**ERC721 & Crypto-Collectibles**

ERC721 & Crypto-Collectibles is the fifth course on cryptozombies.io which teaches about tokens, libraries, overflow prevention and commenting conventions.

Tokens are the most interesting concept covered in this course, but also a simple one to understand. Tokens are simply smart contracts that implement a set of functions, and can be viewed as a means to facilitate custom currency transfers (or valuable entities such as zombies). Typically, tokens will contain some sort of transfer method, for example ERC20 will implement the following function.

* function transfer(address \_to, uint256 \_value) public returns (bool success)

However, ERC721 tokens (which is also implemented in this course) will implement the following, slightly different function.

* function transferFrom(address \_from, address \_to, uint256 \_tokenId) external payable

The largest difference between the two tokens is that the first one acts more like the common currencies we know, such as bitcoin, and can be sent and received in arbitrary amounts, while the second one sends an id between addresses. Each id represents an object on the blockchain. It might be the zombies we are playing with in the course, the bored ape NFT, or other similar ideas you may have heard about. The ERC721 token used in the course has its purpose in keeping track of how a user may transfer their zombie to someone else, and in doing so the game has some core functionality: zombie leveling, attacking and transferring.

A good idea when implementing tokens Is to make sure you can’t send currency to invalid addresses, or address 0. Sending anything to address 0 will simply burn the token, so it’s best to ensure that won’t be possible.

Libraries were also introduced, although for a simple example that simply adds new methods to the uint native data types. It is recommended that we protect against overflows and underflows, corrupting the data can have large impacts and in the worst cases break the contract, so it’s best to just always use safe math even if it feels unnecessary. SafeMath is the library that implements these protections by asserting that the operations didn’t yield unexpected results. For example, when adding two numbers the check used is that the answer is not less than the first number, which is enough in the case of unsigned integers.

You will also notice the use of assert instead of require, the difference pointed by the course is that require will refund gas if it fails, where as assert will not. It is recommended that assert be used for internal errors and checking invariants, and require for situations that a user controls and cannot be assumed true by the developer.

The last subject the course introduces is comment conventions, multiline comments, and comment tags that specialize explanations based on what exactly is elaborated. There are @title and @author which are self-explanatory, @notice which explains to the user what the function/contract does, @dev that explains that to the developers, and lastly @param and @return which explain the usage for the parameters and return variables used. A well-documented code is always preferable, and can be more easily maintained by colleagues, or even you after a while from programming on the project.

This concludes the fifth course on cryptozombies.io, rounding up even more the concept of blockchain, its uses, a fun zombie game that puts everything in practice and most importantly the solidity programming language. The possibilities are limitless when the blockchain powers your application!