Scope of Work

The scope of this audit is based on the material provided by the client. The goal of this audit is to ensure the following:

- Find potential exploits for the contract
- Find issues in the contract that will affect the integrity of the mint and claim processes
- Ensure the contract adheres to the business logics provided by the client

The logic of the contract will be compared to the business logic contained in the materials provided below:

- 5 contract was provided in total including
  3 main contracts:
  - PixelmonSponsoredTrips
  - PixelmonTrainerAdventure
  - PixelmonTrainerGear
  2 library contracts included to `PixelmonTrainerAdventure`:
  - WinnerSelectionManager
  - Utils
- Short project logic description with additional information related to main contracts.

The contracts code was provided on 20th January 2023 with an additional info and description on 25th January 2023. Audit report was delivered on 29th January 2023. It was given in the following formats:

- **PDF document**

Disclaimer

This is a limited report based on our analysis of the Smart Contract audit and performed in accordance with best practices as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the smart contract source code analyzed, the details of which are set out in this report, (hereinafter "Source Code") and the Source Code compiling, deploying and performing the intended functions. In order to get a full view of our findings and the scope of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions.

Summary

Findings summary

	Critical	Medium	Low
PixelmonTrainerAdventure	0	0	1
WinnerSelectionManager	0	0	0
Utils	0	0	1
PixelmonTrainerGear	0	0	0
PixelmonSponsoredTrips	0	0	0

The contracts are well-written and complies with Solidity development best practices;

No critical or medium flaws were found in the provided contracts. However, you can find a few comments based on auditor assumptions as these moments were not covered with delivered documentation. On the next pages, they are marked as low impact problems and need to be confirmed by the contract developer first.

PixelmonTrainerAdventure

getTreasureVaultInfo filtering issues

Description:

Function accepts 3 input parameters:

```
/// @param _collectionAddress Origin of the treasure
/// @param _tokenIds Treasure token ID
/// @param _status Treasure state (0: Invalid, 1: In vault, 2: In pool, 3: Claimed, 4: Rescued)
```

And will return Treasure info only in case if all 3 parameters fit token found.

```
treasureVault[index].collectionAddress == _collectionAddress &&
IsdataExistsInArray(_tokenIds, treasureVault[index].tokenId) &&
IsdataExistsInArray(_status, treasureVault[index].status)
```

Which means that to get all claimed tokens of a collection, you need to provide their IDs, and vice versa. To get information about a token ID, you need to provide its status. Additionally, it makes it not possible to get information about different tokens with different statuses.

Proposed solution:

Replace the code above with

```
treasureVault[index].collectionAddress == _collectionAddress &&
(IsdataExistsInArray(_tokenIds, treasureVault[index].tokenId) ||
IsdataExistsInArray(_status, treasureVault[index].status))
```

Which will work with tokens if status **or** ID is the same as in input parameters.

Utils

getRandomNumber uses onchain random generation

Description:

Chainlink RNG is not used in `Utils.getRandomNumber` used in `claimTreasure` function

```
    function getRandomNumber() internal view returns (uint256) {
        uint256 randomNumber = uint256(
            keccak256(
                abi.encodePacked(
                    block.timestamp +
                        block.difficulty +
                                    ((uint256(keccak256(abi.encodePacked(block.coinbase)))) +
                                    (block.timestamp)) +
                        block.gaslimit +
                                                ((uint256(keccak256(abi.encodePacked(msg.sender))) +
                                                (block.timestamp)) +
                                            block.number
                )
            )
        );

        return randomNumber;
    }
```

Most likely, it was made for test purposes only, so make sure you've changed it back before deploying to mainnet.

Also, note that Chainlink has its contracts deployed on the Goerli testnet, so it's recommended to perform tests with the finalized contracts and Chainlink integration.