**Tokens on Ethereum**

Let's talk about **tokens**.

If you've been in the Ethereum space for any amount of time, you've probably heard people talking about tokens — specifically **ERC20 tokens**.

A **token** on Ethereum is basically just a smart contract that follows some common rules — namely it implements a standard set of functions that all other token contracts share, such as transferFrom(address \_from, address \_to, uint256 \_tokenId) and balanceOf(address \_owner).

Internally the smart contract usually has a mapping, mapping(address => uint256) balances, that keeps track of how much balance each address has.

So basically a token is just a contract that keeps track of who owns how much of that token, and some functions so those users can transfer their tokens to other addresses.

Why does it matter?

Since all ERC20 tokens share the same set of functions with the same names, they can all be interacted with in the same ways.

This means if you build an application that is capable of interacting with one ERC20 token, it's also capable of interacting with any ERC20 token. That way more tokens can easily be added to your app in the future without needing to be custom coded. You could simply plug in the new token contract address, and boom, your app has another token it can use.

One example of this would be an exchange. When an exchange adds a new ERC20 token, really it just needs to add another smart contract it talks to. Users can tell that contract to send tokens to the exchange's wallet address, and the exchange can tell the contract to send the tokens back out to users when they request a withdraw.

The exchange only needs to implement this transfer logic once, then when it wants to add a new ERC20 token, it's simply a matter of adding the new contract address to its database.

Other token standards

ERC20 tokens are really cool for tokens that act like currencies. But they're not particularly useful for representing zombies in our zombie game.

For one, zombies aren't divisible like currencies — I can send you 0.237 ETH, but transfering you 0.237 of a zombie doesn't really make sense.

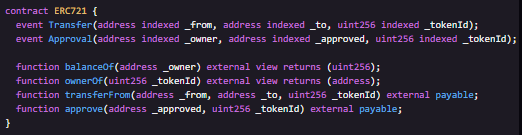
Secondly, all zombies are not created equal. Your Level 2 zombie "**Steve**" is totally not equal to my Level 732 zombie "**H4XF13LD MORRIS 💯💯😎💯💯**". (Not even close, *Steve*).

There's another token standard that's a much better fit for crypto-collectibles like CryptoZombies — and they're called **ERC721 tokens.**

**ERC721 tokens** are **not** interchangeable since each one is assumed to be unique, and are not divisible. You can only trade them in whole units, and each one has a unique ID. So these are a perfect fit for making our zombies tradeable.

*Note that using a standard like ERC721 has the benefit that we don't have to implement the auction or escrow logic within our contract that determines how players can trade / sell our zombies. If we conform to the spec, someone else could build an exchange platform for crypto-tradable ERC721 assets, and our ERC721 zombies would be usable on that platform. So there are clear benefits to using a token standard instead of rolling your own trading logic.*

**ERC721 Standard**



**ERC721: Transfer Logic**

Great, we've fixed the conflict!

Now we're going to continue our ERC721 implementation by looking at transfering ownership from one person to another.

Note that the ERC721 spec has 2 different ways to transfer tokens:

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

and

function approve(address \_approved, uint256 \_tokenId) external payable;

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

1. The first way is the token's owner calls transferFrom with his address as the \_from parameter, the address he wants to transfer to as the \_to paramater, and the \_tokenId of the token he wants to transfer.
2. The second way is the token's owner first calls approve with the address he wants to transfer to, and the \_tokenID . The contract then stores who is approved to take a token, usually in a mapping (uint256 => address). Then, when the owner or the approved address calls transferFrom, the contract checks if that msg.sender is the owner or is approved by the owner to take the token, and if so it transfers the token to him.

Notice that both methods contain the same transfer logic. In one case the sender of the token calls the transferFrom function; in the other the owner or the approved receiver of the token calls it.

So it makes sense for us to abstract this logic into its own private function, \_transfer, which is then called by transferFrom.