694d94. In this hash, I basically made it so that when you are registering, if you are a subaccount of another main account, it automatically removes you as a subaccount and allows you to continue forward with registering. Don't think it's necessary to have an additional signing tx as long as there is nothing else that is limiting the actions of a user.

7.3 [High] Downcast can reduce number of unrevealed NFTs to be minted

File(s): LockManager.sol

Description: When a user locks their assets, they receive some number of unrevealed NFTs based on the `nftCost` for the given asset. While locking assets, the internal function `_lock` is called, which calculates the unrevealed NFT count and makes a call to `NFTOverlord.addReveal` to update the unrevealed NFT count for the user. A snippet from the `_lock` function is shown below:

```
function _lock(...) private {
    // ...
    if (...) {
        // ...
        remainder = quantity % configuredToken.nftCost;
        numberNFTs = (quantity - remainder) / configuredToken.nftCost;

        // @audit The `numberNFTs` is downcast to uint8
        //        This downcast can lead to information loss
        nftOverlord.addReveal(_lockRecipient, uint8(numberNFTs));
    }
    // ...
}
```

If a user locks enough assets such that the `numberNFTs` is calculated to be greater than 255, when it is downcast to a `uint8` for the call to `addReveal` some bits may be truncated, causing the user to receive less unrevealed NFTs than they have paid for. An example is shown below:

```
// 255 is  11111111
// 256 is 100000000, when downcast to uint8 is 00000000
uint8(uint256(255)) == 255
uint8(uint256(256)) == 0
```

Recommendation(s): Consider changing the argument type for `_quantity` in `addReveal` to be a uint16, and add input validation in `_lock` to ensure that no value will be lost when downcasting from `uint256` to `uint16`.

Status: Fixed

Update from the client: PR #179 commit hash 22d1c1f. Used recommendation.

7.4 [High] Function `_calculateLevelBonus(...)` reverts with a Snuggery length more than 10

File(s): `BonusManager.sol`

Description: The function `_calculateLevelBonus(...)` calculates level bonus in two scenarios:

1. Everytime the player wants to calculate Schnibbles to distribute and credit to unfed Schnibbles and;
2. When a user adds to their lock to force claim at the previous locked value;

```
    function _calculateLevelBonus(address _caller) internal view
        returns (uint256 _levelBonus) {
        (
            ,
            MunchablesCommonLib.Player memory player,
            MunchablesCommonLib.SnuggeryNFT[] memory _snuggery,
            uint256 _snuggerySize
        ) = _accountManager.getFullPlayerData(_caller);
        if (_snuggerySize > 0) {
            uint i;
            // @audit _snuggerySize can be greater than _snuggery length
            for (; i < _snuggerySize; i++) {
                _levelBonus += _nftAttributesManager
                    .getAttributes(_snuggery[i].tokenId)
                    .level;
            }
            // ...
        }
    }
```

The function invokes `_accountManager.getFullPlayerData(...)` to get the first 10 munchables in the snuggery and the length of the player snuggery to calculate level bonus. Then, `_calculateLevelBonus(...)` loops `_snuggery` to sum the level of each munchable. However, the variable `_snuggerySize` value can be greater than the `_snuggery` length, causing an index out of bounds error.

Recommendation(s): Set the upper limit in the loop to the `_snuggery` length.

Status: Fixed

Update from the client: We removed pagination all together and set a global maximum for snuggery size in these two commits: `63906fc8366dd48bf17bcdff151059cf32c53594`, `1e6d63c67e967ab000036aaf4b7151299158b2fe`.

7.5 [High] Loss of precision in `_claimPoints`

File(s): `ClaimManager.sol`

Description: The internal function `_claimPoints` is used to calculate and update the number of points that a player should receive. The `claimAmount` is calculated by dividing the player's total chonk count by the global total chonk, and then multiplying by the available points, as shown below:

```
uint256 claimAmount = (snuggeryManager.getTotalChonk(_player) /
                       currentPeriod.globalTotalChonk) * availablePoints;
```

Dividing and then multiplying values can lead to loss of precision. In this case since the player chonk count will always be less than the global chonk count, this will always result in a `claimAmount` of zero, meaning users will never be able to claim points. Consider the example below:

```
totalPlayerChonk = 200
totalGlobalChonk = 300
availablePoints  = 456
// Results in...
(200 / 300) * 456 == 0
// Expected result was 304
```

Recommendation(s): Multiply the player total chonk count by the available points before division.

```
uint256 claimAmount = (snuggeryManager.getTotalChonk(_player) *
                       availablePoints) / currentPeriod.globalTotalChonk;
```

Status: Fixed

Update from the client: Resolved here.

7.6 [High] Players cannot spend points

File(s): SnuggeryManager.sol

Description: Players can call the function `increaseSnuggerySize` to exachange their points towards buying additional snuggery space. This function will invoke `ClaimManager.spendPoints` to verify and adjust their points balance before the snuggery size is increased. The function is shown below:

```
function increaseSnuggerySize(uint8 _quantity) external notPaused {
    // ...

    uint16 previousSize = _player.maxSnuggerySize;
    uint256 pointsCost = NEW_SLOT_COST * uint256(_quantity);

    // @audit The function spendPoints(...) can be called only by ClaimManager
    claimManager.spendPoints(_caller, pointsCost);

    _player.maxSnuggerySize += uint16(_quantity);

    // ...
}
```

The function `ClaimManager.spendPoints` has a modifier which only allows it to be called by the `ClaimManager` contract, which itself. As a result, the function `increaseSnuggerySize` will always revert and users are not able to spend their points. The function is presented below:

```
function spendPoints(
    address _player,
    uint256 _spendPoints
) external override onlyConfiguredContract(StorageKey.ClaimManager) {
    // @audit The modifier will not allow call from SnuggeryManager
    if (_points[_player] < _spendPoints) revert NotEnoughPointsError();
    _points[_player] -= _spendPoints;
    emit PointsSpent(_player, _spendPoints);
}
```

Recommendation(s): Consider changing the modifier to allow a call from the `SnuggeryManager` contract.

Status: Fixed

Update from the client: Resolved here.

7.7 [High] USDPrice cannot be updated in the lockManager

File(s): LockManager.sol

Description: The USD price is maintained in the LockManager manually through a proposal and approval framework. The "pricefeed" roles have entitlements to propose a revised USD Price which will then be approved by other with similar role. The USD price update is submitted as a proposal using the structure below. Note the mapping for approvals and disapprovals:

```
struct USDUpdateProposal {
    uint32 proposedDate;
    address proposer;
    address[] contracts;
    uint256 proposedPrice;
    mapping(address => bool) approvals;
    mapping(address => bool) disapprovals;
    uint8 approvalsCount;
    uint8 disapprovalsCount;
}
```

When a struct contains a storage mapping, deleting the struct does not delete the entries inside the mapping. So, if the structure is again reused, there is a risk of stale data being used in subsequent interactions. When the USD price proposal is submitted by address 0x1234, the approvals mapping for 0x1234 is flagged as true.

```
usdUpdateProposal.approvals[msg.sender] = true;
```

Assuming that the proposal hits the threshold to update the USD price and is executed, post execution, the proposal is deleted as below in _execUSDPriceUpdate. The delete will not be able to clear the mapping entries and hence, 0x1234 will remain in the approvals mapping.

```
function _execUSDPriceUpdate() internal {
    if (
        usdUpdateProposal.approvalsCount >= APPROVE_THRESHOLD &&
        usdUpdateProposal.disapprovalsCount < DISAPPROVE_THRESHOLD
    ) {
        // ...
        delete usdUpdateProposal;
    }
}
```

As the next USD price proposal is submitted, 0x1234 will not be able to approve it because of the following condition in approveUSDPrice since the delete did not clear the mapping entry and hence, it will be accounted as already approved.

```
if (usdUpdateProposal.approvals[msg.sender])
      revert ProposalAlreadyApprovedError();
```

This means that once a given address approved or disapproved a proposal, any interactions on following proposals will revert, effectively preventing the USD price updates.

Recommendation(s): Consider using an array to track votes, or change to a nested mapping in the struct to include the proposal round. This will ensure that previous votes won't remain on a new proposal.

Status: Fixed

Update from the client: Fixed in commit 8b3891d155d4c66e82818eb0381218c495542ea3.

7.8 [High] User cannot export non-zero-chonk Munchables from their Snuggery

File(s): SnuggeryManager.sol

Description: The modifier `chonkUpdated` is used to update the player and global chonk count whenever the functions `feed`, `importMunchable`, or `exportMunchable` are called. After the main function logic has executed, the modifier will invoke the internal function `_recalculateChonks` to determine how the player's chonk count has changed, and reflect that difference on the global chonk count, as shown below:

```
function _recalculateChonks(address _player) internal {
    uint256 previousChonks = playerChonks[_player];
    uint256 _playerChonks;
    for (uint256 i; i < snuggeries[_player].length; i++) {
        _playerChonks += nftAttributesManager
            .getAttributes(snuggeries[_player][i].tokenId)
            .chonks;
    }
    playerChonks[_player] = _playerChonks;

    if (previousChonks != _playerChonks) {
        // @audit When Munchable exported the previousChonks is greater than _playerChonks
        //        This will lead to an overflow error, transaction will revert
        totalGlobalChonk += (_playerChonks - previousChonks);
    }
}
```

When a Munchable is exported using `exportMunchable`, the chonk count of the player's snuggery will decrease (assuming the exported Munchable has a non-zero chonk count). The global chonk count is increased by the difference in player chonks: `_playerChonks - previousChonks`. On a call to `exportMunchable` the `_playerChonks` will be less than the `previousChonks`, causing an overflow and reverting the transaction. This prevents any Munchable with a non-zero chonk count from being exported.

Recommendation(s): Consider changing the global chonk count calculation to handle cases where the player's new chonk count is less than their previous count:

```
if (previousChonks != _playerChonks) {
    totalGlobalChonk = (totalGlobalChonk + _playerChonks) - previousChonks;
}
```

Status: Fixed

Update from the client: Fix at commit hash: 738fc47

7.9 [Medium] A max reveal queue greater than one will overwrite existing reveal data

File(s): BaseRNGProxy.sol

Description: The process of revealing a new Munchable is done in two stages, the `startReveal` which is initiated by the user, and then the `reveal` which is initiated by the API3 `RNGProxy`. There is a storage variable `MAX_REVEAL_QUEUE` which limits the number of unrevealed NFTs that can be in the reveal queue. This is set to a default of 1, with plans to be increased post-deployment. All the logic inside of the `NFTOverlord` contract behaves correctly with any value of `MAX_REVEAL_QUEUE` however, the tracked RNG requests in the `BaseRNGProxy` implementation are not able to track unique requests from the same main-account.

```
function requestRandom(
    address _contract,
    bytes4 _selector,
    uint256 _index
) public virtual onlyConfiguredContract(StorageKey.NFTOverlord) {
    // @audit The `_index` key is the user address, not unique on multiple requests
    //        Each new request will overwrite the last, and the associated NFT will be lost
    requests[_index] = RequestData({
        targetContract: _contract,
        selector: _selector
    });
    emit RandomRequested(_contract, _selector, _index);
}
```

As shown above, the `_index` argument is calculated as `uint256(uint160(useraddress))` which means multiple requests from the same user will have the same value. This will lead to the same mapping storage slot being written to, overwriting previous request data. Once a pending request has been overwritten, the NFT associated with that request will be permanently lost, due to the following check in the `_callback` function:

```
function _callback(uint256 _index, bytes calldata _rand) internal {
    RequestData storage data = requests[_index];
    // @audit Only the latest non-overwritten reveal will work, then that slot is deleted
    if (data.targetContract == address(0)) revert NoRequestError();
    // ...
    delete requests[_index];
}
```

Recommendation(s): Consider calculating `_index` as the hash of the address and another unique field that will allow for unique keys for reveals that came from the same address. For the API3 `RNGProxy` this could be the `requestID`. Alternatively to ensure compatibility with the self-hosted `RNGProxy` a nonce system could be implemented in the base RNG implementation instead.

Status: Fixed

Update from the client: PR #179, commit hash 5d918ff. We prepend a nonce. The index for rng (self hosted and api3) is a uint256 but we only need 160 bits for the address so nonce is prepended when converting address to uint256 and then silently discarded when it is converted back to a uint160.

7.10 [Medium] Player can pet their own Munchables

File(s): SnuggeryManager.sol

Description: Players can call the function `pet` in order to pet another player's Munchable, giving both them and the recipient Schnibbles in the process. The function will recover the main account of the caller, and check the owner of the munchable to be pet to ensure that the caller is not the owner of the Munchable. This is done to prevent petting your own Munchable, however the check done is `owner == msg.sender` instead of `owner == _mainAccount`. As a result, it is possible to pet your own Munchables through a sub-account.

```
function pet(uint256 _tokenId) external notPaused {
    (
        address _mainAccount,
        MunchablesCommonLib.Player memory _player
    ) = _getMainAccountRequireRegistered(msg.sender);

    // @audit owner is the main account of the NFT's owner
    address owner = IERC721(munchNFT).ownerOf(_tokenId);

    (
        address _owner,
        MunchablesCommonLib.Player memory _ownerPlayer
    ) = _getMainAccountRequireRegistered(owner);

    // ...

    // @audit Should check that owner == _mainAccount
    //        Caller could be subaccount and will pass check
    if (owner == msg.sender) revert CannotPetOwnError();
    // ...
}
```

Recommendation(s): Consider changing the check to `msg.sender == _mainAccount` to prevent player's from petting their own Munchables via a subaccount.

Status: Fixed

Update from the client: Fixed in commit `2ece43e5144a6028873ff0d816d99bae77a1acea`.

7.11 [Medium] Player lock durations can be less than the default minimum

File(s): LockManager.sol

Description: The function `setLockDuration(...)` allows players to set the lock duration and updates existing locks. However, it does not check whether the new duration is at least the minimum period defined in `lockdrop.minLockDuration`. The function is presented below:

```
function setLockDuration(uint256 _duration) external notPaused {
    if (_duration > configStorage.getUint(StorageKey.MaxLockDuration))
        revert MaximumLockDurationError();

    // @audit Missing check if new duration is less than lockdrop.minLockDuration
    playerSettings[msg.sender].lockDuration = uint32(_duration);
    uint256 configuredTokensLength = configuredTokenContracts.length;
    // ...
}
```

The function `_lock(...)` uses the `lockDuration` defined by the player when players directly lock tokens or migrate. If the player did not set the lock duration, the minimum lock duration (`lockdrop.minLockDuration`) is used instead. Since the function `setLockDuration` doesn't prevent a player's custom duration from being less than the minimum, a player can use their own duration to unlock before the minimum time period.

```
function _lock(...) private {
    // ...

    // @audit User can have their own lock duration non-zero but less than minimum
    uint32 _lockDuration = playerSettings[_lockRecipient].lockDuration;
    if (_lockDuration == 0) {
        _lockDuration = lockdrop.minLockDuration;
    }

    // ...
}
```

Recommendation(s): Consider adding a check to `setLockDuration` to prevent a duration less than `drop.minLockDuration`.

Status: Fixed

Update from the client: Fix in PR #179 commit hash `39e31d6`. I did not use the suggested fix because let's say someone locked their tokens for 45 days but wanted to increase it to 60 and the min lock duration was 30... Then it would break. Instead, I modified the if statement here such that the migrationManager check happens after the lock duration check.

7.12 [Medium] Players can mint multiple munchables for the same primordial

File(s): src/manager/PrimordialManager.sol

Description: Players call `hatchPrimordialToMunchable(...)` once their primordials reach level 0 to swap for an NFT. The function checks whether the primordials exist and are ready to hatch.

```
function hatchPrimordialToMunchable() external notPaused {
    (address _caller, ) = _getMainAccountRequireRegistered(msg.sender);
    // @audit The function does not check if the primordial has already hatched
    if (primordials[_caller].createdDate == 0)
        revert PrimordialDoesntExistError();
    if (primordials[_caller].level < 0) revert PrimordialNotReadyError();

    primordials[_caller].hatched = true;

    nftOverlord.mintFromPrimordial(_caller);

    emit PrimordialHatched(_caller);
}
```

The function does not check whether the primordial has already hatched. As a result, the user can create multiple NFTs from the same primordial.

Recommendation(s): Ensure the primal isn't hatched before creating a new NFT and marking the `hatched = true` attribute.

Status: Fixed

Update from the client: Fixed in commit `487479d50c8685949485996bbd2873416b5704df`.

7.13 [Medium] Players can reduce their asset unlock time

File(s): LockManager.sol

Description: The function `setLockDuration(...)` allows players to set their asset lock duration. The function ensures that `block.timestamp + _duration` is less than `unlockTime`, ensuring players are not setting the lock duration shorter than the current `unlockTime`.

```
function setLockDuration(uint256 _duration) external notPaused {
    // ...
    for (uint256 i; i < configuredTokensLength; i++) {
        address tokenContract = configuredTokenContracts[i];
        if (lockedTokens[msg.sender][tokenContract].quantity > 0) {
            // Check they are not setting lock time before current unlocktime
            // @audit Check with block.timestamp, but the update is lastLockTime + _duration
            if (
                uint32(block.timestamp) + uint32(_duration) <
                lockedTokens[msg.sender][tokenContract].unlockTime
            ) {
                revert LockDurationReducedError();
            }
            uint32 lastLockTime = lockedTokens[msg.sender][tokenContract]
                .lastLockTime;
            lockedTokens[msg.sender][tokenContract].unlockTime =
                lastLockTime +
                uint32(_duration);
        }
    }
    // ...
}
```

The update to `unlockTime` is calculated by adding the `lastLockTime` with the duration instead of the current block timestamp. This allows a user to reduce their `unlockTime` to be shorter than before, since `lastLockTime < block.timestamp < unlockTime`. Users may be able to call this function multiple times to shorten their lock duration a significant amount. Consider the example below:

Let be new `_duration = 70 days`:

```
block.timestamp = 1716410961 // current timestamp
lastLockTime = uint32(block.timestamp) - uint32(20 days); // 20 days ago
unlockTime =  lastLockTime + uint32(90 days); // Current unlocktime. This current lock period is 90 days
```

Considering the current timestamp in this example, the last lock time was 20 days ago for a locking period of 90 days; we have:

```
lastLockTime = 1714682961
unlockTime   = 1722458961
```

The checking below will pass:

```
uint32(block.timestamp) + uint32(_duration) <
            lockedTokens[msg.sender][tokenContract].unlockTime
```

```
uint32(block.timestamp) + uint32(_duration) = 1722458961
```

The function updates `unlockTime` to:

```
unlockTime = 1720730961
```

The player could reduce in 20 days their locking period.

Recommendation(s): Ensure the new lock duration will not lead to the unlock time being reduced, by changing the check to rely on `lastLockTime` instead of `block.timestamp`.

Status: Fixed

Update from the client: Resolved in PR #1 of munchable-common-core-latest hash `0b12df2`.

7.14 [Medium] The function `getSnuggery` reverts with the index out of bounds error

File(s): SnuggeryManager.sol

Description: The function `getSnuggery(...)` returns token ids and their respective imported dates for a given range defined by the parameter `_start` and the local variable `maxSize`. The function can return 10 tokenIds at most.

```
function getSnuggery( address _account, uint256 _start)
    external view returns (
        MunchablesCommonLib.SnuggeryNFT[] memory _snuggery,
        uint256 _snuggerySize
    )
{
   // ...
    _snuggerySize = snuggeries[_player].length;
    uint256 maxSize = _snuggerySize;
    if (maxSize > 10) {
        maxSize = 10;
    }
    // ..
}
```

The issue arises when `_start` is greater than `five` and the maximum length for the array `_snuggery` is `10`. The variable `i` reaches values out of bounds of the `_snuggery` when the variable `_snuggerySize > 10`.

```
    // @audit the maximum length for _snuggery is 10
    _snuggery = new MunchablesCommonLib.SnuggeryNFT[](maxSize);
    for (uint256 i = _start; i < maxSize + _start; i++) {
        if (i >= _snuggerySize) break;

        MunchablesCommonLib.SnuggeryNFT memory snuggeryNFT = snuggeries[
            _player
        ][I];
        // @audit if i greater than 10, we have index out of bounds error
        _snuggery[i].tokenId = snuggeryNFT.tokenId;
        _snuggery[i].importedDate = snuggeryNFT.importedDate;
    }
}
```

Recommendation(s): Start adding munchables information from index zero in the array `_snuggery`.

Status: Fixed

Update from the client: Resolved here to remove pagination: `63906fc8366dd48bf17bcdff151059cf32c53594`, `1e6d63c67e967ab000036aaf4b7151299158b2fe`.

7.15 [Low] Any user can access the full migration bonus

File(s): MigrationManager.sol

Description: Users can selectively migrate their purchased Munchable NFTs using the function `migratePurchasedNFTs`, passing an array of token IDs which they would like to migrate. After migration is complete the user is eligible for a migration bonus, which increases the amount of Schnibbles that the user earns on a daily basis. It is possible for a user who is not included in the migration snapshot to still access the migration bonus, by calling the `migratePurchasedNFTs` function with an empty `tokenIds` array. The for loop containing input validation is skipped since the array is empty, and the internal function `_migrateNFTs` is called.

```
function migratePurchasedNFTs(
    uint256[] memory tokenIds
) external payable override nonReentrant {
    MigrationSnapshotData memory snapshot;
    uint256 quantity;
    // @audit This loop and its validations are skipped entirely
    for (uint256 i = 0; i < tokenIds.length; i++) {
        // ...
    }
    _migrateNFTs(msg.sender, address(0), tokenIds);
    if (msg.value != (quantity * discountFactor) / 10e12)
        revert InvalidMigrationAmountError();
}
```

Inside the `_migrateNFTs` function, the mapping `_userClaimedOnce[msg.sender]` is set to true, then more token validation logic nested inside a loop is skipped. It will calculate the amount of assets to be locked in the `LockManager` as zero, so a lock with zero assets will be completed. In a later transaction when the user attempts to harvest, `BonusManager.getHarvestBonus` will be called which includes a migration bonus. Since the `_userClaimedOnce` mapping now points to true for the user, they will receive a migration bonus. A snippet from the migration bonus calculations are shown below:

```
function _calculateMigrationBonus(
    address _caller,
    uint256 weightedValue
) internal view returns (uint256 _migrationBonus) {
    (
        bool didMigrate, // This will be true
        IMigrationManager.MigrationTotals memory totals
    ) = _migrationManager.getUserMigrationCompletedData(_caller);
    ILockManager.ConfiguredToken memory configuredToken = _lockManager
        .getConfiguredToken(totals.tokenLocked);
    if (didMigrate) {
        uint256 usdPrice = configuredToken.usdPrice;
        // `totals.totalLockedAmount` is zero, so `migrateHighestAmount` is zero
        uint256 migrateHighestAmount = (2 * totals.totalLockedAmount * usdPrice);
        uint256 halfAmount = (totals.totalLockedAmount * usdPrice) / 2;
        // `migrateHighestAmount` is now always zero, this if statement is a tautology
        // Will always enter this if statement, giving a full bonus
        if (weightedValue >= migrateHighestAmount) {
            // Full bonus
            _migrationBonus = migrationBonus;
        } else if (weightedValue >= halfAmount) {
            // ...
        }
    }
}
```

Since `didMigrate` is true but no tokens were actually migrated, the returned `totals` data is still empty so the `totals.totalLockedAmount` will be zero, leading to `migrateHighestAmount` always being zero. The first if statement will always be entered, granting a full bonus to users even if they did not hold any previous Munchables and were not included in the migration snapshot.

Recommendation(s): Consider adding a check in `BonusManager._calculateMigrationBonus` to set the migration bonus to zero if `totals.totalLockedAmount` is zero. Also consider preventing an array length of zero in the function `MigrationManager.migratePurchasedNFTs`, and as an extra precaution prevent any user from locking zero assets in `LockManager`.

Status: Fixed

Update from the client: People are only eligible for the migration bonus when they transfer over locked NFTs, not purchased ones. Because of this, we moved the _userClaimedOnce check to the lockFundsForAllMigration at commit hash 0ececce.

7.16 [Low] Function `getFullPlayerData` may not return accurate Snuggery data

File(s): `AccountManager.sol`

Description: The function `BonusManager._calculateLevelBonus` is used to calculate the additional harvest bonus a player should receive based on the total level of the Munchables in their Snuggery. To calculate the total, the function `AccountManager.getFullPlayerData` is called which in turn calls `Snuggery.getSnuggery` to get the number of Munchables currently in the Snuggery and the token ID of each.

```
function _calculateLevelBonus(...) internal view returns (...) {
    (
        ,
        MunchablesCommonLib.Player memory player,
        MunchablesCommonLib.SnuggeryNFT[] memory _snuggery,
        uint256 _snuggerySize
    ) = _accountManager.getFullPlayerData(_caller);

    if (_snuggerySize > 0) {
        uint i;
        for (; i < _snuggerySize; i++) {
            _levelBonus += _nftAttributesManager.getAttributes(_snuggery[i].tokenId).level;
        }
        // ...
    }
}
```

The function `Snuggery.getSnuggery` is limited to return a maximum of 10 Munchables per call, as it is intended to be called multiple times with an incrementing `_start` input in order to collect data on all Munchables in the Snuggery. When used by `AccountManager.getFullPlayerData` it only calls the function once and does not consider that there may be more than 10 Munchables. For bonus calculations, this means that users with more than 10 Munchables in their Snuggery will not receive the full rewards they would expect.

Recommendation(s): Consider applying one of the recommendations below:

1 - Consider refactoring the function `Snuggery.getSnuggery`. This refactoring would consist of extracting a private function `C` where `Snuggery.getSnuggery` would call it. This function `C` would loop the player's snuggery according to parameters `start` and `end`, which could represent a pagination or the whole snuggery. Then, `AccountManager.getFullPlayerData` would call function `C` to get the entire player's snuggery.

2 - Consider changing `AccountManager.getFullPlayerData` to call `Snuggery.getSnuggery` with increasing `_start` inputs until all Munchables have been retrieved.

Status: Fixed

Update from the client: Resolved here to remove pagination:
`63906fc8366dd48bf17bcdff151059cf32c53594`, `1e6d63c67e967ab000036aaf4b7151299158b2fe`.

7.17 [Low] MunchNFT contract's pause mechanism is unreachable

File(s): `MunchNFT.sol`

Description: The `MunchNFT` contract derives from `ERC721Pausable` implementation but does not expose the internal `_pause` and `_unpause` functions. This means it is not possible to access the pausable features of the token.

Recommendation(s): Add an external function allowing a permissioned user to pause and unpause the contract. Also, consider adding the `whenNotPaused` modifier to the `mint` function to ensure that when the contract is paused, minting is also prevented.

Status: Fixed

Update from the client: Fix in PR request #11 (we use the ConfigStorage Pause): https://github.com/munchablesorg/munchables-common-core-latest/pull/11/files

7.18 [Low] The last entry in a user's sub-account array cannot be removed

File(s): AccountManager.sol

Description: When removing a sub-account in the function `removeSubAccount`, an array of all sub-accounts are iterated through until the matching address is found. Once found, all following elements in the array are shuffled to overwrite the address to be removed. The function is shown below:

```
function removeSubAccount(
    address _subAccount
) external notPaused onlyRegistered(msg.sender) {
    delete mainAccounts[_subAccount];
    uint256 subAccountLength = subAccounts[msg.sender].length;
    bool found = false;
    for (uint256 i = 0; i < subAccountLength - 1; i++) {
        if (subAccounts[msg.sender][i] == _subAccount || found) {
            subAccounts[msg.sender][i] = subAccounts[msg.sender][i + 1];
            found = true;
        }
    }
    if (!found) revert SubAccountNotRegisteredError();
    subAccounts[msg.sender].pop();
    // ...
}
```

The for loop in `removeSubAccount` will execute until the second last index of the array, after which it will stop. This means the last element in the array will never be "found", and the function will revert. Consider an example with an array size of 3, the loop will terminate at `i < arr.len - 1` which would be `i < 2`. In this case the loop would only execute twice with `i=0,1`, the third element won't be checked.

Recommendation(s): Provided that sub-account array ordering does not matter, every element in the array can be checked, and once found a swap-then-pop approach can be used instead of shuffling each element.

```
function removeSubAccount(...) external /* ... */ {
    // ...
    bool found = false;
    // @audit Replace the _subAccount to be removed by the one in the last position
    for (uint256 i = 0; i < subAccountLength; i++) {
        if (subAccounts[msg.sender][i] == _subAccount) {
            subAccounts[msg.sender][i] = subAccounts[msg.sender][subAccountLength-1];
            found = true;
            subAccounts[msg.sender].pop();
            break;
        }
    }
    if (!found) revert SubAccountNotRegisteredError();

    emit SubAccountRemoved(msg.sender, _subAccount);
}
```

Status: Fixed

Update from the client: PR #179 commit hash 215bcee. Implemented recommendation.

7.19 [Low] Thresholds can be updated while a proposal is in progress

File(s): `LockManager.sol`

Description: The `admin` invokes the function `setUSDThresholds(...)` to update the thresholds `APPROVE_THRESHOLD` and `DISAPPROVE_THRESHOLD` .

```
function setUSDThresholds(
    uint8 _approve,
    uint8 _disapprove
) external onlyAdmin {
    // @audit it should check if there not proposal in progress
    APPROVE_THRESHOLD = _approve;
    DISAPPROVE_THRESHOLD = _disapprove;

    emit USDThresholdUpdated(_approve, _disapprove);
}
```

However, if the `admin` by mistake calls this function to update the thresholds while a proposal is in progress, this would directly impact the number of approvals and disapprovals to update the USD price.

Recommendation(s): Add the following check below in the function `setUSDThresholds(...)`:

```
+if (usdUpdateProposal.proposer != address(0))
+revert ProposalInProgressError();
```

Status: Fixed

Update from the client: Fixed in commit hash d88305b

7.20 [Low] User can set their own address as a referrer

File(s): `AccountManager.sol`

Description: When calling the `register` function as a new user, you can specify a referral address which will receive some additional points whenever the `ClaimManager._claimPoints` is called. It is possible for a user to do a "self referral" by passing their own address into this function to receive extra points.

```
function register(
    MunchablesCommonLib.Realm _snuggeryRealm,
    address _referrer
) external onlyUnregistered(msg.sender) {
    // @audit User can pass their own address as the referral
    //        This will grant them additional undeserved points

    if (mainAccounts[msg.sender] != address(0))
        revert SubAccountCannotRegisterError();
    if (_snuggeryRealm >= MunchablesCommonLib.Realm.Invalid)
        revert InvalidRealmError();

    // ...
}
```

Recommendation(s): Consider adding a check to prevent players from using their own address as a referral.

Status: Fixed

Update from the client: Fix in PR #179 at commit hash `9aaff75`.

7.21 [Info] A price proposal can be both approved and disapproved

File(s): LockManager.sol

Description: The function approveUSDPrice allows an authorized address to voter for or against a price proposal. A series of validations are done before the vote is counted, including a check to ensure that the caller hasn't already voted for the proposal before. However, it fails to validate that the voter has not voted against the proposal before.

```
function approveUSDPrice(uint256 _price) external onlyOneOfRoles(...)
{
        /// ...
        /// @audit Check that the voter has not voted in favor of the proposal before
        if (usdUpdateProposal.approvals[msg.sender]) revert ProposalAlreadyApprovedError();

        /// ...

        /// @audit Counts the vote as valid
        usdUpdateProposal.approvals[msg.sender] = true;
        usdUpdateProposal.approvalsCount++;
        /// ...
}
```

This would allow a voter to call the function disapproveUSDPrice to vote against a proposal, and then call approveUSDPrice to vote again, but this time in favour of the proposal.

Recommendation(s): Consider adding a check to ensure the proposal has not been previously voted against, similar to what is done in disapproveUSDPrice.

Status: Fixed

Update from the client: Fixed in commit hash 06caf27.

7.22 [Info] Double approval account check when approving USD price

File(s): LockManager.sol

Description: The function approveUSDPrice increments the approvals count with each valid approval, and checks if the approval count meets or exceeds the APPROVAL_THRESHOLD. If the approval count is met then the price is updated by calling _execUSDPriceUpdate. Then, a duplicate approvals count check is done inside _execUSDPRiceUpdate, as shown below:

```
function approveUSDPrice(uint256 _price) external onlyOneOfRoles(...) {
    //...
    // @audit Approval count is checked
    if (usdUpdateProposal.approvalsCount >= APPROVE_THRESHOLD) {
        _execUSDPriceUpdate();
    }
    //...
}

// @audit This can only be called if approvalsCount >= APPROVE_THRESHOLD
function _execUSDPriceUpdate() internal {
    if (
        // @audit Approval count is checked again
        usdUpdateProposal.approvalsCount >= APPROVE_THRESHOLD
            && usdUpdateProposal.disapprovalsCount < DISAPPROVE_THRESHOLD
    ) {
        // ...
    }
}
```

Recommendation(s): Consider removing one of these checks, so the approval count is only checked once.

Status: Acknowledged

Update from the client: Acknowledged.

7.23 [Info] Duplicate notifiable addresses and handled differently in add/removal

File(s): ConfigStorage.sol

Description: The ConfigStorage contract has a notifiableAddresses storage array containing a list of addresses that will be notified for a configuration update upon storage changes. The functions addNotifiableAddress, addNotifiableAddresses, and removeNotifiableAddress are responsible for managing the addresses in this array.

The behaviour of these functions differs slightly however, when adding addresses it is possible to include duplicate entries that already exist. When removing addresses, handling of duplicates is not supported and the function will exit upon removal of the first entry, even if there are duplicate addresses in the array. Consequently, a contract may remain susceptible to updates via configUpdated, leading to unexpected behavior.

Recommendation(s): Consider changing removeNotifiableAddresses to not exit early on the first removal. This will improve consistency between the add and remove functions, where duplicates are supported for both.

Status: Acknowledged

Update from the client: Acknowledged

7.24 [Info] Non configured but active tokens result in division by zero

File(s): LockManager.sol

Description: The functions lockOnBehalf and lock allow users to lock their tokens. They use the modifier onlyActiveToken to ensure that the given token is active before it can be locked. However, these functions do not check that the given token is configured. If a token is active but not configured, a division by zero error can occur, causing asset lock attempts to revert. Relevant function snippets are shown below:

```
// @audit This function accepts unconfigured tokens
function lockOnBehalf(...) external payable notPaused onlyActiveToken(_tokenContract) nonReentrant {
    // ...

    _lock(
        _tokenContract,
        _quantity,
        tokenOwner,
        lockRecipient,
        _lockDuration
    );
}

// @audit This function accepts unconfigured tokens
function lock(...) external payable notPaused onlyActiveToken(_tokenContract) nonReentrant {
    _lock(_tokenContract, _quantity, msg.sender, msg.sender, _lockDuration);
}

function _lock(...) private {
    // ...

    // @audit Division by zero if nftCost == 0, i.e., non configured token
    remainder = quantity % configuredToken.nftCost;
    numberNFTs = (quantity - remainder) / configuredToken.nftCost;

    // ...
}
```

Recommendation(s): Consider adding the modifier onlyConfiguredToken to the functions lock and lockOnBehalf.

Status: Fixed

Update from the client: Used your suggestion in this PR: https://github.com/munchablesorg/munchables-common-core/pull/164.

7.25 [Info] Primordials can be fed after hatching

File(s): src/managers/PrimordialManager.sol

Description: A player invokes `feedPrimordial` to feed their primordials to increase their `chonks`. When the primordial reaches level 0, it can be swapped for a Munchable. It is possible to call `feedPrimordial` multiple times after the primordial has already hatched, and the players schnibbles will be consumed, increasing it's chonks without further increasing the level.

```
function feedPrimordial(
    uint256 _schnibbles
) external notPaused onlyPrimordialsEnabled {
    (
        address _mainAccount,
        MunchablesCommonLib.Player memory _player
    ) = _getMainAccountRequireRegistered(msg.sender);

    if (_player.unfedSchnibbles < _schnibbles)
        revert InsufficientSchnibblesError(_player.unfedSchnibbles);

    if (primordials[_mainAccount].createdDate == 0)
        revert PrimordialDoesntExistError();
    // @audit No check if primordial already hatched
    //        Can feed already hatched primordial, and schnibbles will be spent
    primordials[_mainAccount].chonks += _schnibbles;
    // ...
}
```

Recommendation(s): Consider adding a check to prevent feeding if the player's primordial has already hatched.

Status: Fixed

Update from the client: Fixed in commit `f001ee9fa6c8f090093290e6f752e1d314226114`.

7.26 [Info] Redundant loop in function addSubAccount

File(s): AccountManager.sol

Description: In the function `addSubAccount` a loop is used to check that the sub-account address to be added is not already a sub account for the caller. This loop is not necessary, as a previous check makes sure that the given `_subAccount` doesn't have an associated main account, as shown below:

```
function addSubAccount(
    address _subAccount
) ... {
    // @audit This check makes sure the sub account doesn't have a main account
    if (mainAccounts[_subAccount] != address(0))
        revert SubAccountAlreadyRegisteredError();
    mainAccounts[_subAccount] = msg.sender;

    // @audit Sub account already doesn't have a main account
    //        This check is unnecessary
    for (uint256 i; i < subAccounts[msg.sender].length; i++) {
        if (subAccounts[msg.sender][i] == _subAccount)
            revert SubAccountAlreadyRegisteredError();
    }
    // ...
}
```

Recommendation(s): Consider removing the unnecessary loop.

Status: Acknowledged

Update from the client: Acknowledged

7.27 [Info] The same spray proposal can be executed multiple times

File(s): AccountManager.sol

Description: The function `execSprayProposal` allows accounts with the "Social Approval" role to execute proposals that will distribute Schnibbles to users. Once a proposal has been executed however, the proposal data is not cleared, allowing the same proposal to be executed multiple times. A snippet from the function is shown below:

```
function execSprayProposal(address _proposer) external onlyOneOfRoles([...]) {
    uint256 numberEntries = sprayProposals[_proposer].squirts.length;
    for (uint256 i; i < numberEntries; i++) {
        /// ...

        if (players[player].registrationDate != 0) {
            players[player].unfedSchnibbles += schnibbles;
        } else {
            unclaimedSchnibbles[player] += schnibbles;
        }
    }
    /// @audit Upon completion, the proposal not deleted
}
```

Recommendation(s): Consider deleting proposal data after the proposal has been executed in `execSprayProposal`.

Status: Fixed

Update from the client: Resolved here

7.28 [Info] Unnecessary storage write when player chonks haven't changed

File(s): managers/SnuggeryManager.sol

Description: The function `_recalculateChonks` recalculates the player chonks and will update the `totalGlobalChonk` if a difference has been observed. As shown below, the `playerChonks` mapping entry is updated every time, even when `previousChonks` is equal to `_playerChonks`:

```
function _recalculateChonks(address _player) internal {
    uint256 previousChonks = playerChonks[_player];
    uint256 _playerChonks;
    for (uint256 i; i < snuggeries[_player].length; i++) {
        _playerChonks += nftAttributesManager
            .getAttributes(snuggeries[_player][i].tokenId)
            .chonks;
    }

    //@audit This update can be moved into the if branch below to save gas
    playerChonks[_player] = _playerChonks;

    if (previousChonks != _playerChonks) {
        totalGlobalChonk += (_playerChonks - previousChonks);
    }
}
```

Recommendation(s): Consider updating the code to change `playerChonks` only if `previousChonks` != `_playerChonks`, since if there is no difference the storage write will be unnecessary.

Status: Acknowledged

Update from the client: Acknowledged

7.29 [Info] Wrong error message in `burnUnrevealedForPoints(...)`

File(s): `MigrationManager.sol`

Description: The function `burnUnrevealedForPoints` can only be called when the migration data is sealed, and will revert otherwise. However, revert error used is `MigrationDataSealedError`, where `MigrationDataNotSealedError` should be used instead, as shown below:

```
function burnUnrevealedForPoints() external {
    // @audit Error should be MigrationDataNotSealedError
    if (!seal) revert MigrationDataSealedError();
    // ...
}
```

Recommendation(s): Consider replacing the `MigrationDataSealedError(...)` error with `MigrationDataNotSealedError(...)`.

Status: Fixed

Update from the client: Fixed at commit hash d88305b.

7.30 [Info] tokenLocked can be overwritten during migration snapshot loading

File(s): `MigrationManager.sol`

Description: The function `loadMigrationSnapshot` is called by the admin to load migration snapshots for multiple users. The function is presented below:

```
function loadMigrationSnapshot(
    address[] calldata users,
    MigrationSnapshotData[] calldata data
) external override onlyUniversalRole(Role.Admin) {
        // ...
        if (data[i].lockAmount != 0) {
            _userLockedAmounts[_user].totalLockedAmount += data[I]
                .lockAmount;
            // @audit tokenLocked can be overwritten if users contains
            //        more than one occurrence of the same user address
            //        and different token (USDB or WETH)
            _userLockedAmounts[_user].tokenLocked = data[i].token;
        } else {
        // ...
}
```

The function validates if the tokens in the struct `data` are `USDB`, `WETH`, or `zero` address for `ETH`. it also validates if the migration snapshot for a particular `user` and `tokenId` is not loaded yet. However, the `loadMigrationSnapshot(...)` does not check if `tokenLocked` is equal to `data[i].token` (in case the `tokenLocked` is already set) when `data[i].lockAmount != 0`. A particular user may occur multiple times in the array `users` or can be passed to the function in different invokes. In these both scenarios, if `_userLockedAmounts` is already set for a particular user, then the `tokenLocked` will be overwritten with a different token address and the `totalLockedAmount` will be incremented for different tokens.

Recommendation(s): Ensure `_userLockedAmounts[_user].tokenLocked` is equal to `data[I].token` when it is already set.

Status: Acknowledged

Update from the client: Considered, but all data coming in will be one token type per user so this can be ignored.

7.31 [Best Practices] Common storage does not have provision for additional features in future

File(s): `BaseConfigStorage.sol`

Description: `BaseConfigStorage` is a storage location for generic functionality of the protocol. As of now, only pause and reference to interface for business specific storage is stored. As the protocol grows, there may be a need to add more state variable for generic functionality of the protocol. Currently, there is no gap provisioned in `BaseConfigStorage` contract for any additional state variables needed to support new generic/common functionality for the protocol in future.

Recommendation(s): Consider adding a storage gap to create reserve space in the storage layout for future functionality: `uint256[20] __gap`. The gap size only serves as a suggestion.

Status: Acknowledged

Update from the client: Acknowledged

7.32 [Best Practices] Lock pragmas to specific compiler version

File(s): src/*

Description: The smart contracts in this project use different pragma versions. Contracts should be deployed with the same fixed compiler version to ensure consistency between testing and production environments. The following is a list of the varying pragma versions throughout the codebase:

```
pragma solidity 0.8.25;
pragma solidity ^0.8.0;
pragma solidity ^0.8.25;
```

Recommendation(s): Consider using the same version for all contracts in the codebase.

Status: Fixed

Update from the client: Fixed at commit hash 9674508.

7.33 [Best Practices] Migration data token ID should not be zero

File(s): MigrationManager.sol

Description: The function `loadMigrationData` is called by the admin to set migration data which will be used to lock and convert previous Munchables over to the new implementation. The token ID of a Munchable available to be migrated is part of the migration data, and can be set to any value by the admin, including zero. However, in other functions such as `_migrateNFTs` a token ID of zero is treated differently, causing the function to exit early or revert in some cases, as shown below:

```
// Function burnNFTs
if (snapshot.tokenId == 0) revert NoMigrationExistsError();
// Function burnRemainingPurchasedNFTs
if (snapshot.tokenId == 0) revert NoMigrationExistsError();
// Function _migrateNFTs
if (tokenId == 0) continue;
if (snapshot.tokenId == 0) revert NoMigrationExistsError();
```

As a general best practice it should not be possible to set valid data to the same value that is reserved for "empty", in this case, a token ID of zero as it may lead to unexpected behavior such as locked migration funds.

Recommendation(s): Consider adding a check to `loadMigrationData` to prevent the `MigrationSnapshotData.tokenId` fields from being zero.

Status: Fixed

Update from the client: Fixed in PR #179 at commit hash `7cd19ec`.

7.34 [Best Practices] Missing `_disableInitializers(...)` function in upgradeable contracts

File(s): src/*

Description: The contracts `AccountManager` and `ClaimManager` are designed to be upgradeable, inheriting from the OpenZeppelin Initializable abstract contract implementation. A security recommendation issued by OpenZeppelin is to lock the initialization of the implementation contracts after deployment. This will any users from being able to execute the `initialize` function and potentially take control of the implementation contract.

Recommendation(s): Consider following OpenZeppelin's best practices by invoking the `_disableInitializers` function in the contract's constructor. This action will automatically lock the contract upon deployment, thereby preventing the implementation contract from being controlled.

Status: Fixed

Update from the client: Fix in PR #18: https://github.com/munchablesorg/munchables-common-core-latest/pull/18

7.35 [Best Practices] Redundant inheritance

File(s): BonusManager.sol

Description: The BonusManager contract has an explicit inheritance, which is already implicitly included in a parent contract. Consider the example shown below:

```
// BonusManager inherits BaseConfigStorage
// However BaseConfigStorage is already inherited via BaseBlastManager

contract BonusManager is BaseConfigStorage, BaseBlastManager {...}
abstract contract BaseBlastManager is BaseConfigStorage {...}
```

Recommendation(s): Consider evaluating the redundant inheritance from BonusManager.

Status: Fixed

Update from the client: Fixed at d302fda.

7.36 [Best Practices] Unused imports

File(s): RewardsManager.sol

RewardsManager.sol

Description: The codebase contains some unused code, affecting overall readability and code quality. A list of files and their unused declarations are shown below:

Contract RewardsManager.sol

```
import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "../interfaces/IDistributor.sol";
import "../interfaces/IClaimManager.sol";
import "../interfaces/IConfigStorage.sol";
import "../interfaces/IBlast.sol";
import "../interfaces/IConfigNotifiable.sol";
```

Contract LockManager.sol

```
import "../interfaces/IConfigStorage.sol";
```

Recommendation(s): To keep the code clean and readable, consider evaluating if the unused functionality should be used and, if not, remove it from the codebase.

Status: Acknowledged

Update from the client: Acknowledged

7.37 [Best Practices] Use of legacy call to transfer native tokens

File(s): FundTreasuryDistributor.sol

Description: The function receiveTokens is using transfer() to move native assets to the treasury account. The transfer function operates with a 2300 gas limit. As gas costs are subject to change, it is not recommended to rely on a fixed gas cost native asset transfer mechanism.

Recommendation(s): Consider using the low level call() function to transfer native assets instead of transfer().

Status: Fixed

Update from the client: Fixed at commit hash 9c8e121.

8 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about the Munchable Protocol documentation

The documentation for Munchables is contained in the project's Github directory guides. It consists of five files with:

- Explanation of how the game works.
- General overview of the system.
- Description of the features.
- Logic and structural flow.

The information in the documentation is concise and technical, with well-written explanations. **Code comments are also of high quality**, with descriptions for every function explaining the context and situations where the function is called. There are comments in other parts of the code where extra information is needed, allowing readers to comprehend the code faster. **During the audit, the Munchables Team had effective communication**. They were available for meetings to answer questions, explain further, and discuss the detected issues. In addition, they were consistently accessible, ensuring that our team could seek and receive timely clarifications and support as needed.

9 Test Suite Evaluation

9.1 Contracts Compilation

```
% pnpm build

> munchables-common-core@1.0.0 build /munchables-common-core
> pnpm build:solidity && pnpm build:abi


> munchables-common-core@1.0.0 build:solidity /munchables-common-core
> forge build

[] Compiling...
[] Compiling 147 files with 0.8.25
[] Solc 0.8.25 finished in 9.23s
Compiler run successful!

> munchables-common-core@1.0.0 build:abi /munchables-common-core
> pnpm wagmi generate

+ Validating plugins
+ Resolving contracts
+ Running plugins
+ Writing to abi/generated.ts
```

9.2 Tests Output

```
> munchables-common-core@1.0.0 test /munchables-common-core
> pnpm run test:typescript && pnpm run test:solidity


> munchables-common-core@1.0.0 test:typescript /munchables-common-core
> node --import tsx tests/run.ts

Running tests that match **/*.test.ts
Deploying test contracts...
Done! Transaction hash:  0xf5bc0b5628098f79849e3489242686e497cc6832da385e001212ad088775bc70
Done! Transaction hash:  0xe70142491eae9810dfae8af91e403fbc88e70512452e40f403f7f4e71645a451
Done! Transaction hash:  0xfe8a76e5b6fd1216899aff90a685b71e7ddb46bacb33a4a9c0f44cf358bd56fa
Done! Transaction hash:  0x92e0810d37fc48f6d0ec2a97d4b9154cdc4a03e4aa51273445069f212d28a312
Done! Transaction hash:  0xe66b9752fee77b88664d83701e298231dc9518b0115248aefab32a066c5c8325
Done! Transaction hash:  0xa8001923031f36209a31153762fe5e84689dc34a12ab8ab7ce1e52042ab8571a
Done! Transaction hash:  0x57d630c5dd2125bfa6285b2a5993d72b24ff604dc34bdcefaf4293a2a7641b21
Done! Transaction hash:  0x3e784ff5c80e64dccf02a833d7628a833351ba4151a21ba0d724cc1d61e39cc7
Done! Transaction hash:  0x474bbfa5a13c8d2c28f0e2ec734995a7db0598bef90075ef83125c1ce3e679e8
Done! Transaction hash:  0xf1321402a900616fbd776fe2425eba7e0d8bad1532aab6d75be40b1df038084d
Done! Transaction hash:  0xa680121f89ece0d4ba8b0ed08227ed38e9c56a58a9ba09b3561714d7708daca7
Done! Transaction hash:  0x632c8cca58b009a29d2ed9244cbd7f722fa3230c8aaab58ab07d64d5faf018a7
Done! Transaction hash:  0xbd4b5c770b1fd331602c861d67f6cec57855ff775cc8a7799b8f3a846bf5da4d
Done! Transaction hash:  0xe97b437858e23762eb7d30c4d3512b2c0741f69cc9db8f50245771be54b782a5
Done! Transaction hash:  0xd87bc28cd284bcc49defb503939bfb38b2a1f883aabba852b587e3f76d1a4445
Done! Transaction hash:  0x567e941d374da69c6b72ecfdad5f918ab733d7a707d1e755f89d02e4488a1a2b
Done! Transaction hash:  0x61e5f37a32249cd62227b05bb546554f7023b7f7ebac9eab4e118bd40c049435
Configuring test contracts...
Configuring test roles...
 RNGProxySelfHosted: requestRandom
   requestRandom() - Ensure proper event emission (5.269375ms)mm
 mRNGProxySelfHosted: requestRandom (1916.938083ms)m

 RNGProxySelfHosted: provideRandom
   ensure proper callback
     provideRandom() - Ensure proper event emission (12.313125ms)mm
   mensure proper callback (13.2415ms)m

 mRNGProxySelfHosted: provideRandom (1887.93725ms)m

 RNGProxySelfHosted: inaccessible external functions check
   try calling functions that are only accessible via contracts or never accessible
     requestRandom() (17.347083ms)mm
     provideRandom() (12.595292ms)mm
   mtry calling functions that are only accessible via contracts or never accessible (31.797083ms)m

 mRNGProxySelfHosted: inaccessible external functions check (53.976875ms)m

 NFTOverlord: startReveal
   should revert with PlayerNotRegisteredError when player not registered (3.134875ms)mm
   when player is registered
     should revert with NoUnrevealedMunchablesError when player has no unrevealed NFTs (5.013708ms)mm
     when player has unrevealed NFTs
       should succeed (10.628459ms)mm
       when reveal queue is full
         should revert with RevealQueueFullError when player reveal queue is full (9.649083ms)mm
       mwhen reveal queue is full (10.118166ms)m

     mwhen player has unrevealed NFTs (21.259834ms)m

   mwhen player is registered (26.998625ms)m

 mNFTOverlord: startReveal (1837.395125ms)m

 NFTOverlord: revealFromPrimordial
   should revert with UnauthorisedError when not called by RNGProxy (3.654584ms)mm
   should revert with RevealQueueEmptyError when nothing in queue (2.227458ms)mm
   when there is a reveal in queue
     should revert with NotEnoughRandomError when called with not enough RNG (5.952416ms)mm
     should reveal NFT with rarity 1 (45.278583ms)mm
```

```
    should reveal NFT with rarity 1 (21.109125ms)mm
    should reveal NFT with rarity 2 (17.384458ms)mm
    should reveal NFT with rarity 2 (13.561ms)mm
  mwhen there is a reveal in queue (107.023916ms)m

mNFTOverlord: revealFromPrimordial (2083.30975ms)m

NFTOverlord: reveal
  should revert with UnauthorisedError when not called by RNGProxy (3.158667ms)mm
  should revert with RevealQueueEmptyError when nothing in queue (1.902542ms)mm
  when there is a reveal in queue
    should revert with NotEnoughRandomError when called with not enough RNG (11.552167ms)mm
    should reveal NFT with rarity 1 (13.98525ms)mm
    should reveal NFT with rarity 1 (11.71425ms)mm
    should reveal NFT with rarity 2 (11.755666ms)mm
    should reveal NFT with rarity 3 (13.740167ms)mm
    should reveal NFT with rarity 4 (10.7125ms)mm
    should reveal NFT with rarity 5 (10.674417ms)mm
  mwhen there is a reveal in queue (87.455666ms)m

  allow reveal queue > 1 (4088.759334ms)mm
mNFTOverlord: reveal (6143.632333ms)m

NFTOverlord: munchableFed
  should revert with UnauthorisedError when not called by SnuggeryManager (5.780166ms)mm
  when level up not applicable
    should succeed and do nothing (6.7815ms)mm
  mwhen level up not applicable (7.320333ms)m

  when level up to 2 applicable
    should succeed (7.813375ms)mm
  mwhen level up to 2 applicable (11.800625ms)m

  when level up to 3 applicable
    should succeed (7.122833ms)mm
  mwhen level up to 3 applicable (7.603708ms)m

  when level up to 4 applicable
    should succeed (6.225375ms)mm
  mwhen level up to 4 applicable (6.7785ms)m

  when level up to 5 applicable
    should succeed (6.469375ms)mm
  mwhen level up to 5 applicable (6.915083ms)m

  when level up to 6 applicable
    should succeed (6.128ms)mm
  mwhen level up to 6 applicable (6.586916ms)m

  when level up to 7 applicable
    should succeed (6.133792ms)mm
  mwhen level up to 7 applicable (6.537083ms)m

  when level up to 8 applicable
    should succeed (6.122166ms)mm
  mwhen level up to 8 applicable (6.531833ms)m

  when level up to 9 applicable
    should succeed (9.045667ms)mm
  mwhen level up to 9 applicable (9.441916ms)m

  when level up to 10 applicable
    should succeed (6.16675ms)mm
  mwhen level up to 10 applicable (6.669667ms)m

  when level up to 11 applicable
    should succeed (5.810458ms)mm
  mwhen level up to 11 applicable (6.1995ms)m

  when level up to 12 applicable
```

```
  should succeed (5.669084ms)mm
mwhen level up to 12 applicable (6.053ms)m

when level up to 13 applicable
  should succeed (5.932875ms)mm
mwhen level up to 13 applicable (6.367334ms)m

when level up to 14 applicable
  should succeed (6.150958ms)mm
mwhen level up to 14 applicable (6.540416ms)m

when level up to 15 applicable
  should succeed (10.920167ms)mm
mwhen level up to 15 applicable (11.412041ms)m

when level up to 16 applicable
  should succeed (5.975709ms)mm
mwhen level up to 16 applicable (6.347417ms)m

when level up to 17 applicable
  should succeed (5.6505ms)mm
mwhen level up to 17 applicable (6.098917ms)m

when level up to 18 applicable
  should succeed (5.636833ms)mm
mwhen level up to 18 applicable (6.00975ms)m

when level up to 19 applicable
  should succeed (5.508791ms)mm
mwhen level up to 19 applicable (5.886958ms)m

when level up to 20 applicable
  should succeed (5.460417ms)mm
mwhen level up to 20 applicable (5.874625ms)m

when level up to 21 applicable
  should succeed (5.548292ms)mm
mwhen level up to 21 applicable (5.954625ms)m

when level up to 22 applicable
  should succeed (5.727292ms)mm
mwhen level up to 22 applicable (6.120584ms)m

when level up to 23 applicable
  should succeed (8.860666ms)mm
mwhen level up to 23 applicable (9.275959ms)m

when level up to 24 applicable
  should succeed (5.632959ms)mm
mwhen level up to 24 applicable (5.981625ms)m

when level up to 25 applicable
  should succeed (5.330334ms)mm
mwhen level up to 25 applicable (5.679875ms)m

when level up to 26 applicable
  should succeed (5.368291ms)mm
mwhen level up to 26 applicable (5.730833ms)m

when level up to 27 applicable
  should succeed (5.598375ms)mm
mwhen level up to 27 applicable (5.986667ms)m

when level up to 28 applicable
  should succeed (8.457291ms)mm
mwhen level up to 28 applicable (8.945791ms)m

when level up to 29 applicable
  should succeed (5.705084ms)mm
mwhen level up to 29 applicable (6.113417ms)m
```

```
when level up to 30 applicable
  should succeed (5.883458ms)mm
mwhen level up to 30 applicable (6.328875ms)m

when level up to 31 applicable
  should succeed (5.687625ms)mm
mwhen level up to 31 applicable (6.055084ms)m

when level up to 32 applicable
  should succeed (9.728875ms)mm
mwhen level up to 32 applicable (10.151333ms)m

when level up to 33 applicable
  should succeed (5.454458ms)mm
mwhen level up to 33 applicable (5.850334ms)m

when level up to 34 applicable
  should succeed (6.040625ms)mm
mwhen level up to 34 applicable (6.434ms)m

when level up to 35 applicable
  should succeed (5.372959ms)mm
mwhen level up to 35 applicable (5.718375ms)m

when level up to 36 applicable
  should succeed (5.206458ms)mm
mwhen level up to 36 applicable (5.565375ms)m

when level up to 37 applicable
  should succeed (5.479292ms)mm
mwhen level up to 37 applicable (5.825833ms)m

when level up to 38 applicable
  should succeed (5.519916ms)mm
mwhen level up to 38 applicable (5.903583ms)m

when level up to 39 applicable
  should succeed (8.6995ms)mm
mwhen level up to 39 applicable (9.129333ms)m

when level up to 40 applicable
  should succeed (5.526958ms)mm
mwhen level up to 40 applicable (5.897167ms)m

when level up to 41 applicable
  should succeed (5.135833ms)mm
mwhen level up to 41 applicable (5.5025ms)m

when level up to 42 applicable
  should succeed (5.164083ms)mm
mwhen level up to 42 applicable (5.511375ms)m

when level up to 43 applicable
  should succeed (5.123542ms)mm
mwhen level up to 43 applicable (5.448917ms)m

when level up to 44 applicable
  should succeed (5.603625ms)mm
mwhen level up to 44 applicable (5.995166ms)m

when level up to 45 applicable
  should succeed (8.356083ms)mm
mwhen level up to 45 applicable (8.737917ms)m

when level up to 46 applicable
  should succeed (5.141834ms)mm
mwhen level up to 46 applicable (5.65625ms)m

when level up to 47 applicable
```

```
  should succeed (8.983167ms)mm
mwhen level up to 47 applicable (9.349667ms)m

when level up to 48 applicable
  should succeed (5.839958ms)mm
mwhen level up to 48 applicable (6.237167ms)m

when level up to 49 applicable
  should succeed (5.691166ms)mm
mwhen level up to 49 applicable (6.090458ms)m

when level up to 50 applicable
  should succeed (5.073167ms)mm
mwhen level up to 50 applicable (5.407583ms)m

when level up to 51 applicable
  should succeed (5.132417ms)mm
mwhen level up to 51 applicable (5.4655ms)m

when level up to 52 applicable
  should succeed (5.308458ms)mm
mwhen level up to 52 applicable (5.657041ms)m

when level up to 53 applicable
  should succeed (5.0195ms)mm
mwhen level up to 53 applicable (5.387709ms)m

when level up to 54 applicable
  should succeed (8.832542ms)mm
mwhen level up to 54 applicable (9.215584ms)m

when level up to 55 applicable
  should succeed (5.040583ms)mm
mwhen level up to 55 applicable (5.369792ms)m

when level up to 56 applicable
  should succeed (5.23325ms)mm
mwhen level up to 56 applicable (5.578167ms)m

when level up to 57 applicable
  should succeed (5.32325ms)mm
mwhen level up to 57 applicable (5.676083ms)m

when level up to 58 applicable
  should succeed (5.109709ms)mm
mwhen level up to 58 applicable (5.489125ms)m

when level up to 59 applicable
  should succeed (5.461ms)mm
mwhen level up to 59 applicable (5.810958ms)m

when level up to 60 applicable
  should succeed (5.061458ms)mm
mwhen level up to 60 applicable (8.693416ms)m

when level up to 61 applicable
  should succeed (5.554375ms)mm
mwhen level up to 61 applicable (5.898958ms)m

when level up to 62 applicable
  should succeed (8.619375ms)mm
mwhen level up to 62 applicable (9.003417ms)m

when level up to 63 applicable
  should succeed (5.052167ms)mm
mwhen level up to 63 applicable (5.420792ms)m

when level up to 64 applicable
  should succeed (5.054375ms)mm
mwhen level up to 64 applicable (5.411583ms)m
```

```
when level up to 65 applicable
  should succeed (5.053709ms)mm
mwhen level up to 65 applicable (5.417208ms)m

when level up to 66 applicable
  should succeed (5.094875ms)mm
mwhen level up to 66 applicable (5.436917ms)m

when level up to 67 applicable
  should succeed (5.144333ms)mm
mwhen level up to 67 applicable (5.743917ms)m

when level up to 68 applicable
  should succeed (5.636708ms)mm
mwhen level up to 68 applicable (5.993084ms)m

when level up to 69 applicable
  should succeed (5.446291ms)mm
mwhen level up to 69 applicable (8.990334ms)m

when level up to 70 applicable
  should succeed (5.113209ms)mm
mwhen level up to 70 applicable (5.465917ms)m

when level up to 71 applicable
  should succeed (5.143542ms)mm
mwhen level up to 71 applicable (5.486084ms)m

when level up to 72 applicable
  should succeed (4.892459ms)mm
mwhen level up to 72 applicable (5.208209ms)m

when level up to 73 applicable
  should succeed (5.1405ms)mm
mwhen level up to 73 applicable (5.496541ms)m

when level up to 74 applicable
  should succeed (5.262458ms)mm
mwhen level up to 74 applicable (5.601792ms)m

when level up to 75 applicable
  should succeed (4.9875ms)mm
mwhen level up to 75 applicable (5.351833ms)m

when level up to 76 applicable
  should succeed (8.245292ms)mm
mwhen level up to 76 applicable (8.591583ms)m

when level up to 77 applicable
  should succeed (5.227583ms)mm
mwhen level up to 77 applicable (5.559667ms)m

when level up to 78 applicable
  should succeed (5.033208ms)mm
mwhen level up to 78 applicable (5.33525ms)m

when level up to 79 applicable
  should succeed (4.962084ms)mm
mwhen level up to 79 applicable (5.28225ms)m

when level up to 80 applicable
  should succeed (5.130083ms)mm
mwhen level up to 80 applicable (5.42925ms)m

when level up to 81 applicable
  should succeed (5.086708ms)mm
mwhen level up to 81 applicable (5.394167ms)m

when level up to 82 applicable
```

```
  should succeed (10.367459ms)mm
mwhen level up to 82 applicable (10.745417ms)m

when level up to 83 applicable
  should succeed (5.149042ms)mm
mwhen level up to 83 applicable (5.475958ms)m

when level up to 84 applicable
  should succeed (5.035458ms)mm
mwhen level up to 84 applicable (5.351542ms)m

when level up to 85 applicable
  should succeed (5.068625ms)mm
mwhen level up to 85 applicable (5.397208ms)m

when level up to 86 applicable
  should succeed (4.880375ms)mm
mwhen level up to 86 applicable (5.186375ms)m

when level up to 87 applicable
  should succeed (4.930167ms)mm
mwhen level up to 87 applicable (5.249542ms)m

when level up to 88 applicable
  should succeed (7.615208ms)mm
mwhen level up to 88 applicable (7.951709ms)m

when level up to 89 applicable
  should succeed (4.965375ms)mm
mwhen level up to 89 applicable (5.277416ms)m

when level up to 90 applicable
  should succeed (4.918167ms)mm
mwhen level up to 90 applicable (5.25525ms)m

when level up to 91 applicable
  should succeed (5.088333ms)mm
mwhen level up to 91 applicable (5.401375ms)m

when level up to 92 applicable
  should succeed (4.88175ms)mm
mwhen level up to 92 applicable (5.187833ms)m

when level up to 93 applicable
  should succeed (4.952625ms)mm
mwhen level up to 93 applicable (5.254792ms)m

when level up to 94 applicable
  should succeed (8.018792ms)mm
mwhen level up to 94 applicable (8.346417ms)m

when level up to 95 applicable
  should succeed (4.9285ms)mm
mwhen level up to 95 applicable (5.233125ms)m

when level up to 96 applicable
  should succeed (4.854084ms)mm
mwhen level up to 96 applicable (5.17825ms)m

when level up to 97 applicable
  should succeed (5.028209ms)mm
mwhen level up to 97 applicable (5.326459ms)m

when level up to 98 applicable
  should succeed (4.784125ms)mm
mwhen level up to 98 applicable (5.072792ms)m

when level up to 99 applicable
  should succeed (4.814583ms)mm
mwhen level up to 99 applicable (5.165667ms)m
```

```
  when level up to 100 applicable
    should succeed (7.7655ms)mm
  mwhen level up to 100 applicable (8.111667ms)m

  when level up to 101 applicable
    should succeed (4.828458ms)mm
  mwhen level up to 101 applicable (5.182625ms)m

mNFTOverlord: munchableFed (4254.182ms)m

NFTOverlord: mintFromPrimordial
  should revert with UnauthorisedError when not called by PrimordialManager (2.997625ms)mm
  should succeed when called by PrimordialManager (6.750042ms)mm
mNFTOverlord: mintFromPrimordial (1818.913ms)m

NFTOverlord: mintForMigration
  should revert with UnauthorisedError when not called by MigrationManager (4.389416ms)mm
  should succeed when called by MigrationManager (9.560125ms)mm
  should succeed when called twice by MigrationManager (13.465625ms)mm
mNFTOverlord: mintForMigration (1838.744792ms)m

NFTOverlord: levelUp
  should revert with UnauthorisedError when not called by RNGProxy (3.032209ms)mm
  when no level up request exists
    should revert with InvalidLevelUpRequest (1.778125ms)mm
  mwhen no level up request exists (2.393166ms)m

  when level up request exists
    when level has changed
      should revert with InvalidLevelUpRequest (14.084459ms)mm
    mwhen level has changed (14.906125ms)m

    should revert with NotEnoughRandomError when not enough RNG (6.915958ms)mm
    when game attributes not set
      should increase level and set game attributes (17.262375ms)mm
    mwhen game attributes not set (17.80725ms)m

    when level and game attributes already increased
      should increase level and increase game attributes (27.823833ms)mm
    mwhen level and game attributes already increased (28.385458ms)m

  mwhen level up request exists (68.603958ms)m

mNFTOverlord: levelUp (3832.076ms)m

NFTOverlord: addReveal
  should revert with UnauthorisedError when not called by LockManager (3.378334ms)mm
  should add quantity to unrevealedNFTs for player when called by LockManager (5.262416ms)mm
mNFTOverlord: addReveal (1828.995958ms)m

SnuggeryManager: snuggery size
  Increase snuggery size
    must fail if price is not configured (4095.545875ms)mm
    must fail if player does not have enough points (114.354958ms)mm
    successful increase in size (120.63075ms)mm
  mIncrease snuggery size (4340.164208ms)m

mSnuggeryManager: snuggery size (7962.448416ms)m

SnuggeryManager: import export Munchables
  importing munchables
    importMunchable fails without approval (76.876416ms)mm
    importMunchable succeeds with approval for all (85.93875ms)mm
    importMunchable succeeds with individual approval (88.221084ms)mm
    importMunchable fails if not owner (65.339167ms)mm
    importMunchable fails if max snuggery size is reached (127.696042ms)mm
  mimporting munchables (448.249916ms)m

  exporting munchables
```

```
    exportMunchable fails if player does not own token (88.851125ms)mm
    exportMunchable fails if token is not in snuggery (80.65825ms)mm
    exportMunchable succeeds with event (68.405208ms)mm
  mexporting munchables (244.069208ms)m

mSnuggeryManager: import export Munchables (2600.403792ms)m

SnuggeryManager: feeding
  Feed own munchable
    feed without any schnibbles (146.854584ms)mm
    feed with schnibbles (139.0875ms)mm
  mFeed own munchable (289.665ms)m

mSnuggeryManager: feeding (4241.15125ms)m

PrimordialManager: hatchPrimordialToMunchable
  all paths
    revert hatching when paused (120.660542ms)mm
    revert if primordial doesn't exist (63.119458ms)mm
    revert if primordial not ready (64.46375ms)mm
    mint if ready (73.79175ms)mm
    cant claim again if already hatched (70.092917ms)mm
    mint if ready - subaccount (73.294833ms)mm
  mall paths (469.15625ms)m

mPrimordialManager: hatchPrimordialToMunchable (2306.802167ms)m

PrimordialManager: feedPrimordial
  all paths
    revert feeding when paused (112.301333ms)mm
    revert feeding when primordials not enabled (107.094167ms)mm
    revert - tries to feed too much (58.416584ms)mm
    revert - primordial doesn't exist (62.551542ms)mm
    normal primordial feeding (62.590542ms)mm
    normal subaccount feeding (64.302292ms)mm
    normal primordial feeding - levelup (59.980917ms)mm
    capped primordial feeding (4093.527625ms)mm
  mall paths (4626.89325ms)m

mPrimordialManager: feedPrimordial (6457.319416ms)m

PrimordialManager: claimPrimordial
  all paths
    revert claiming when paused (63.590125ms)mm
    revert claiming when primordials not enabled (50.027ms)mm
    try claiming double (51.213125ms)mm
    claim normal (53.455584ms)mm
    claim by subaccount (57.052292ms)mm
  mall paths (279.32425ms)m

mPrimordialManager: claimPrimordial (308.528ms)m

NFTAttributeManagerV1: read/write
  Setting attributes
    createWithImmutable() (24.613459ms)mm
    setAttributes() - NFTOverlord (18.24575ms)mm
    setAttributes() - SnuggeryManager (23.492667ms)mm
    setGameAttributes() (16.790875ms)mm
    getImmutableAttributes() (18.202583ms)mm
    getAttributes() (23.807916ms)mm
    getGameAttributes() (18.994292ms)mm
  mSetting attributes (149.272833ms)m

mNFTAttributeManagerV1: read/write (2025.610375ms)m

NFTAttributeManagerV1: auth
  Authorisation
    setAttributes() - invalid auth (27.017542ms)mm
    setGameAttributes() - invalid auth (15.945459ms)mm
    createWithImmutable() - invalid auth (16.737042ms)mm
```

```
  mAuthorisation (62.205167ms)m

mNFTAttributeManagerV1: auth (1934.2335ms)m

MunchadexManager: updateMunchadex tests
  should correctly handle minting and Munchadex update (20.733125ms)mm
  should correctly handle NFT transfers between users (17.575375ms)mm
  proper ignoring
    import/export snuggery (32.317833ms)mm
  mproper ignoring (33.275ms)m

  proper counting
    should correctly add unique munchables to same realm (29.25225ms)mm
    should correctly add new munchables to different realms and transfers reduce unique (28.207584ms)mm
    should correctly add non-unique munchables and not decrement numUnique until all copies are transferred
    ↪  (42.611084ms)mm
  mproper counting (102.097292ms)m

mMunchadexManager: updateMunchadex tests (2059.525291ms)m

MunchadexManager: inaccessible external functions check
  try calling functions that are only accessible via contracts or never accessible
    updateMunchadex() (14.711917ms)mm
  mtry calling functions that are only accessible via contracts or never accessible (15.892583ms)m

mMunchadexManager: inaccessible external functions check (41.749542ms)m

MigrationManager: sealData
  All paths
    Revert - Not Admin (5.560291ms)mm
    Revert - Already sealed (25.965208ms)mm
    Succeed - Seal (5.430375ms)mm
  mAll paths (40.990917ms)m

mMigrationManager: sealData (69.471583ms)m

MigrationManager: migratePurchasedNFTs
  All paths
    Revert - Not sealed (30.648625ms)mm
    Revert - Not bought (29.787458ms)mm
    Revert - Not 0 lock amount (21.735667ms)mm
    Revert - invalid token id (25.981458ms)mm
    Revert - Didn't send enough ETH (20.812833ms)mm
    Revert - Tries to call migratePurchasedNFTs after burning (28.467541ms)mm
    Success - Migrate nothing (token id 0 skipped) (24.339375ms)mm
    Success - One (and nothing when redoing the call) (37.953625ms)mm
    Success - Multiple (28.044167ms)mm
  mAll paths (253.371084ms)m

mMigrationManager: migratePurchasedNFTs (477.16425ms)m

MigrationManager: migrateAllNFTs
  All paths
    Revert - Not locked (32.365458ms)mm
    Revert - Invalid Skip (33.33425ms)mm
    Revert - Tried to burn then migrate (40.096375ms)mm
    Success - Migrate Purchased then rest of all then migrate again for nothing (61.893333ms)mm
    Success - ERC-20s (44.160292ms)mm
  mAll paths (219.5745ms)m

mMigrationManager: migrateAllNFTs (858.726833ms)m

MigrationManager: lockFundsForAllMigration
  All paths
    Revert - Not sealed (28.113167ms)mm
    Revert - Locked (37.503083ms)mm
    Revert - Nothing to migrate (25.023416ms)mm
    Revert - Not enough ETH sent (27.243875ms)mm
    Revert - ETH sent for ERC20 lock (24.399375ms)mm
    Success - ETH (29.416292ms)mm
    Success - ERC20 (33.652542ms)mm
```

```
  mAll paths (210.093625ms)m

mMigrationManager: lockFundsForAllMigration (652.3825ms)m

MigrationManager: loadUnrevealedSnapshot
  All paths
    Revert - Not Admin (5.377125ms)mm
    Revert - Already sealed (26.091125ms)mm
    Revert - args not equal length (1.978708ms)mm
    Succeed - Load multiple times (10.214708ms)mm
  mAll paths (47.767ms)m

mMigrationManager: loadUnrevealedSnapshot (72.821833ms)m

MigrationManager: loadMigrationSnapshot
  All paths
    Revert - Not Admin (5.547834ms)mm
    Revert - Already sealed (26.410875ms)mm
    Revert - Same token for same user more than once (8.0835ms)mm
    Revert - non-ETH/USDB/WETH token (1.830417ms)mm
    Revert - No lock amount for non-ETH token (2.583583ms)mm
    Succeed - Load multiple times (10.533583ms)mm
  mAll paths (60.266833ms)m

mMigrationManager: loadMigrationSnapshot (86.527416ms)m

MigrationManager: burnUnrevealedForPoints
  All paths
    Revert - Not sealed (17.037667ms)mm
    Revert - Nothing available (17.482ms)mm
    Revert - Tried to burn twice (11.718625ms)mm
    Success (10.985042ms)mm
  mAll paths (60.589375ms)m

mMigrationManager: burnUnrevealedForPoints (85.837167ms)m

MigrationManager: burnRemainingPurchasedNFTs
  All paths
    Revert - Not sealed (17.443291ms)mm
    Revert - Nothing available (14.83875ms)mm
    Success - burns 4 of first 5 on first call, nothing on second call (36.5605ms)mm
    Success with skip - burns last token (19.01875ms)mm
    Burns on behalf after initial burn (22.719417ms)mm
  mAll paths (114.581917ms)m

mMigrationManager: burnRemainingPurchasedNFTs (337.576375ms)m

MigrationManager: burnNFTs
  All paths
    Revert - Loading token id 0 (12.932625ms)mm
    Revert - Not sealed (12.448041ms)mm
    Revert - Nothing available (14.32325ms)mm
    Revert - user hasn't locked action (16.322708ms)mm
    Revert - user has locked full migration (16.293625ms)mm
    Success - burns 4 of first 5 on first call, nothing on second call (22.76275ms)mm
    Success with skip - burns last token (21.299ms)mm
  mAll paths (120.430875ms)m

mMigrationManager: burnNFTs (344.740792ms)m

LockManager: unlock
  when player has no locked tokens
    should revert with InsufficientLockAmountError (5.885041ms)mm
  mwhen player has no locked tokens (7.569458ms)m

  when player has locked zero-address
    should revert with TokenStillLockedError when unlock time is in future (11.337834ms)mm
    when unlock time is past
      should revert with InsufficientLockAmountError when attempting to unlock a different token (11.279959ms)mm
      should revert with InsufficientLockAmountError when quantity > locked quantity (11.276792ms)mm
```

```
      should succeed when quantity == locked quantity (17.704ms)mm
      should succeed and leave some locked when quantity < locked quantity (16.972292ms)mm
    mwhen unlock time is past (63.213333ms)m

  mwhen player has locked zero-address (75.421375ms)m

  when player has locked ERC20 tokens
    when unlock time is past
      should revert with InsufficientLockAmountError when attempting to unlock a different token (11.799083ms)mm
      should succeed when quantity == locked quantity (17.728042ms)mm
    mwhen unlock time is past (30.68975ms)m

  mwhen player has locked ERC20 tokens (30.7475ms)m

mLockManager: unlock (314.270417ms)m

LockManager: lock
  when zero-address token is configured on the contract
    should revert with AccountNotRegisteredError when player not registered for account (17.179125ms)mm
    when sub-account is available
      should revert with SubAccountCannotLockError when sub-account is used (12.152958ms)mm
    mwhen sub-account is available (13.392625ms)m

    should revert with ETHValueIncorrectError when quantity != value (6.19925ms)mm
    should succeed when quantity == value (17.320709ms)mm
    should succeed twice and increase quantity of lock (25.783083ms)mm
    when lockdrop is active
      when lock duration is too short
        should revert with InvalidLockDurationError (10.479834ms)mm
      mwhen lock duration is too short (11.00775ms)m

      when lock duration is too long
        should revert with InvalidLockDurationError (10.318125ms)mm
      mwhen lock duration is too long (10.972833ms)m

      should succeed with remainder and numberNFTs (19.913792ms)mm
      when lock duration is 0
        should succeed with min lock duration from lockdrop (17.353292ms)mm
      mwhen lock duration is 0 (18.023583ms)m

    mwhen lockdrop is active (60.624583ms)m

    when using lockOnBehalf
      should lock token for other player (13.66275ms)mm
    mwhen using lockOnBehalf (14.199459ms)m

  mwhen zero-address token is configured on the contract (158.382834ms)m

  when an ERC20 token is used
    when the token is not configured on the contract
      should revert with TokenNotConfiguredError (2.339583ms)mm
    mwhen the token is not configured on the contract (2.766958ms)m

    when the token is configured but is inactive on the contract
      should revert with TokenNotConfiguredError (4.62725ms)mm
    mwhen the token is configured but is inactive on the contract (5.139458ms)m

    when the token is configured on the contract
      should revert with InvalidMessageValueError when value is not 0 (7.312ms)mm
      should revert with InsufficientAllowanceError when quanity > allowance (12.163666ms)mm
      should succeed when quantity <= allowance (14.315416ms)mm
      when lockdrop is active
        should succeed with remainder and numberNFTs (16.937583ms)mm
      mwhen lockdrop is active (17.425792ms)m

    mwhen the token is configured on the contract (52.720084ms)m

  mwhen an ERC20 token is used (60.755292ms)m

  when ETH and multiple ERC20 tokens are used
```

```
    should allow all 3 tokens to be locked at once (41.010542ms)mm
    when lockdrop is active
      should allow all 3 tokens to be locked at once with remainder and numberNFTs set (44.620666ms)mm
    mwhen lockdrop is active (45.142708ms)m

  mwhen ETH and multiple ERC20 tokens are used (86.60775ms)m

mLockManager: lock (656.242416ms)m

LockManager: subacount prevention
  subaccount proper locking/unlocking behavior
    allow lock for main account on behalf but not unlock (29.382666ms)mm
  msubaccount proper locking/unlocking behavior (30.202875ms)m

mLockManager: subacount prevention (32.024667ms)m

LockManager: setUSDThresholds
  should revert with InvalidRoleError when called as non-admin (4.476042ms)mm
  should succeed when called as admin (14.314042ms)mm
mLockManager: setUSDThresholds (47.710791ms)m

LockManager: setLockDuration
  should revert with MaximumLockDurationError when setting lock duration greater than configured max (10.217708ms)mm
  when player has no locked tokens
    should succeed when setting to max duration (12.367625ms)mm
  mwhen player has no locked tokens (13.489542ms)m

  when player has locked tokens for less than max duration
    should revert with LockDurationReducedError when reducing lock duration (28.420792ms)mm
    should succeed when locking for max duration and increase unlock time (41.919125ms)mm
  mwhen player has locked tokens for less than max duration (71.729042ms)m

mLockManager: setLockDuration (282.029208ms)m

LockManager: proposeUSDPrice
  should revert with InvalidRoleError when called as non price-feed (5.779792ms)mm
  should revert with ProposalInvalidContractsError when called with no contracts (25.652917ms)mm
  should succeed when called as price-feed (10.455583ms)mm
  when a proposal is in progress
    should revert with ProposalInProgressError (1.69025ms)mm
  mwhen a proposal is in progress (5.691ms)m

mLockManager: proposeUSDPrice (79.063917ms)m

LockManager: lock
  when zero-address token is configured on the contract
    should revert with AccountNotRegisteredError when player not registered for account (17.805833ms)mm
    when sub-account is available
      should revert with SubAccountCannotLockError when sub-account is used (18.363459ms)mm
    mwhen sub-account is available (19.735292ms)m

    should revert with ETHValueIncorrectError when quantity != value (15.156583ms)mm
    should succeed when quantity == value (25.513542ms)mm
    should succeed twice and increase quantity of lock (33.061416ms)mm
    when lockdrop is active
      when lock duration is too short
        should revert with InvalidLockDurationError (11.888333ms)mm
      mwhen lock duration is too short (12.58125ms)m

      when lock duration is too long
        should revert with InvalidLockDurationError (11.940209ms)mm
      mwhen lock duration is too long (12.564083ms)m

      should succeed with remainder and numberNFTs (24.547625ms)mm
      when lock duration is 0
        should succeed with min lock duration from lockdrop (20.014042ms)mm
      mwhen lock duration is 0 (20.543959ms)m

    mwhen lockdrop is active (70.888708ms)m
```

```
    when using lockOnBehalf
      should lock token for other player (12.705459ms)mm
    mwhen using lockOnBehalf (13.280292ms)m

  mwhen zero-address token is configured on the contract (199.54925ms)m

  when an ERC20 token is used
    when the token is not configured on the contract
      should revert with TokenNotConfiguredError (2.18325ms)mm
    mwhen the token is not configured on the contract (2.6305ms)m

    when the token is configured but is inactive on the contract
      should revert with TokenNotConfiguredError (4.87775ms)mm
    mwhen the token is configured but is inactive on the contract (5.432959ms)m

    when the token is configured on the contract
      should revert with InvalidMessageValueError when value is not 0 (13.431917ms)mm
      should revert with InsufficientAllowanceError when quanity > allowance (7.438875ms)mm
      should succeed when quantity <= allowance (13.7275ms)mm
      when lockdrop is active
        should succeed with remainder and numberNFTs (20.815ms)mm
      mwhen lockdrop is active (21.373083ms)m

    mwhen the token is configured on the contract (57.642ms)m

  mwhen an ERC20 token is used (65.81675ms)m

  when ETH and multiple ERC20 tokens are used
    should allow all 3 tokens to be locked at once (42.748167ms)mm
    when lockdrop is active
      should allow all 3 tokens to be locked at once with remainder and numberNFTs set (44.69125ms)mm
    mwhen lockdrop is active (45.213958ms)m

  mwhen ETH and multiple ERC20 tokens are used (88.477125ms)m

mLockManager: lock (706.006083ms)m

LockManager: disapproveUSDPrice
  should revert with NoProposalError when no proposal in progress (5.465833ms)mm
  when a proposal is in progress
    should revert with ProposalAlreadyDisapprovedError when already disapproved by sender (11.485541ms)mm
    should revert when not properly defined role (1.6115ms)mm
    should revert with ProposalAlreadyApprovedError if sender already approved (4.857375ms)mm
    should revert with ProposalPriceNotMatchedError when called with incorrect price (1.936959ms)mm
    should succeed and remove proposal when disapproval threshold is reached (9.415541ms)mm
  mwhen a proposal is in progress (46.890375ms)m

mLockManager: disapproveUSDPrice (78.820292ms)m

LockManager: configureToken
  should revert with InvalidRoleError when called as non-admin (10.248583ms)mm
  when configuring already configured token
    should succeed and update all token config properties (18.132375ms)mm
  mwhen configuring already configured token (18.874292ms)m

  when configuring new token
    should succeed and set all token config properties (7.144333ms)mm
  mwhen configuring new token (7.92875ms)m

mLockManager: configureToken (214.638834ms)m

LockManager: configureLockdrop
  should revert with InvalidRoleError when called as non-admin (8.847334ms)mm
  should revert with LockdropEndedError when setting lockdrop end in past (24.505875ms)mm
  should revert with LockdropInvalidError when setting lockdrop start > end (4.151167ms)mm
  should revert with LockdropInvalidError when setting lockdrop start == end (2.647541ms)mm
  should succeed when lockdrop starts in past and ends in future (8.254834ms)mm
  should succeed when overwriting lockdrop with start in future (9.807833ms)mm
mLockManager: configureLockdrop (92.731417ms)m
```

```
LockManager: approveUSDPrice
  should revert with NoProposalError when no proposal in progress (7.285667ms)mm
  when a proposal is in progress
    should revert with InvalidRoleError when called as non price-feed (16.895333ms)mm
    should revert with ProposerCannotApproveError when called as proposer (6.198167ms)mm
    should revert with ProposalPriceNotMatchedError when called with incorrect price (6.41175ms)mm
    should revert with ProposalAlreadyApprovedError (9.587334ms)mm
  mwhen a proposal is in progress (41.723375ms)m

  2 approvals needed
    should succeed and update price with 2 approvals (11.88225ms)mm
  m2 approvals needed (12.393125ms)m

  3 approvals needed
    should succeed and NOT update price with 2/3 (16.800125ms)mm
    should succeed and update price with 3/3 (13.987292ms)mm
  m3 approvals needed (32.140834ms)m

mLockManager: approveUSDPrice (130.754375ms)m

ClaimManager: spendPoints
  all paths
    revert if tries to spend too much (46.915375ms)mm
    spend points properly (48.981583ms)mm
  mall paths (97.756459ms)m

mClaimManager: spendPoints (3751.472459ms)m

ClaimManager: newPeriod
  paused (46.735916ms)mm
  period not ended (14.199834ms)mm
  successful successive calls (8.083041ms)mm
mClaimManager: newPeriod (98.385875ms)m

ClaimManager: inaccessible external functions check
  try calling functions that are only accessible via contracts or never accessible
    initialize() - proxy (17.595625ms)mm
    initialize() - root (16.322375ms)mm
    configUpdated() (5.475166ms)mm
    newPeriod() (4.915291ms)mm
    burnNFTsForPoints() (5.973125ms)mm
    burnUnrevealedForPoints() (5.104958ms)mm
    forceClaimPoints() (5.413ms)mm
    spendPoints() (4.895625ms)mm
  mtry calling functions that are only accessible via contracts or never accessible (71.509416ms)m

mClaimManager: inaccessible external functions check (97.54725ms)m

ClaimManager: forceClaimPoints
  normal behavior
    claim, with referral, no excess (48.476083ms)mm
    claim, no referrer, no excess (42.758792ms)mm
    claim, successive calls, zero points (39.232041ms)mm
    claim, successive calls on successive periods, excess skips over (47.717167ms)mm
    revert claim outside of valid period (40.517916ms)mm
  mnormal behavior (223.895917ms)m

mClaimManager: forceClaimPoints (3843.853084ms)m

ClaimManager: convertPointsToTokens()
  all pathways
    revert if paused (84.910625ms)mm
    revert if not swappable (39.81125ms)mm
    revert if no points available (74.4915ms)mm
    revert if send in zero points (80.294292ms)mm
    revert if points per token is 0 (4160.959709ms)mm
    revert if tries to convert too many points (122.676583ms)mm
    revert if token mint is zero (120.644333ms)mm
    revert if munch token is not set (4200.885875ms)mm
    successful points to token converstion (126.41075ms)mm
```

```
  mall pathways (9017.807416ms)m

mClaimManager: convertPointsToTokens() (12838.61975ms)m

ClaimManager: claimPoints
  all paths
    claim, with referral, no excess (46.327ms)mm
    claim, no referrer, no excess (37.493625ms)mm
    claim, successive calls, zero points (43.810541ms)mm
    claim, successive calls on successive periods, excess skips over (46.379958ms)mm
    claim, subaccount (44.755625ms)mm
    revert claim outside of valid period (31.287583ms)mm
  mall paths (254.120042ms)m

mClaimManager: claimPoints (3885.012334ms)m

ClaimManager: burnUnrevealedForPoints
  burn unrevealed for points
    test 1 unrevealed (9.996917ms)mm
    test 3 unrevealed (6.517333ms)mm
  mburn unrevealed for points (18.48ms)m

mClaimManager: burnUnrevealedForPoints (1832.082042ms)m

ClaimManager: burnNFTsForPoints
  burn nfts for points
    test single rarity (8.806167ms)mm
    test various rarities (5.860709ms)mm
    nothing for outside of range (5.414125ms)mm
  mburn nfts for points (22.648583ms)m

mClaimManager: burnNFTsForPoints (1852.197ms)m

BonusManager: getPetBonus
  when lock duration is 29 days
    should have pet bonus 0 (12.985792ms)mm
  mwhen lock duration is 29 days (14.22325ms)m

  when lock duration is 30 days
    should have pet bonus 0 (6.38775ms)mm
  mwhen lock duration is 30 days (7.1955ms)m

  when lock duration is 31 days
    should have pet bonus 4999999999999999 (4.4305ms)mm
  mwhen lock duration is 31 days (5.116792ms)m

  when lock duration is 32 days
    should have pet bonus 9999999999999999 (7.481417ms)mm
  mwhen lock duration is 32 days (8.054792ms)m

  when lock duration is 33 days
    should have pet bonus 15000000000000000 (4.006084ms)mm
  mwhen lock duration is 33 days (4.634708ms)m

  when lock duration is 34 days
    should have pet bonus 19999999999999999 (3.657041ms)mm
  mwhen lock duration is 34 days (4.1465ms)m

  when lock duration is 35 days
    should have pet bonus 24999999999999999 (3.878458ms)mm
  mwhen lock duration is 35 days (4.333166ms)m

  when lock duration is 36 days
    should have pet bonus 30000000000000000 (4.209125ms)mm
  mwhen lock duration is 36 days (4.660916ms)m

  when lock duration is 37 days
    should have pet bonus 34999999999999999 (3.639875ms)mm
  mwhen lock duration is 37 days (4.079958ms)m
```

```
when lock duration is 38 days
  should have pet bonus 39999999999999999 (4.412917ms)mm
mwhen lock duration is 38 days (4.844ms)m

when lock duration is 39 days
  should have pet bonus 45000000000000000 (3.8015ms)mm
mwhen lock duration is 39 days (4.403625ms)m

when lock duration is 40 days
  should have pet bonus 49999999999999999 (8.670958ms)mm
mwhen lock duration is 40 days (9.197167ms)m

when lock duration is 41 days
  should have pet bonus 54999999999999999 (3.649083ms)mm
mwhen lock duration is 41 days (4.070375ms)m

when lock duration is 42 days
  should have pet bonus 60000000000000000 (3.366083ms)mm
mwhen lock duration is 42 days (3.879458ms)m

when lock duration is 43 days
  should have pet bonus 64999999999999999 (3.486917ms)mm
mwhen lock duration is 43 days (3.905625ms)m

when lock duration is 44 days
  should have pet bonus 69999999999999999 (3.256833ms)mm
mwhen lock duration is 44 days (3.660542ms)m

when lock duration is 45 days
  should have pet bonus 75000000000000000 (3.189583ms)mm
mwhen lock duration is 45 days (3.582917ms)m

when lock duration is 46 days
  should have pet bonus 79999999999999999 (3.532292ms)mm
mwhen lock duration is 46 days (3.977583ms)m

when lock duration is 47 days
  should have pet bonus 84999999999999999 (3.275375ms)mm
mwhen lock duration is 47 days (3.716125ms)m

when lock duration is 48 days
  should have pet bonus 90000000000000000 (4.736917ms)mm
mwhen lock duration is 48 days (5.201167ms)m

when lock duration is 49 days
  should have pet bonus 94999999999999999 (3.512666ms)mm
mwhen lock duration is 49 days (7.951458ms)m

when lock duration is 50 days
  should have pet bonus 99999999999999999 (3.414291ms)mm
mwhen lock duration is 50 days (3.840375ms)m

when lock duration is 51 days
  should have pet bonus 105000000000000000 (3.330708ms)mm
mwhen lock duration is 51 days (3.724417ms)m

when lock duration is 52 days
  should have pet bonus 109999999999999999 (3.024416ms)mm
mwhen lock duration is 52 days (3.568ms)m

when lock duration is 53 days
  should have pet bonus 114999999999999999 (3.279625ms)mm
mwhen lock duration is 53 days (3.787584ms)m

when lock duration is 54 days
  should have pet bonus 120000000000000000 (3.173458ms)mm
mwhen lock duration is 54 days (3.63625ms)m

when lock duration is 55 days
  should have pet bonus 124999999999999999 (3.093291ms)mm
```

```
mwhen lock duration is 55 days (3.543083ms)m

when lock duration is 56 days
  should have pet bonus 129999999999999999 (3.199625ms)mm
mwhen lock duration is 56 days (3.660708ms)m

when lock duration is 57 days
  should have pet bonus 135000000000000000 (3.829209ms)mm
mwhen lock duration is 57 days (4.297334ms)m

when lock duration is 58 days
  should have pet bonus 139999999999999999 (3.772583ms)mm
mwhen lock duration is 58 days (4.205708ms)m

when lock duration is 59 days
  should have pet bonus 144999999999999999 (7.3635ms)mm
mwhen lock duration is 59 days (7.801416ms)m

when lock duration is 60 days
  should have pet bonus 150000000000000000 (3.177583ms)mm
mwhen lock duration is 60 days (3.681292ms)m

when lock duration is 61 days
  should have pet bonus 154999999999999999 (3.313666ms)mm
mwhen lock duration is 61 days (3.781375ms)m

when lock duration is 62 days
  should have pet bonus 159999999999999999 (3.08725ms)mm
mwhen lock duration is 62 days (3.453ms)m

when lock duration is 63 days
  should have pet bonus 165000000000000000 (2.891708ms)mm
mwhen lock duration is 63 days (3.293208ms)m

when lock duration is 64 days
  should have pet bonus 169999999999999999 (3.43525ms)mm
mwhen lock duration is 64 days (4.059916ms)m

when lock duration is 65 days
  should have pet bonus 174999999999999999 (3.7635ms)mm
mwhen lock duration is 65 days (4.238792ms)m

when lock duration is 66 days
  should have pet bonus 180000000000000000 (3.588875ms)mm
mwhen lock duration is 66 days (4.013084ms)m

when lock duration is 67 days
  should have pet bonus 184999999999999999 (2.992375ms)mm
mwhen lock duration is 67 days (3.3415ms)m

when lock duration is 68 days
  should have pet bonus 189999999999999999 (2.9375ms)mm
mwhen lock duration is 68 days (3.295041ms)m

when lock duration is 69 days
  should have pet bonus 195000000000000000 (7.567375ms)mm
mwhen lock duration is 69 days (7.956875ms)m

when lock duration is 70 days
  should have pet bonus 199999999999999999 (3.353709ms)mm
mwhen lock duration is 70 days (3.738666ms)m

when lock duration is 71 days
  should have pet bonus 204999999999999999 (3.181666ms)mm
mwhen lock duration is 71 days (3.524792ms)m

when lock duration is 72 days
  should have pet bonus 210000000000000000 (3.116541ms)mm
mwhen lock duration is 72 days (3.5135ms)m
```

```
  when lock duration is 73 days
    should have pet bonus 214999999999999999 (3.372375ms)mm
  mwhen lock duration is 73 days (3.804917ms)m

  when lock duration is 74 days
    should have pet bonus 219999999999999999 (3.081875ms)mm
  mwhen lock duration is 74 days (3.4285ms)m

  when lock duration is 75 days
    should have pet bonus 225000000000000000 (2.886125ms)mm
  mwhen lock duration is 75 days (3.456334ms)m

  when lock duration is 76 days
    should have pet bonus 229999999999999999 (9.364625ms)mm
  mwhen lock duration is 76 days (9.802917ms)m

  when lock duration is 77 days
    should have pet bonus 234999999999999999 (3.69175ms)mm
  mwhen lock duration is 77 days (4.0915ms)m

  when lock duration is 78 days
    should have pet bonus 240000000000000000 (3.248125ms)mm
  mwhen lock duration is 78 days (3.727583ms)m

  when lock duration is 79 days
    should have pet bonus 244999999999999999 (4.077792ms)mm
  mwhen lock duration is 79 days (4.463167ms)m

  when lock duration is 80 days
    should have pet bonus 249999999999999999 (3.501208ms)mm
  mwhen lock duration is 80 days (3.89975ms)m

  when lock duration is 81 days
    should have pet bonus 255000000000000000 (3.27ms)mm
  mwhen lock duration is 81 days (3.78675ms)m

  when lock duration is 82 days
    should have pet bonus 259999999999999999 (3.408458ms)mm
  mwhen lock duration is 82 days (3.790667ms)m

  when lock duration is 83 days
    should have pet bonus 264999999999999999 (2.973291ms)mm
  mwhen lock duration is 83 days (3.3635ms)m

  when lock duration is 84 days
    should have pet bonus 270000000000000000 (2.815542ms)mm
  mwhen lock duration is 84 days (3.160667ms)m

  when lock duration is 85 days
    should have pet bonus 274999999999999999 (2.792125ms)mm
  mwhen lock duration is 85 days (3.174959ms)m

  when lock duration is 86 days
    should have pet bonus 279999999999999999 (3.4725ms)mm
  mwhen lock duration is 86 days (3.995ms)m

  when lock duration is 87 days
    should have pet bonus 285000000000000000 (3.583458ms)mm
  mwhen lock duration is 87 days (4.024875ms)m

  when lock duration is 88 days
    should have pet bonus 289999999999999999 (8.236584ms)mm
  mwhen lock duration is 88 days (8.820167ms)m

  when lock duration is 89 days
    should have pet bonus 294999999999999999 (3.40375ms)mm
  mwhen lock duration is 89 days (3.768917ms)m

  when lock duration is 90 days
    should have pet bonus 300000000000000000 (3.097208ms)mm
```

```
  mwhen lock duration is 90 days (3.587208ms)m

mBonusManager: getPetBonus (322.087041ms)m

BonusManager: getHarvestBonus
  when bonus comes only from lock bonus
    when lock duration is 30 days
      should have harvest bonus 0 (5.788584ms)mm
    mwhen lock duration is 30 days (6.69425ms)m

    when lock duration is 45 days
      should have harvest bonus 75000000000000000 (4.369334ms)mm
    mwhen lock duration is 45 days (5.064541ms)m

    when lock duration is 60 days
      should have harvest bonus 150000000000000000 (4.181583ms)mm
    mwhen lock duration is 60 days (4.734792ms)m

    when lock duration is 75 days
      should have harvest bonus 225000000000000000 (4.041292ms)mm
    mwhen lock duration is 75 days (4.804209ms)m

    when lock duration is 90 days
      should have harvest bonus 300000000000000000 (3.5985ms)mm
    mwhen lock duration is 90 days (4.085166ms)m

  mwhen bonus comes only from lock bonus (25.760458ms)m

  when bonus comes only from migration bonus
    when locked weighted value = 2x migrated total locked amount
      when migration is claimed
        should return full migration bonus (50.012792ms)mm
        when migration bonus time is past
          should return 0 (97.244ms)mm
        mwhen migration bonus time is past (97.827875ms)m

      mwhen migration is claimed (148.409417ms)m

      when migration is not claimed
        should return 0 (48.648917ms)mm
      mwhen migration is not claimed (49.304458ms)m

    mwhen locked weighted value = 2x migrated total locked amount (197.936292ms)m

    when locked weighted value < 2x migrated total locked amount
      when locked weighted value > half migrated total locked amount
        should return full migration bonus (49.740542ms)mm
      mwhen locked weighted value > half migrated total locked amount (50.349458ms)m

      when locked weighted value = half migrated total locked amount
        should return 0 (49.108792ms)mm
      mwhen locked weighted value = half migrated total locked amount (49.8915ms)m

    mwhen locked weighted value < 2x migrated total locked amount (100.307042ms)m

  mwhen bonus comes only from migration bonus (298.3085ms)m

  when bonus comes only from level bonus
    when 1 NFT with level 1
      should return 1e16 / 200 (17.05625ms)mm
    mwhen 1 NFT with level 1 (17.518959ms)m

    when 5 NFTs with average level 3
      should return 3e16 / 200 (19.003708ms)mm
    mwhen 5 NFTs with average level 3 (19.499084ms)m

    when 6 (max) NFTs with average level 3.5
      should return 3.5e16 / 200 + 3e16 (22.466208ms)mm
    mwhen 6 (max) NFTs with average level 3.5 (22.939792ms)m
```

```
  mwhen bonus comes only from level bonus (60.051167ms)m

  when bonus comes only from munchadex bonus
    when all 125 owned
      should return 100e16 (3.9925ms)mm
    mwhen all 125 owned (4.59325ms)m

    when all 30 in 1 realm owned
      should return 3e16 (8.170667ms)mm
    mwhen all 30 in 1 realm owned (8.679625ms)m

    when all 30 in 4 of 5 realms owned
      should return 12e16 (10.209584ms)mm
    mwhen all 30 in 4 of 5 realms owned (10.613833ms)m

    when 6 unique per realm owned
      should return 2e16 (11.99025ms)mm
    mwhen 6 unique per realm owned (12.428917ms)m

    when 1 unique per realm owned
      should return 1e16 (16.825209ms)mm
    mwhen 1 unique per realm owned (17.269625ms)m

    when all 0 of rarity 0 owned
      should return rarity set bonus 0 (3.873041ms)mm
    mwhen all 0 of rarity 0 owned (4.404375ms)m

    when all 42 of rarity 1 owned
      should return rarity set bonus 3 (5.950208ms)mm
    mwhen all 42 of rarity 1 owned (6.335541ms)m

    when all 23 of rarity 2 owned
      should return rarity set bonus 6 (3.492125ms)mm
    mwhen all 23 of rarity 2 owned (3.869666ms)m

    when all 13 of rarity 3 owned
      should return rarity set bonus 12 (2.826834ms)mm
    mwhen all 13 of rarity 3 owned (3.250958ms)m

    when all 6 of rarity 4 owned
      should return rarity set bonus 20 (3.038708ms)mm
    mwhen all 6 of rarity 4 owned (3.384334ms)m

    when all 1 of rarity 5 owned
      should return rarity set bonus 25 (2.795791ms)mm
    mwhen all 1 of rarity 5 owned (3.190583ms)m

    when combining realm and rarity bonuses
      should return combined 3 + 20 bonus (5.143208ms)mm
    mwhen combining realm and rarity bonuses (5.515875ms)m

  mwhen bonus comes only from munchadex bonus (83.735042ms)m

  when eligible for max bonus from all bonus types
    should return combined max bonus (34.873125ms)mm
  mwhen eligible for max bonus from all bonus types (35.293208ms)m

  when eligible for partial bonus from all bonus types
    should return combined partial bonus (47.697791ms)mm
  mwhen eligible for partial bonus from all bonus types (48.110917ms)m

mBonusManager: getHarvestBonus (9454.975958ms)m

BonusManager: getFeedBonus
  when NFT rarity is invalid
    should revert with InvalidRarityError (16.909958ms)mm
  mwhen NFT rarity is invalid (18.4035ms)m

  when NFT realm is invalid
    should revert with InvalidRealmBonus (8.031875ms)mm
```

```
mwhen NFT realm is invalid (8.8525ms)m

should return correct bonus for rarity 0, NFT realm 0, snuggery realm 0 (8.706791ms)mm
should return correct bonus for rarity 0, NFT realm 0, snuggery realm 1 (13.141375ms)mm
should return correct bonus for rarity 0, NFT realm 0, snuggery realm 2 (6.163667ms)mm
should return correct bonus for rarity 0, NFT realm 0, snuggery realm 3 (6.092ms)mm
should return correct bonus for rarity 0, NFT realm 0, snuggery realm 4 (6.797666ms)mm
should return correct bonus for rarity 0, NFT realm 1, snuggery realm 0 (6.264ms)mm
should return correct bonus for rarity 0, NFT realm 1, snuggery realm 1 (8.1475ms)mm
should return correct bonus for rarity 0, NFT realm 1, snuggery realm 2 (12.239834ms)mm
should return correct bonus for rarity 0, NFT realm 1, snuggery realm 3 (6.994584ms)mm
should return correct bonus for rarity 0, NFT realm 1, snuggery realm 4 (6.431417ms)mm
should return correct bonus for rarity 0, NFT realm 2, snuggery realm 0 (6.023125ms)mm
should return correct bonus for rarity 0, NFT realm 2, snuggery realm 1 (5.973833ms)mm
should return correct bonus for rarity 0, NFT realm 2, snuggery realm 2 (8.289834ms)mm
should return correct bonus for rarity 0, NFT realm 2, snuggery realm 3 (6.825958ms)mm
should return correct bonus for rarity 0, NFT realm 2, snuggery realm 4 (10.11575ms)mm
should return correct bonus for rarity 0, NFT realm 3, snuggery realm 0 (6.647875ms)mm
should return correct bonus for rarity 0, NFT realm 3, snuggery realm 1 (6.267208ms)mm
should return correct bonus for rarity 0, NFT realm 3, snuggery realm 2 (6.146666ms)mm
should return correct bonus for rarity 0, NFT realm 3, snuggery realm 3 (5.9865ms)mm
should return correct bonus for rarity 0, NFT realm 3, snuggery realm 4 (5.7285ms)mm
should return correct bonus for rarity 0, NFT realm 4, snuggery realm 0 (7.297042ms)mm
should return correct bonus for rarity 0, NFT realm 4, snuggery realm 1 (6.546083ms)mm
should return correct bonus for rarity 0, NFT realm 4, snuggery realm 2 (6.828125ms)mm
should return correct bonus for rarity 0, NFT realm 4, snuggery realm 3 (10.198958ms)mm
should return correct bonus for rarity 0, NFT realm 4, snuggery realm 4 (5.71675ms)mm
should return correct bonus for rarity 1, NFT realm 0, snuggery realm 0 (5.531583ms)mm
should return correct bonus for rarity 1, NFT realm 0, snuggery realm 1 (5.253125ms)mm
should return correct bonus for rarity 1, NFT realm 0, snuggery realm 2 (6.088708ms)mm
should return correct bonus for rarity 1, NFT realm 0, snuggery realm 3 (6.520708ms)mm
should return correct bonus for rarity 1, NFT realm 0, snuggery realm 4 (8.967834ms)mm
should return correct bonus for rarity 1, NFT realm 1, snuggery realm 0 (6.278375ms)mm
should return correct bonus for rarity 1, NFT realm 1, snuggery realm 1 (5.273042ms)mm
should return correct bonus for rarity 1, NFT realm 1, snuggery realm 2 (6.0365ms)mm
should return correct bonus for rarity 1, NFT realm 1, snuggery realm 3 (5.939417ms)mm
should return correct bonus for rarity 1, NFT realm 1, snuggery realm 4 (6.192125ms)mm
should return correct bonus for rarity 1, NFT realm 2, snuggery realm 0 (6.397542ms)mm
should return correct bonus for rarity 1, NFT realm 2, snuggery realm 1 (8.748583ms)mm
should return correct bonus for rarity 1, NFT realm 2, snuggery realm 2 (5.261ms)mm
should return correct bonus for rarity 1, NFT realm 2, snuggery realm 3 (5.117125ms)mm
should return correct bonus for rarity 1, NFT realm 2, snuggery realm 4 (5.933291ms)mm
should return correct bonus for rarity 1, NFT realm 3, snuggery realm 0 (5.586292ms)mm
should return correct bonus for rarity 1, NFT realm 3, snuggery realm 1 (5.6275ms)mm
should return correct bonus for rarity 1, NFT realm 3, snuggery realm 2 (9.32025ms)mm
should return correct bonus for rarity 1, NFT realm 3, snuggery realm 3 (5.554542ms)mm
should return correct bonus for rarity 1, NFT realm 3, snuggery realm 4 (5.073625ms)mm
should return correct bonus for rarity 1, NFT realm 4, snuggery realm 0 (4.862959ms)mm
should return correct bonus for rarity 1, NFT realm 4, snuggery realm 1 (5.207ms)mm
should return correct bonus for rarity 1, NFT realm 4, snuggery realm 2 (5.81975ms)mm
should return correct bonus for rarity 1, NFT realm 4, snuggery realm 3 (5.227209ms)mm
should return correct bonus for rarity 1, NFT realm 4, snuggery realm 4 (6.585625ms)mm
should return correct bonus for rarity 2, NFT realm 0, snuggery realm 0 (9.995209ms)mm
should return correct bonus for rarity 2, NFT realm 0, snuggery realm 1 (5.300417ms)mm
should return correct bonus for rarity 2, NFT realm 0, snuggery realm 2 (4.82275ms)mm
should return correct bonus for rarity 2, NFT realm 0, snuggery realm 3 (4.807583ms)mm
should return correct bonus for rarity 2, NFT realm 0, snuggery realm 4 (4.966041ms)mm
should return correct bonus for rarity 2, NFT realm 1, snuggery realm 0 (7.77475ms)mm
should return correct bonus for rarity 2, NFT realm 1, snuggery realm 1 (9.399666ms)mm
should return correct bonus for rarity 2, NFT realm 1, snuggery realm 2 (5.087958ms)mm
should return correct bonus for rarity 2, NFT realm 1, snuggery realm 3 (4.807167ms)mm
should return correct bonus for rarity 2, NFT realm 1, snuggery realm 4 (4.701792ms)mm
should return correct bonus for rarity 2, NFT realm 2, snuggery realm 0 (6.269542ms)mm
should return correct bonus for rarity 2, NFT realm 2, snuggery realm 1 (5.369792ms)mm
should return correct bonus for rarity 2, NFT realm 2, snuggery realm 2 (8.843542ms)mm
should return correct bonus for rarity 2, NFT realm 2, snuggery realm 3 (4.684542ms)mm
should return correct bonus for rarity 2, NFT realm 2, snuggery realm 4 (4.856541ms)mm
should return correct bonus for rarity 2, NFT realm 3, snuggery realm 0 (5.508209ms)mm
should return correct bonus for rarity 2, NFT realm 3, snuggery realm 1 (5.301917ms)mm
should return correct bonus for rarity 2, NFT realm 3, snuggery realm 2 (5.838208ms)mm
```

```
  should return correct bonus for rarity 2, NFT realm 3, snuggery realm 3 (8.456334ms)mm
  should return correct bonus for rarity 2, NFT realm 3, snuggery realm 4 (4.585375ms)mm
  should return correct bonus for rarity 2, NFT realm 4, snuggery realm 0 (4.739125ms)mm
  should return correct bonus for rarity 2, NFT realm 4, snuggery realm 1 (4.870042ms)mm
  should return correct bonus for rarity 2, NFT realm 4, snuggery realm 2 (9.900625ms)mm
  should return correct bonus for rarity 2, NFT realm 4, snuggery realm 3 (10.49125ms)mm
  should return correct bonus for rarity 2, NFT realm 4, snuggery realm 4 (14.781667ms)mm
  should return correct bonus for rarity 3, NFT realm 0, snuggery realm 0 (5.52325ms)mm
  should return correct bonus for rarity 3, NFT realm 0, snuggery realm 1 (5.049833ms)mm
  should return correct bonus for rarity 3, NFT realm 0, snuggery realm 2 (5.025792ms)mm
  should return correct bonus for rarity 3, NFT realm 0, snuggery realm 3 (5.035542ms)mm
  should return correct bonus for rarity 3, NFT realm 0, snuggery realm 4 (5.266833ms)mm
  should return correct bonus for rarity 3, NFT realm 1, snuggery realm 0 (5.490709ms)mm
  should return correct bonus for rarity 3, NFT realm 1, snuggery realm 1 (6.136708ms)mm
  should return correct bonus for rarity 3, NFT realm 1, snuggery realm 2 (8.700916ms)mm
  should return correct bonus for rarity 3, NFT realm 1, snuggery realm 3 (4.93525ms)mm
  should return correct bonus for rarity 3, NFT realm 1, snuggery realm 4 (4.664917ms)mm
  should return correct bonus for rarity 3, NFT realm 2, snuggery realm 0 (4.799625ms)mm
  should return correct bonus for rarity 3, NFT realm 2, snuggery realm 1 (5.445ms)mm
  should return correct bonus for rarity 3, NFT realm 2, snuggery realm 2 (5.392166ms)mm
  should return correct bonus for rarity 3, NFT realm 2, snuggery realm 3 (8.616333ms)mm
  should return correct bonus for rarity 3, NFT realm 2, snuggery realm 4 (4.782459ms)mm
  should return correct bonus for rarity 3, NFT realm 3, snuggery realm 0 (4.856375ms)mm
  should return correct bonus for rarity 3, NFT realm 3, snuggery realm 1 (4.734791ms)mm
  should return correct bonus for rarity 3, NFT realm 3, snuggery realm 2 (4.815167ms)mm
  should return correct bonus for rarity 3, NFT realm 3, snuggery realm 3 (4.729ms)mm
  should return correct bonus for rarity 3, NFT realm 3, snuggery realm 4 (8.294167ms)mm
  should return correct bonus for rarity 3, NFT realm 4, snuggery realm 0 (4.949ms)mm
  should return correct bonus for rarity 3, NFT realm 4, snuggery realm 1 (5.120292ms)mm
  should return correct bonus for rarity 3, NFT realm 4, snuggery realm 2 (4.989917ms)mm
  should return correct bonus for rarity 3, NFT realm 4, snuggery realm 3 (5.48275ms)mm
  should return correct bonus for rarity 3, NFT realm 4, snuggery realm 4 (4.810167ms)mm
  should return correct bonus for rarity 4, NFT realm 0, snuggery realm 0 (8.392333ms)mm
  should return correct bonus for rarity 4, NFT realm 0, snuggery realm 1 (4.544583ms)mm
  should return correct bonus for rarity 4, NFT realm 0, snuggery realm 2 (5.164541ms)mm
  should return correct bonus for rarity 4, NFT realm 0, snuggery realm 3 (5.520583ms)mm
  should return correct bonus for rarity 4, NFT realm 0, snuggery realm 4 (4.673458ms)mm
  should return correct bonus for rarity 4, NFT realm 1, snuggery realm 0 (4.805958ms)mm
  should return correct bonus for rarity 4, NFT realm 1, snuggery realm 1 (4.746291ms)mm
  should return correct bonus for rarity 4, NFT realm 1, snuggery realm 2 (7.924875ms)mm
  should return correct bonus for rarity 4, NFT realm 1, snuggery realm 3 (4.706041ms)mm
  should return correct bonus for rarity 4, NFT realm 1, snuggery realm 4 (5.224ms)mm
  should return correct bonus for rarity 4, NFT realm 2, snuggery realm 0 (5.360625ms)mm
  should return correct bonus for rarity 4, NFT realm 2, snuggery realm 1 (5.289125ms)mm
  should return correct bonus for rarity 4, NFT realm 2, snuggery realm 2 (6.444416ms)mm
  should return correct bonus for rarity 4, NFT realm 2, snuggery realm 3 (5.944375ms)mm
  should return correct bonus for rarity 4, NFT realm 2, snuggery realm 4 (8.65625ms)mm
  should return correct bonus for rarity 4, NFT realm 3, snuggery realm 0 (4.632167ms)mm
  should return correct bonus for rarity 4, NFT realm 3, snuggery realm 1 (4.593166ms)mm
  should return correct bonus for rarity 4, NFT realm 3, snuggery realm 2 (4.341ms)mm
  should return correct bonus for rarity 4, NFT realm 3, snuggery realm 3 (4.499125ms)mm
  should return correct bonus for rarity 4, NFT realm 3, snuggery realm 4 (4.502166ms)mm
  should return correct bonus for rarity 4, NFT realm 4, snuggery realm 0 (9.026209ms)mm
  should return correct bonus for rarity 4, NFT realm 4, snuggery realm 1 (4.483416ms)mm
  should return correct bonus for rarity 4, NFT realm 4, snuggery realm 2 (6.146667ms)mm
  should return correct bonus for rarity 4, NFT realm 4, snuggery realm 3 (4.501542ms)mm
  should return correct bonus for rarity 4, NFT realm 4, snuggery realm 4 (4.782375ms)mm
  should return correct bonus for rarity 5, NFT realm 0, snuggery realm 0 (5.130292ms)mm
  should return correct bonus for rarity 5, NFT realm 0, snuggery realm 1 (9.561459ms)mm
  should return correct bonus for rarity 5, NFT realm 0, snuggery realm 2 (4.434792ms)mm
  should return correct bonus for rarity 5, NFT realm 0, snuggery realm 3 (4.49725ms)mm
  should return correct bonus for rarity 5, NFT realm 0, snuggery realm 4 (4.336833ms)mm
  should return correct bonus for rarity 5, NFT realm 1, snuggery realm 0 (4.325ms)mm
  should return correct bonus for rarity 5, NFT realm 1, snuggery realm 1 (4.835125ms)mm
  should return correct bonus for rarity 5, NFT realm 1, snuggery realm 2 (9.122375ms)mm
  should return correct bonus for rarity 5, NFT realm 1, snuggery realm 3 (4.752167ms)mm
  should return correct bonus for rarity 5, NFT realm 1, snuggery realm 4 (5.3135ms)mm
  should return correct bonus for rarity 5, NFT realm 2, snuggery realm 0 (4.376667ms)mm
  should return correct bonus for rarity 5, NFT realm 2, snuggery realm 1 (4.332083ms)mm
  should return correct bonus for rarity 5, NFT realm 2, snuggery realm 2 (5.108833ms)mm
```

```
    should return correct bonus for rarity 5, NFT realm 2, snuggery realm 3 (8.757167ms)mm
    should return correct bonus for rarity 5, NFT realm 2, snuggery realm 4 (5.337375ms)mm
    should return correct bonus for rarity 5, NFT realm 3, snuggery realm 0 (5.395708ms)mm
    should return correct bonus for rarity 5, NFT realm 3, snuggery realm 1 (4.974208ms)mm
    should return correct bonus for rarity 5, NFT realm 3, snuggery realm 2 (5.374666ms)mm
    should return correct bonus for rarity 5, NFT realm 3, snuggery realm 3 (5.734833ms)mm
    should return correct bonus for rarity 5, NFT realm 3, snuggery realm 4 (5.136291ms)mm
    should return correct bonus for rarity 5, NFT realm 4, snuggery realm 0 (8.652667ms)mm
    should return correct bonus for rarity 5, NFT realm 4, snuggery realm 1 (5.309959ms)mm
    should return correct bonus for rarity 5, NFT realm 4, snuggery realm 2 (4.807417ms)mm
    should return correct bonus for rarity 5, NFT realm 4, snuggery realm 3 (5.043959ms)mm
    should return correct bonus for rarity 5, NFT realm 4, snuggery realm 4 (4.684166ms)mm
  mBonusManager: getFeedBonus (2847.816292ms)m

  AccountManager: subacount functionality
    successful: add and remove multiple subaccounts
      add multiple subaccounts (32.272834ms)mm
      add/remove multiple subaccounts out of order (18.62675ms)mm
      add/remove multiple subaccounts out of order back to (18.222292ms)mm
      remove all sub accounts (16.291208ms)mm
    msuccessful: add and remove multiple subaccounts (88.024333ms)m

    subaccount calls when paused
      reject addSubaccount when paused (50.891ms)mm
      reject removeSubaccount when paused (4064.120959ms)mm
    msubaccount calls when paused (4118.732084ms)m

    subaccount does malintent
      tries to add subaccount twice (26.960959ms)mm
      tries to register as player (18.828167ms)mm
      tries to remove itself as subaccount (11.933ms)mm
      tries to remove subaccount that doesn't exist (13.814834ms)mm
      tries to add subaccount (8.591916ms)mm
    msubaccount does malintent (83.879542ms)m

    subaccount does good functions
      subaccount harvests (23.248542ms)mm
    msubaccount does good functions (23.840708ms)m

    subaccount additional getter tests
      getSubAccounts (21.661667ms)mm
    msubaccount additional getter tests (22.173042ms)m

  mAccountManager: subacount functionality (4364.059875ms)m

  AccountManager: spray schnibbles functionality
    improper proposal calls
      spraySchnibblesPropose() - not social role (8.779917ms)mm
      spraySchnibblesPropose() - empty players (21.859708ms)mm
      spraySchnibblesPropose() - inconsistent parameters (5.619333ms)mm
      spraySchnibblesPropose() - duplicate sprayers (3.7485ms)mm
      spraySchnibblesPropose() - too many inputs (6.334375ms)mm
      spraySchnibblesPropose() - double proposals (7.276291ms)mm
      execSprayProposal() - not social role (3.683ms)mm
      execSprayProposal() - empty proposal (2.851666ms)mm
      removeSprayProposal() - not social role (2.609375ms)mm
      removeSprayProposal() - empty proposal (2.949083ms)mm
    mimproper proposal calls (74.802792ms)m

    proper proposal calls
      spraySchnibblesPropose() and execSprayProposal() and claim after register (25.812417ms)mm
      spraySchnibblesPropose() and removeSprayProposal() (16.120583ms)mm
    mproper proposal calls (42.950209ms)m

  mAccountManager: spray schnibbles functionality (145.364375ms)m

  AccountManager: register functionality
    improper use
      should revert if the player is already registered (18.648ms)mm
      should revert if the player puts an invalid realm (15.072458ms)mm
```

```
  mimproper use (36.255541ms)m

  proper register
    should register a player (5.578292ms)mm
  mproper register (6.232042ms)m

mAccountManager: register functionality (69.160375ms)m

AccountManager: inaccessible external functions check
  try calling functions that are only accessible via contracts or never accessible
    initialize() - proxy (18.336666ms)mm
    initialize() - base (12.65575ms)mm
    updatePlayer() (7.770166ms)mm
    forceHarvest() (6.982708ms)mm
    configUpdated() (6.144667ms)mm
  mtry calling functions that are only accessible via contracts or never accessible (58.884917ms)m

mAccountManager: inaccessible external functions check (85.545625ms)m

AccountManager: harvest functionality
  improper harvest calls
    harvest() - paused (55.193625ms)mm
    harvest() - player not registered (6.290708ms)mm
  mimproper harvest calls (63.379042ms)m

  harvest() - successful
    no-bonus harvest (23.312ms)mm
  mharvest() - successful (24.030667ms)m

mAccountManager: harvest functionality (127.594ms)m

Example Test
  Test 1 (10.415625ms)mm
  Test 2 (7.599333ms)mm
mExample Test (43.958083ms)m

ConfigStorage: Roles
  add/remove roles
    add role to a specific contract (12.655833ms)mm
    add universal role (9.075625ms)mm
  madd/remove roles (24.064125ms)m

mConfigStorage: Roles (49.617084ms)m

ConfigStorage: Read/Write
  reading/writing standard variables
    set/getUint (93.636208ms)mm
    set/getUintArray (88.9475ms)mm
    set/getSmallUintArray (84.46325ms)mm
    set/getSmallInt (81.658708ms)mm
    set/getSmallIntArray (84.4535ms)mm
    set/getBool (83.216208ms)mm
    set/getAddress (85.562291ms)mm
    setAddresses/getAddress (4089.011ms)mm
    set/getAddressArray (120.4195ms)mm
    set/getBytes32 (95.533667ms)mm
  mreading/writing standard variables (4917.703708ms)m

mConfigStorage: Read/Write (4943.966917ms)m

ConfigStorage: Notify
  notifications
    receives notification after changing config (48.704ms)mm
    does not receive notification if notify set to false (6.625834ms)mm
    notifications stop after being removed (47.553958ms)mm
  mnotifications (105.674ms)m

mConfigStorage: Notify (290.069458ms)m

ConfigStorage: Auth
```

```
   unauthorised setting variables
     setUint (6.1195ms)mm
     setUintArray (24.2915ms)mm
     setSmallUintArray (2.613292ms)mm
     setSmallInt (2.380833ms)mm
     setSmallIntArray (1.960584ms)mm
     setBool (1.799792ms)mm
     setAddress (2.653583ms)mm
     setAddresses (2.482334ms)mm
     setAddressArray (2.540083ms)mm
     setBytes32 (2.137958ms)mm
   munauthorised setting variables (57.6135ms)m

 mConfigStorage: Auth (83.528292ms)m

- tests 659
- suites 359
- pass 659
- fail 0
- cancelled 0
- skipped 0
- todo 0
- duration_ms 373448.873709

> munchables-common-core@1.0.0 test:solidity /munchables-common-core
> forge test

No files changed, compilation skipped

Ran 1 test for src/test/MigrationManager.t.sol:MrMigrate
[PASS] test_MigrateNFTs() (gas: 63451445)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 18.87ms (13.82ms CPU time)

Ran 1 test for src/test/SpeedRun.t.sol:MrTester
[PASS] test_SpeedRun() (gas: 65950174)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 22.16ms (17.22ms CPU time)
```

10 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.