

Doggochain — A doggo game blockchain

2nd report

Guilherme Christopher Michaelson Cardoso (14100831)
João Paulo Taylor Ienczak Zanette (14200743)

November 19, 2019

1 Contracts structure

This section will only describe contracts briefly just to clarify **what** they are (i.e. their role in the system). Further explanations of **how** they behave will be cleared in Section 2.

In short, the contracts directly related to Doggochain are structured as follows:

DoggoChain: The main contract, hence responsible for managing database in respect of Players and Doggos, as well as the smart-contract's operations (player challenging, hunting, breeding, etc.). Abstracts player's data as an internal **struct**, since there's no much use for it outside this contract¹.

Doggo: Works as a simple structure for every Doggo's stats, providing an interface that abstract how Doggos evolve (as in stat gaining) and what's their real current stats.

UsageTest: A unit test to check if the system works properly from the point of view of who's going to *use* the API.

Other implementation-specific contracts are designed to compensate some lack of features from Solidity 0.5, e.g. no proper dynamic array handling in storage data. Currently these contracts are:

DoggoList: A simple list data structure implementation, simulated using a mapping from **uint32** to **Doggo** (since there's no generics in Solidity, this implementation must be replicated for every different type) and a **length** variable for element counting.

¹The reason is that the **Player** structure is needed only for players to own their Doggos and mark their existence in the block-chain (i.e. in the game). Every other operation is done considering Doggo-to-Doggo interactions.

OptionalPlayer: Actually not a contract itself, but a structure. Used to compensate the lack of Optional type² in Solidity. Needed since Solidity mappings assume *all* possible keys actually exist and maps to some value (meaning no **hasKey**-related checking), and it does not belong directly to **Player** structure since existence itself is not a player’s property, but rather is a mapping’s constraint.

Apart from that, there are libraries for the more complex operations, so the amount of responsibility in DoggoChain contract’s code becomes reduced:

Battle: For battle simulation. Its only public function is **simulate**, which gives a simulated battle’s results.

Breed: For breeding operations. Its only public function is **breed**, which gives result containing the (possibly) generated Doggo and if the breeding was a success.

2 Function details

This section will be splitted in a subsection for each **main contract and libraries**, since other contracts are merely for implementation details.

2.1 DoggoChain

²An “Optional” is a type that handles existence/inexistence of data (without the security/usage issues of using **null**-like value).