Dragonereum

Digital scarcity game with blockchain and dragons

November 28, 2018

Version 1.1.0

Abstract

Dragonereum is a cryptocollectible game whereby users can own a dragon, trade their dragons, interbreed them and battle other dragons, collecting rewards and achievements along the way. All of these are done on the blockchain in an open, trusted and decentralized manner.

Our aim is to build the first truly decentralized digital scarcity game, where developers do not have any control over the game once it is deployed. The community will have the ability to verify all the transactions and actions without the involvement of trusted third parties.

This document discusses the technological and economic implementation of the system. It aims to describe such elements as: random number generation, decentralized breeding algorithm, battle implementation, game mechanics, and a broader economic model which emphasizes the strength of a collectible game built on blockchain.

In this document, we propose the philosophical background, technical foundations and the economic model of Dragonereum, whereby dragon supply is limited by the proposed game mechanics.

Contents

1 Perspective

The appearance of decentralized games defined a new era for digital collectibles: players no longer have to rely on a trusted third party which issues assets in order to preserve the rarity of their belongings, but can fully rely on the autonomous, censorship resistant, trustless and transparent entity - the code of the smart contract.

However, the current offerings lack many important features of the decentralized technology, which limits the player's experience to just storing assets securely on the blockchain:

- The scope of actions one can do with his or her items is limited. Most cryptocollectible games function without a game mechanics or carefully crafted game ecosystem.
- Not properly set supply mechanisms. Unlimited supply reduces the attractiveness of a particular cryptocollectibe.
- The development teams retain full control over the released product and can add new custom-built assets at any time (though within some articulated limits) or even shut down the entire project.
- The presale monetization model is widely used thus shifting the development team's efforts from game development to marketing and PR.
- Additionally, the current solutions still heavily rely upon a centralized infrastructure, which diminishes the advantages of blockchain usage and creates a weak spot and a single point of failure.

We propose a system which is built upon the basics of current technology, with advancements such as rich gamification and an in-game ecosystem with its own in-game cryptocurrency.

The supply of cryptocollectibles is limited by the proposed game design, and, following the game's release, will no longer be under the team's control. Once the game is formally released, the team will relinquish the control of token issue.

An improved free-to-play model is utilized where developers do not collect any fees from players for game assets and the application is built to serve with minimal ties to any servers.

Our long-term goal is to create a fully decentralized game, leveraging all of the unique advantages of the blockchain. More on this in 8.1.

In the following, we describe how we aim to achieve those goals. In Section 2 we give an overview of Dragonereum's game ecosystem. In Section 3 we describe the technological limitations faced by Dragonereum and how they are overcome, we outline how we obtain randomness and present the genetics and diversity of our dragon population. Section 6 explains in detail the game mechanics, and how the in-game currency is obtained and distributed. The implementation of the game is described in Section 8, and the possible

platforms with which Dragonereum can be integrated in Section 9. Lastly in Section 10 and Section 11 the details of the release of the genesis eggs and the Gold allocation are presented correspondingly.

2 Introduction to Dragonereum

Dragonereum is a cryptocollectible, person vs person game whereby users collect and breed unique dragons, engaging in battles along the way.

With our initial version of the decentralized application (dapp), a user will have the possibility to own and manage a dragon or a thunder of dragons. In addition to breeding and battling their dragons, users can buy and sell dragons, collect rewards, compete with other players for in-game achievements and make deals on the game's marketplace.

Each dragon is owned and controlled by a particular address on the Ethereum network. Dragons are securely stored on the blockchain and cannot be modified, replicated or destroyed by any third party.

Every dragon has its unique set of genes which determines all features and traits of the dragon.

Therefore, a player will have to choose a type of dragon to control (Figure 1), decide on a battle strategy, determine which skills to train with experience earned in battles and how to get better offsprings with further improved *basic skills* (*attack, defense, stamina, speed,* and *intelligence*).

Figure 1: A selection of different types of dragons. Each row represents two varieties of the fire, ice, earth, air, and cyber type.

The player's goal might be presented by two paths: a dragon training master or a dragon breeding master. The dragon training master will be an expert in leveling up dragons so he/she can get the strongest dragon and earn in-game rewards as a result of winning battles. The dragon breeding master will know how to interbreed dragons in order to get highly unique and valuable dragons which might be sold on the game's marketplace. Additionally there might be some other unforeseen game paths beneficial to players which will be discovered by them.

The game starts with the purchase of a dragon egg or a dragon from another player or during the initial distribution of genesis eggs. In the case of an egg, a user knows the

egg's parents, but not what kind of dragon will appear from the egg, as this is decided during the egg incubation in accordance with the rules set by Dragonereum's genetics.

After receiving a certain amount of experience and upgrading to the next level, a dragon will be able to either improve its skills or to breed. At that point, the user will have to decide whether to upgrade a dragon's skillset or breed it in order to acquire a new dragon.

Offsprings will rarely outperform their parents right from the hatchery. However, their initial skills (at level 0) would generally be higher than the initial skills of the parent dragons. Therefore, with some training and a few level upgrades, the dragon will eventually outperform its parents.

As in the real world, occasionally there might be a mutation which will cause some of the traits of a new dragon to be much weaker or much stronger than those of its parents. Mutations affect individual genes, so any trait can receive a boost or a skill reset during breeding, but no one will be able to predict which gene will be affected and when.

As going up levels will require even more experience, it will become increasingly difficult to get any dragon's descendants, thus limiting the total population of dragons and avoiding exponential growth.As such, the system is designed to encourage users to be active, and allow them to choose the winning strategy of their own choice.

There are five different dragon types (air, water, earth, fire, and cyber). Dragons of different types can interbreed, creating unique offsprings, as can be seen in Figure 2.

Every dragon has the ability to participate in battles. There will be two types of battles: regular battles and gladiator battles. Regular battles are described below, where they are referred to simply as battles, while gladiator battles will be outlined at the end of this section.

Taking into account the fact that battles are held on-chain they are represented not by a conjoined combat of two dragons but rather by an attack of one active dragon on another passive dragon. When a user wants their dragon to engage in a battle, providing their dragon has enough Health Points (HP), the application offers a choice of dragons of similar Dragon Strength (DS) with which it can battle. The DS is described in detail in Section 6.8. Once a user chooses an opponent, the transaction with the battle is sent to the blockchain. As soon as the transaction is mined and included into the block, a user can see an animated battle together with the battle results[1].

The attacking dragon always has an advantage as it can adjust its battle tactics according to its opponent's skills. Defending dragons use the default battle tactics defined previously by the owner. All battle results are dependent on the overall skills of the battling dragons, as well as the battle tactics used during the battle. The battle tactics can be defined by

[1]The defending dragon's owner will similarly have the opportunity to see the animated battle once they go online

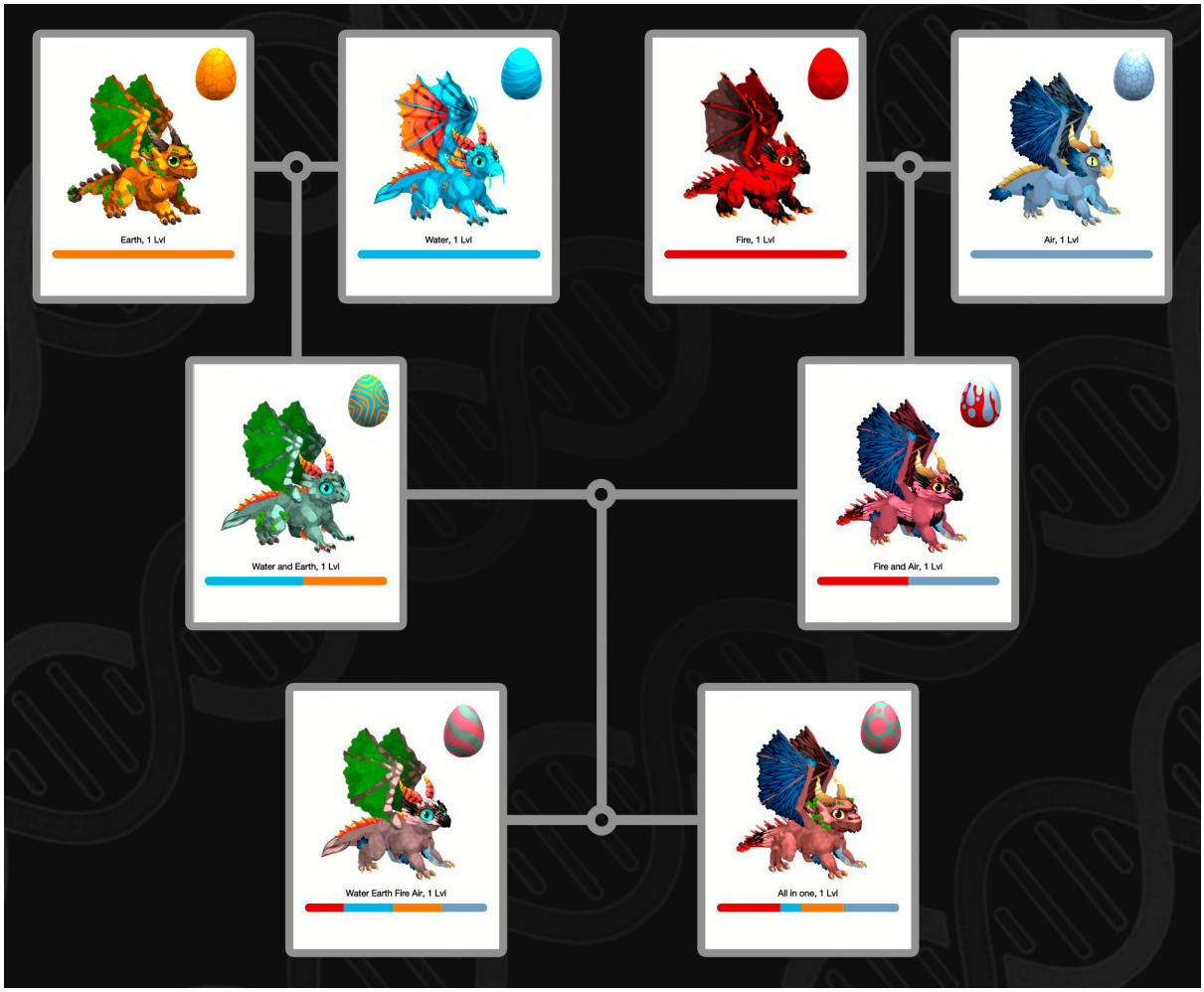

Figure 2: Possible offsprings produced by dragons of different types.

two sliders:

- The attack slider selects a value from a range between melee (short-distance attack) and ranged (long-distance attack)
- The action slider selects a value from a range between defense and attack

Every move during a battle is determined by a weighted random number with the value of the weight determined by the positioning of the sliders (for example if the Action Slider is in the position of 20% defense and 80% attack - the dragon will make on average 20% defense moves and 80% attack moves). The dragons with faster speeds always make the first move.

Based on the battle tactics, settings and results of the RNG, dragons are able to:

- Change position
- Attack
- Stand on the defensive
- Use special attack or special defense

Once the faster dragon has made the first move, the opponent has its turn. This continues until the HP for one of the dragons are depleted or the battle exceeds the move limit. If the move limit is exceeded or available gas is depleted, the dragon with higher percentage of HP compared to initial level of HP wins. If the dragons have an equal amount of HP left, the dragon which made the first move will be a winner.

The defeated dragon does not receive any points or bonuses. It only receives a flag protecting it from further attacks, which is valid for one day only.

The higher the intelligence level, the greater the chances of a dragon exploiting its *special attack* or *special defense* skills during the battle.

Special attack or *special defense* is the ability to use the special powers available for each type of dragon. It improves attack or defense during a move. These special moves use Mana.

The results of a battle will be displayed in a rich visual animation, which can be replayed and shared with friends.

A battle winner receives Experience Points (XP). When the required quantity of XP are earned, the dragon gets a level up and receives some DNA Points (DP), which can be used for breeding or converted into body part level ups and thereby improvements of skills.

Additionally, if the attacking dragon wins, it is credited with Gold as a bonus. Gold is an in-game cryptocurrency which will be distributed to winners from the game's balance, and the ticker symbol for it is denoted by GOLD. The game's creators will not have any control over Gold stored in the Dragonereum's Treasury (which is one of the game's smart contracts) once the control over the upgradability and halting mechanism will be transferred to the 0x000..0001 address (more details can be found in Section 8.1).

As Gold has limited availability, the reward for battle winners will decrease over time. The value of the reward will be aligned with the number of dragons in the population, Dragon Skillfulness Index (described in Section 6.5) and the amount of Gold in the Treasury. Once the initial balance in the game's Treasury becomes depleted, the reward adjusts itself according to the sum of fees collected. These are collected for egg incubation and additional in-game features. Consequently, the first players will have an advantage with regards to Gold accumulation.

The only price set by Dragonereum is the amount of Gold required to incubate an egg. This way, Dragonereum controls the number of incubations along with the quantity of new dragons in its overall population.

Gold can also be spent on in-game services which will be offered by other players or a system. This will create a platform for social interactions between dragon owners.

Furthermore, as an additional incentive for players to get higher valued dragons and to engage in battles, there will be an internal ranking system, represented by a system of leaderboards, where players can keep track of their achievements and how they score against other dragon owners.

As already mentioned the battles described above are regular battles. Those battles allow players to earn XP and Gold from the Treasury.

Here, we will briefly overview gladiator battles, which are technically the same battles as regular battles, but they give players the possibility to place a bet on one of the dragons. They do not let players earn XP or Gold from the Treasury. The simplified process of the gladiator battle can be presented as follows:

- A player decides to offer a dragon on the gladiator battle. He/she offers a dragon together with the amount of Eth or Gold he/she would like to place on the battle.
- Other players can offer their dragons for the battle together with their bets.
- The player who initiated the battle confirms one of the offers once the time is up for offers.
- Once opponents are set, other players (not necessarily those who participate in this battle) will be able to place bets.
- Once the timer set during the battle initiation has expired, any player can finalize the battle.

3 Game Design Constraints

As decentralized technologies are in their early period of development, the game was designed with several limitations in mind.

First of all, the Ethereum network is currently limited to approximately 8M of gas or 15 tx/sec per block. Therefore, we had to limit the dragon population and gas-intensive events (battles, level ups, breeding, incubation, etc.) so that the game's transactions will take just a part of the Ethereum's bandwidth.

This is achieved by offering a limited quantity of genesis eggs and constraining the breeding speed which will limit the supply of new dragons and thus keep the overall game bandwidth within the Ethereum's network capacity.

These initial limitations set a foundation for a broader game economy: supply of in-game currency, cost of incubation, size of a battle reward.

Secondly, as every interaction with smart contracts requires a payment of a miners fee we have to limit the associated mental costs by reducing the number of actions that players have to make and make the code as efficient as possible.

Thirdly, as Ethereum blocks are mined every 15 seconds we have to limit the interactivity of the game mechanics to suit this model.

Additionally, we want to mention that the proposed random number generator (RNG) is in reality a pseudorandom number generator (PRNG) and true randomness is currently unachievable on the blockchain. Therefore, for high value transactions we offer a solution which minimizes the chances of exploiting the game's algorithms.

Lastly, as we rely on Metamask, the current usage is limited to desktop browsers, however it will be possible to move to mobile once the web platform is operational.

4 Obtaining Randomness

There are two main approaches used for random number generation. The first relies on some physical phenomenon which is expected to be random. The second relies on a computational algorithm. The latter uses some initial value to generate a random number. However, a generated random number can be reproduced if the initial value (or source of it) and the RNG algorithm is known to the attacker. Hence, this approach, known as pseudorandom, is not truly random.

For some industries (e.g. online gambling), obtaining randomness, or to be more precise pseudorandomness, is a complex issue as the random numbers which influence a game's outcome should be equally unpredictable for all parties.

Within the current online apps, there are multiple ways to obtain randomness, either independently or by relying on third parties. However, users of such apps rarely have the possibility to validate the process of obtaining these random numbers.

On the other hand, for decentralized apps (dapps), where an outcome relies upon the random number, this issue becomes increasingly important due to the limited availability of RNG methods, the price associated with the the use of current solutions and the time required for RNG.

To build a proper decentralized game, where an outcome is defined by a random number, the following specifications of RNG must be met [1]:

1) The RNG can generate a random number in a short period of time, ideally in less than 1 second.
2) The RNG can be trusted and verified by being stored on a blockchain.
3) The RNG should be capable of serving a large number of players simultaneously.
4) A low gas/transaction cost to obtain a random number.

Below we describe several different approaches to obtaining random numbers for Ethereum dapps available today.

4.1 Data from blockchain blocks

It is possible to obtain a random (pseudo-random) number from the block data directly, i.e. based on a block number, a block hash, etc. This approach is fast (the random number is generated in the next mined block) and relatively cheap, but there is a possibility that the miners can influence the output of the block in order to affect a generated number. More on this further down.

4.2 Use of oracles

This approach is based on some trusted external source of randomness, e.g. www.random.org, which is then delivered to the blockchain by other trusted external sources, e.g. http://www.oraclize.it [2].

This implementation is simple, but its weak spot is that oracles could also be compromised [3]. It also takes longer to obtain a random number as two blockchain transactions are needed. In our tests, it took around 30 seconds to get a random value using this approach.

4.3 Commit-reveal

RNG by this method requires two steps. During the first step, participants send hashes of random values and deposit a pledge. During the second step, the sent values are disclosed and a random number is generated based on these initial values [3]. The pledge is required to ensure the faithfulness of the participants. This method is used by RanDAO, Sleth, and Maker-Darts [3].

The disadvantages of this method are that a fee is required for most implementations in order to involve participants for random number generation, and that this method is subject to DDOS attacks. The latter results in a loss of pledges by honest participants [3].

4.4 Future blocks

During this RNG, a user sends a transaction that calls the RNG function. The sender's address is added to a list of requests by the RNG smart contract. In the next step, a user requests a random number via another transaction after some blocks have been mined by the network. The required action is taken using a generated random number, based on an initial input and some hashing algorithm, e.g. sha256.

This approach is slower than the other solutions (2 blocks will take about 30 seconds) and the current limitations of the EVM (Ethereum Virtual Machine) allow us to look back only 256 blocks. Hence, a user could just skip a generated number if it does not match his/her expectations.

4.5 Signidice algorithm

This approach of RNG is described in Gluk256 repository [3], [4]. The Ethereum smart contract receives a newly generated public key from the owner and awaits a value from the user. After receipt of this value, the owner hashes it with a private key, thus generating a required random number, which might be confirmed by an already published public key.

Regardless of the fact that a penalty for parties which do not reveal their signatures should be implemented, it is still possible to use this method for our use-cases. However, we consider this approach to be too complex for our initial requirements (more on this below).

4.6 Dragonereum's solution

According to the initial requirements, the RNG has to generate a random number as fast as possible (ideally it should take less than 1 second, however, the processing time is currently limited by Ethereum's rate of block creation). The best solution in this case is to use data from a blockchain block.

Our solution uses a block's hash to generate random numbers. As every transaction rely on different input data (dragon genetics, battle skills and tactics, etc), different dragons will receive different results even if several transactions are included in one block and therefore have same random number,

uint256(keccak256(abi.encodePacked(blockhash(block.number - 1), now)));

where *now* is the block.timestamp.

For complex computations where several different random numbers are required we use the remainder of the division of the initially generated random number and repeat the procedure until the necessary amount of random numbers is obtained.

4.7 Miner attack resistance analysis

The aforementioned algorithm is subject to miner attacks as the timestamp can be manipulated by a miner and several transactions can be included in the block in order to receive the *_seed* which matches the desired value.

The timestamp submitted by a miner has to comply with the following rules:

- The timestamp of the current block has to be greater than the timestamp of the parent block
- The timestamp can not be set too far into the future
- The timestamp should not be earlier than when the block was actually mined in order to keep the difficulty as low as possible.

Larger miners can safely manipulate the timestamp for some time without negative consequences. Therefore, we assume that a timestamp can be manipulated by a miner in the range of approximately 10 seconds (the timestamp in the Ethereum blockchain stored in

a second format), as constant higher deviation of the timestamp will result in a rejection of the block by other nodes.

Therefore, a malicious miner can have 10 times higher possibilities to manipulate the random number generation outcome. However, the use of the pure implementation of this RNG algorithm is limited in Dragonereum just by the battle transactions.

Low-value transactions

Battle transactions (regular battles only) allow players to earn Gold and XP for dragons participating in battles. We assume that these kind of transactions are low-value as each battle rewards users with a limited amount of in-game cryptocurrency and XP which does not have direct monetary value.

Battle transactions can not be compromised by regular players as the RNG includes block.timestamp which is set by a miner. Players can guess what the timestamp will be by looking into historical data and making some assumptions, however it will still be random as it can not be determined in advance with certainty.

However, this approach is easily exploitable by a sophisticated attacker who can employ a smart contract to check the value of block.timestamp during the block creation. We overcome this issue by allowing this function to be called only by an externally owned accounts [5] (onlyHuman modifier).

Miners on the other hand will have the possibility to alter the timestamp in order to benefit their own battle transactions. However, their possibilities for manipulation are limited by the factors outlined above. Additionally, to gain advantage by manipulating a timestamp the miner will have to do it over a long period of time, which will have a high chance of being noticed by the community. Thus, hurting the goodwill of the company behind it and potentially costing much more in lost profits since part of the miners might potentially leave the mining pool.

Additionally, regular players will be able to win the same amount of Gold (a maximum of 200 GOLD at the initial stage of the game, given the player has a dragon of DS close to the maximum) just by battling dragons of similar DS.

Also, recent developments evidentiated that miners do not easily tend towards manipulation techniques even when a much higher value is at stake [6]. Therefore, we assume that the proposed RNG is sufficient for these use cases.

High-value transactions

There are two types of transactions that we consider to be high-value. Those are:

- Breeding transactions
- Gladiator battle transactions

In the case of high-value transactions an attacker will have a much greater economic motivation to engage in a possible transaction manipulation. For both of these types of

transactions we offer certain tweaks to the original algorithm in order to make it more manipulation resistant.

With an original algorithm an attacking miner has a very high chance to clone the required genes (in an ideal situation it is 7/16 or 43.75% for an individual gene and 0.09% ($10 * (7/16 * 9/10)^{10}$) for the whole genome, please refer to Section 5.1). In order to prevent this, we supplemented our RNG algorithm with additional steps in the case of a breeding transaction.

Instead of "direct" breeding, in which case a miner will have an unlimited number of chances to initiate a breeding during a suitable moment, we introduce a Nest - an incubator with two eggs waiting to be hatched. Only one egg can be send to the Nest in a block and each forthcoming egg (i) opens the earliest placed egg in the Nest (i-2).

A regular player will have the same likelihood of predicting the timestamp that a miner sets for a block, however, now the miner only has a limited amount of time to find a suitable moment to implement the attack, as other miners might include the transaction of someone else sending an egg to the Nest, thus hatching the egg of the attacker.

In the case of gladiator battles - the feature which potentially might store the highest amount of value for the widest array of stakeholders, we offer a variation of the Future blocks approach (see Section 4.4). When a gladiator battle is set and an initial timer has run out any player can run the battle at any moment. In the case when the 256 block limit is exceeded, a new timer is set in the future and once it runs out, anyone can start the battle at any time.

This approach allows any interested party to conduct a battle, therefore withdrawing the possibility of waiting for an appropriate moment by those players who will not be happy with the outcome.

The proposed solutions are not ideal and have a degree of vulnerability, however at the current state of decentralized technology and given the use cases, those solutions offer a balance between the simplicity of implementation/usability and the resistance to an attack.

We hope that with the introduction of true randomness in a future development of the Ethereum ecosystem there will be a possibility to switch to more secure algorithms which at the same time will not be more difficult from the player's point of view.

5 Blockchain Genetics

5.1 Blockchain breeding algorithm

A major limitation of current non-blockchain games is the process of new character creation. Current solutions allow the user to choose some characteristics of a character or supply a default character with a preconfigured range of abilities. Additionally, all current solutions are centralized by design, so the player does not control or have the ability to verify the process of game character creation, the number of created characters or their set of skills.

As such, the game's creators can, at any moment, release newly created characters with an improved skill set or unique appearance, thus destroying all economic incentives of current players who invested time and money into their characters. This is particularly important for cases where all created characters are collectibles.

We propose a system whereby any third party (even the creators) has no control over new character creation after the release of the system.

To build such a system, all steps of the creation process should be done on blockchain. In order to do that, two main principles must be met:

1) A random number has to be generated on the blockchain or at least stored on the blockchain.

2) Character breeding also has to be done on the blockchain via a publicly accessible smart contract.

RNG was already described in this document and here we will dig into on-chain breeding in the context of its implementation on the EVM.

In Dragonereum, dragon genetics play a crucial role as they determine every aspect of a dragon's character, appearance and fighting abilities. Thus they define the market value of a particular dragon as a cryptocollectible item. As such we attempted to make the genetics of dragons similar to the genetics of real-world animals, nevertheless adding some additional features as we are talking about dragons, aren't we? Additionally, we have to keep the complexity at a level appropriate to EVM.

Also in order to minimize costs, we reduced the complexity of the smart contract to the level where users pay less, but the predictability of breeding is not sacrificed.

We hope that with future releases of new solutions and Ethereum protocols, the cost of breeding will be reduced and there will be a possibility to switch to more advanced forms of breeding algorithms.

In the following text we will describe the breeding algorithm.

Two dragons upgraded to a certain level can breed and produce an egg. This egg can be incubated[2] and a new dragon with the traits of both parents will be born.

[2]The egg is placed in the Nest. Once the third egg is placed to the Nest, the first egg placed there

As can be seen from Dragonereum's egg code (part of a ERC721 token) it stores just the information about parents so the traits of a dragon are determined from this information during the incubation.

```
struct Egg {
       uint256[2] parents;
       uint8 type; // type of a dragon (only for genesis eggs)
   }
```

A hatched dragon on the other hand has much more information stored inside its ERC721 code.

```
// health and mana are multiplied by 100
struct HealthAndMana {
       uint256 timestamp; // timestamp of last HP or mana update
       uint32 remainingHealth; // remaining from last update
       uint32 remainingMana; // remaining from last update
       uint32 maxHealth;
       uint32 maxMana;
}
struct Level {
       uint8 level; // current level of dragon
       uint8 experience; // XP current level
       uint16 points; // DNA points
}
struct Tactics {
       uint8 melee; // ranged/melee tactics in percentages
       uint8 attack; // defense/attack tactics in percentages
}
struct Battles {
       uint16 wins;
       uint16 defeats;
}
// multilpied by 100
struct Skills {
       uint32 attack;
       uint32 defense;
       uint32 stamina;
       uint32 speed;
       uint32 intelligence;
}
// types:
// 0 - water, 1 - fire, 2 - air, 3 - earth, 4 - cyber
struct Dragon {
       uint16 generation;
       uint256[4] genome;
       uint256[2] parents;
       uint8[11] types; // array of weights of dragon's types
       uint256 birth; // timestamp
}
Dragon[] public dragons;
// dragon id -> value
```

will hatch, so the addition of the third egg in the incubator opens the first egg. The process of hatching continues with the addition of every new egg, out of the eggs inside the Nest, the egg that was in there first hatches everytime a new egg is added.

```
mapping (uint256 => bytes32) public names;
mapping (uint256 => HealthAndMana) public healthAndMana;
mapping (uint256 => Tactics) public tactics;
mapping (uint256 => Battles) public battles;
mapping (uint256 => Skills) public skills;
mapping (uint256 => Level) public levels;
mapping (uint256 => uint32) public coolness; // Dragon Skillfulness Index multiplied by 100
// id -> type of skill (dragon type)
mapping (uint256 => uint8) public specialAttacks;
mapping (uint256 => uint8) public specialDefenses;
// classes:
// 0 - no skill
// 1 - attack multiplier
// 2 - defense multiplier
// 3 - stamina multiplier
// 4 - speed multiplier
// 5 - intelligence multiplier
// 6 - healing
// 7 - mana regeneration
// id -> class
mapping (uint256 => uint8) public peacefulSkills;
// classes:
// 1 - attack, 2 - defense, 3 - stamina, 4 - speed, 5 - intelligence
// id -> class -> effect
mapping (uint256 => mapping (uint8 => uint32)) public buffs;
```

The genome is stored in an array of 4 numbers of uint256 type, that equals to 128 bytes in total. As it is shown below this is a sufficient size from the diversity perspective.

Dragonereum's dragons have 10 body parts, each consisting of 4 sets of genes, therefore, an example of the whole genome of a dragon will look as follows:

0302841 0400941 0204180 0207320 0105800 **0102291** 0407630 0401391 **0201591** 0306510 0300351 0202210 **0005190** 0305350 0007531 0204551 **0305021** 0004421 0005011 0409360 **0202431** 0107660 0006810 0103420 0202250 **0002831** 0103821 0304430 **0407111** 0107070 0004201 0102570 **0405551** 0303861 0104861 0306670 0200110 **0105031** 0104531 0106750

The order of the body parts is: head, eyes, horns, body, wings, arms, legs, tail, spikes, and skin. The active gene describing the body part in each set of 4 is written in bold type. Each particular body part gives the dragons a boost for a set of *basic skills*. For example, the head is responsible for attack, stamina, and intelligence, while the eyes are responsible for defense, speed and intelligence (see Section 6.1).

The genome sequence determines the appearance and skills of a dragon. Also individual genes from it take part in the breeding process. In this example the first body part is represented by the following code:

0302841 0400941 0204180 0207320

Let's break down the first gene, which is the dominant one in this case, of the first set of genes and see how it influences the appearance of our theoretical dragon:

03 - dragon type (earth dragon)

02 - gene variety (copper dragon)
84 - gene level
1 - dominant

Similar to the natural world, every gene can be recessive (weak) or dominant (strong) and has an influence on the outcome of genome creation during breeding. Every new dragon has two sets of genes (one from each parent) according to the following rules:

- The dominant gene always suppresses the recessive gene.
- A pair of recessive genes will always be transferred to the next generation.

An active gene determines the appearance and skills of a dragon. Active genes are defined by another set of rules:

1) If a pair of genes consists of two dominant genes, the first dominant gene will become the active gene for this dragon.
2) If a pair of genes consists of a dominant and a recessive gene, the dominant gene will become the active gene.
3) If a pair of genes consists of two recessive genes, then the first of the pair will become the active gene.

Therefore, according to rule 1, 0302841 from the example above, is our active gene which determines the appearance of the head as both genes in the first pair are dominant. As can be seen from Figure 3 this dragon will have a Copper dragon's head.

Figure 3: A dragon with an Earth Copper head, whose genome is studied in the text.

Dragonereum's breeding algorithm is represented below.

```
function breeding(
       uint8[16][10] memory _momGenome, // parsed genome
       uint8[16][10] memory _dadGenome, // parsed genome
       uint8 _uglinessChance // if inbreeding
   ) internal returns (uint8[16][10] genome) {
       uint256 _seed = random.random(2**256 - 1);
       uint256 _random;
       uint8 _mutationChance = _uglinessChance == 0 ? MUTATION_CHANCE :
_uglinessChance;
       uint8 _geneType;
       for (uint8 i = 0; i < 10; i++) {
           (_random, _seed) = _getSpecialRandom(_seed, 4); // 0..9999
           genome[i] = _calculateGen(_momGenome[i], _dadGenome[i], (_random %
16).toUint8()); // get 1 random pair from 16 genes pairs
           (_random, _seed) = _getSpecialRandom(_seed, 1); // 0..9
           if (_random < _mutationChance) {
               _geneType = 0;
               if (_uglinessChance == 0) {
                   (_random, _seed) = _getSpecialRandom(_seed, 2); // 0..99
                   _geneType = (_random % 9).add(1).toUint8(); // 1..9
               }
               genome[i] = _mutateGene(genome[i], _geneType);
           }
       }
```

```
}
```

An example of a breeding can be found in our article explaining breeding, mutations and inbreeding effects on dragons [7].

This approach allows us to create a truly trusted world of cryptocollectible dragons, where everyone will be able to observe the breeding process, and no one will be able to influence the results, thus completely removing the necessity of trust as is the case with other solutions available today.

5.2 Dragon diversity

Five different dragon types will be released during Dragonereum's launch. Each type will inherit Regular genes according to the gene variety Table shown in Figure 4. There will be 10000 genesis eggs released during the Genesis. Inbreeding genes and genes of highest quality (Mystery Genes) will play no part in the genesis dragon creation process.

Since every dragon has ten different body parts, each dragon type has 282 475 249 (7^{10}) different dragon varieties. This is not including Inbreeding and Mystery Genes. Therefore, the chance of receiving two similar dragons during Genesis is 0.00070802663492828 7% (1 in 141 237). Consequently, we do not do additional checks for the existence of a particular dragon appearance as chances are very small but this check will require additional gas usage by the smart contract.

Five types of dragons will result in 2 758 547 353 515 625 different breeds when combined. Again, that is without inbreeding and Mystery genes. If all those dragons are to be evenly distributed among the population of the Earth, each person would receive around 360 000 varieties of dragons.

Together with inbreeding and Mystery Genes, there will be 97 656 250 000 000 000 different dragons originating from those initial five types. This is 12 800 000 for each person in the World.

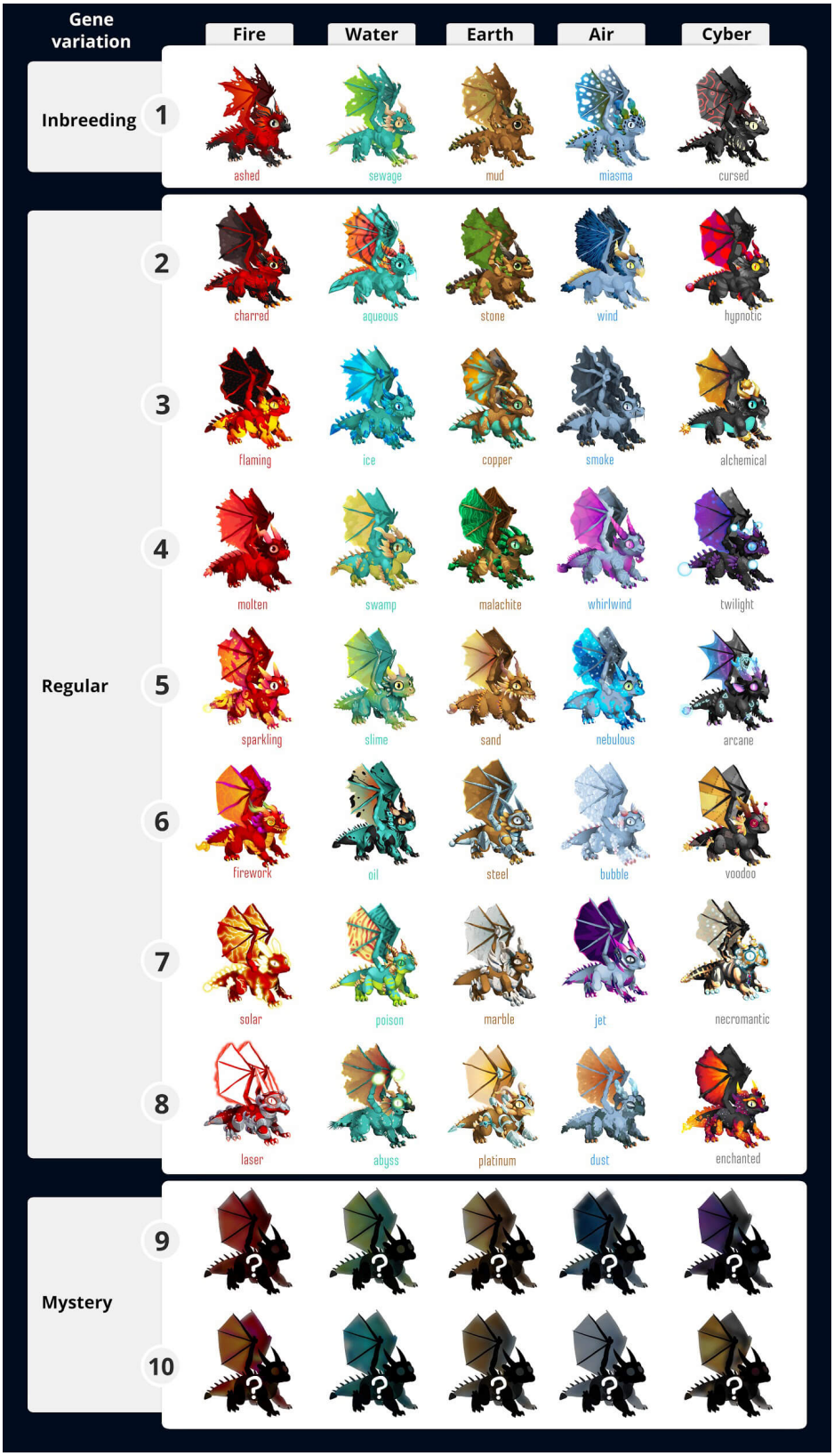

Figure 4: The different dragon types, with all of their possible variations, including inbreeding - row 1, regular rows 2-8, and mystery rows 9, and 10.

6 Game Mechanics

Our aim is to create a cryptocollectible game, whereby those users who spend more time playing the game have the possibility to own higher valued dragons and accumulate higher amounts of Gold.

In order to do this the game ecosystem should be balanced and should meet the following criteria:

- The growth of the dragon population should be limited by the game mechanics.
- Early game adopters should have an advantage in accumulating Gold over those players who joined the game at a later date, given that all other comparing factors are on an equal footing.
- There should not be one winning strategy that will allow someone to gain advantage over other players.

In this section we address these issues and explain the solutions that are implemented in order to comply with those requirements.

However, before we do this we have to get a deeper understanding of the game mechanics and underlying math. We have explained some game concepts in Section 2, and here we will dig into the details and formulas.

6.1 Body parts and basic skills

As it was discussed, there are five Basic Skills: *attack*, *defense*, *stamina*, *speed*, and *intelligence*. The basic skills are calculated based on the dragon's genetics and obtained experience (body parts level ups).

- *Attack* – affects the degree of damage a dragon can cause in a battle.
- *Defense* – measures the ability of a dragon to defend itself during a battle. If the dragon engages in a battle with a dragon which has less *attack*, *defense* determines the damage it can do to the weaker attacking dragon.
- *Stamina* – measures the maximum amount of HP and the speed of regeneration.
- *Speed* – defines which dragon moves first and each dragon's maximum reach.
- *Intelligence* – determines the maximum amount of Mana, along with *speed* for regeneration and the probability of *special attack* or *special defense* being used in battles.

The total skills of a dragon are calculated by summing up all body parts increased by Level Ups. So if you upgrade the wings of a dragon, the total skills of a dragon will also receive a boost.

As it was noted in Section 5.1 each dragon has ten body parts. Each body part is responsible for three particular basic skills and it does not contribute to the remaining two, see Table 1.

	Attack	Defense	Stamina	Speed	Intelligence
Head	✓	×	✓	×	✓
Eyes	×	✓	×	✓	✓
Horns	✓	✓	×	×	✓
Body	✓	✓	✓	×	×
Wings	×	×	✓	✓	✓
Arms	✓	×	✓	✓	×
Legs	×	✓	✓	✓	×
Tail	✓	×	✓	✓	×
Spike	✓	✓	×	×	✓
Pattern	×	✓	×	✓	✓

Table 1: The table shows which body parts are responsible for which particular skills.

6.2 Dragon types/families

A dragon might have a combination of body parts of different types and varieties depending on the genome of the particular dragon.

Each dragon type/family is better at a particular basic skill, as shown in Table 2. For example, fire dragons have more *attack* compared to other dragon families.

	Attack	Defense	Stamina	Speed	Intelligence
Fire	1.5	1	1	1	1
Water	1	1	1.5	1	1
Earth	1	1.5	1	1	1
Air	1	1	1	1.5	1
Cyber	1	1	1	1	1.5

Table 2: The dragon type factor for each basic skill.

Additionally, every dragon type has a unique advantage (in the form of higher damage) over other dragon types, similar to the "rock-paper-scissors" game, as shown in Figure 5. For example, water dragons will generally outperform fire dragons as water puts out fire.

Also, each dragon family has a particular *special attack, special defense* and *special peaceful skill.*

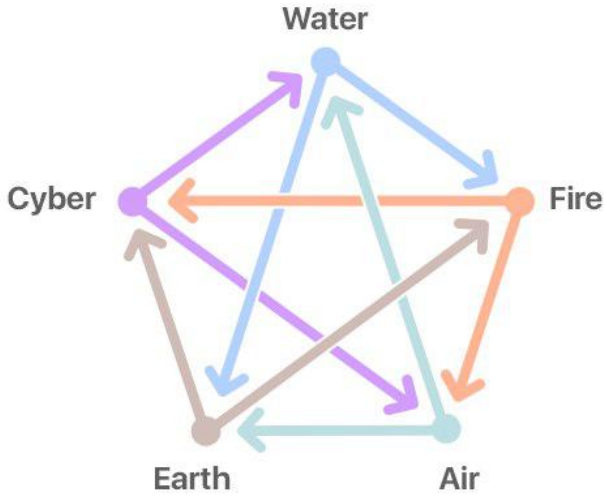

Figure 5: Each dragon family has an advantage over the two dragon families towards which the arrows are pointed: water over earth and fire, fire over air and cyber, air over earth and water, earth over cyber and fire, and cyber over water and air.

6.3 Dragon varieties

Each dragon family has genes of different quality. The differences between these varieties influence the dragon's appearance and strength of skills. Therefore, one water dragon might be stronger than another water dragon even without updates or leveling up.

The appearance was already displayed in Figure 4 of Section 5.2, and here we discuss the skills of the different varieties.

Table 3 below provides the dragon variety factors for the five basic skills depending on the different dragon varieties

In Table 3, the first row shows the dragon variety factor for inbreeding dragons, as such it is penalized by our genetics algorithm to be 0.5.

The genes of the highest quality (9–10 in Table 3 above) are Mystery Genes and will only be accessible as a result of mutations. Genesis dragons will not have these genes at all (more on it in Section 10).

The chance for a regular mutation to occur (excluding the case when two relatives interbreed) can be described as follows:

- During the incubation there is a 10% probability for each gene to mutate.
- The mutated gene might be of any of the varieties between 2 (regular) and 10 (mystery) listed in column 1 of Table 3 with an equal probability (the 1 (Inbreeding) variety is received only in the case when close relatives interbreed).

Details of inbreeding penalties and chances are described in our Medium

Variety	Codename	Attack	Defense	Stamina	Speed	Intelligence
1 (inbreeding)	Inbreeding	0.5	0.5	0.5	0.5	0.5
2 (regular)	Common 0	1.5	1.5	1	1	1
3 (regular)	Common 1	1	1.5	1.5	1	1
4 (regular)	Common 2	1	1	1.5	1.5	1
5 (regular)	Common 3	1	1	1	1.5	1.5
6 (regular)	Rare 0	2	2	2	1	1
7 (regular)	Rare 1	1	2	2	2	1
8 (regular)	Rare 2	1	1	2	2	2
9 (mystery)	Epic 0	4	4	4	1	1
10 (mystery)	Epic 1	1	1	4	4	4

Table 3: The dragon variety factors for each basic skills depending on the dragon gene variation.

articles (https://medium.com/@dragonereum).

If a player chooses to inbreed close relatives there is a high probability for a negative mutation will occur. This probability depends on different cases and is:

- Very high for inbreeding full siblings (80%)
- High for inbreeding half-siblings (70%)
- 50% for second and third cousins. The chances of negative mutations might increase if these dragons have several close relatives.

6.4 Calculating the overall basic skills of a dragon

The overall skills of a dragon will be calculated as the sum of skills for all its body parts:

$$Sk(j) = \sum_{i=1}^{10} bp^i(j) * f^i_{dt}(j) * f^i_{dv}(j) * L^i, \tag{1}$$

where $Sk(j)$ is one of the five *basic skills*, bp is the body part influence on the particular skill as shown in Table 1, a checkmark corresponds to $bp = 1$ and a cross corresponds to $bp = 0$, f_{dt} is the dragon type factor shown in Table 2, f_{dv} is the dragon variety factor in Table 3 and L is the body part level[3]. The index j denotes the basic skill, $j \in$ *attack, defense, stamina, speed, intelligence* , and the summation index i in the superscript, denotes the body part, $i \in$ head, eyes, horns, body, wings, arms, legs, tail,

[3]The body part level is the level of the active gene for each body part.

spike, pattern. The following notation for the basic skill will also be used through the text: Sk(attack)≡A, Sk(defense)≡D, Sk(stamina)≡S, Sk(speed)≡V, Sk(intelligence)≡I.

This can be illustrated by the following example in Table 4. The numbers in the last column will be added to the other body parts and the result will show the dragon's aptitude for in the different basic skills.

Skill	Body part	Dragon type	Gene variation	Level of a body part	Total
	Leg	Fire	Inbreeding	3	
Attack	0	1.5	0.5	3	0
Defense	1	1	0.5	3	1.5
Stamina	1	1	0.5	3	1.5
Speed	1	1	0.5	3	1.5
Intelligence	0	1	0.5	3	0

Table 4: An example, illustrating the calculation of the basic skill for a specific body part - the Ashed Legs at level 3 of a particular dragon.

The maximum available health and mana points are calculated in Equation 2 and 3.

$$H = 5 \times S, \tag{2}$$

$$M = 5 \times I, \tag{3}$$

The current levels of health and mana points available to a dragon might be lower than the maximum calculated in Equation 2 and 3, as a result of a battle, in which the Mana could have been used to activate the *special attack* or *special defense* skills, while the HP could be lost as a result of damage done by an opponent. Mana and HP restore automatically over time.

6.5 Dragon Skillfulness Index

In order to represent the dragon skillset in an easier way we offer an aggregative parameter called the Dragon Skillfulness Index. The Dragon Skillfulness Index will be displayed next to the dragon's name, Figure 6.

The Dragon Skillfulness Index is calculated as follows:

$$I_{ds} = \sum_{k=1}^{40} P_g(k), \tag{4}$$

Where, I_{ds} is the Dragon Skillfulness Index, the gene points, P_g are calculated in Equation 5 below, and the index k runs from 1 to 40 for each gene in the dragon's genome as shown in Section 5.1 .

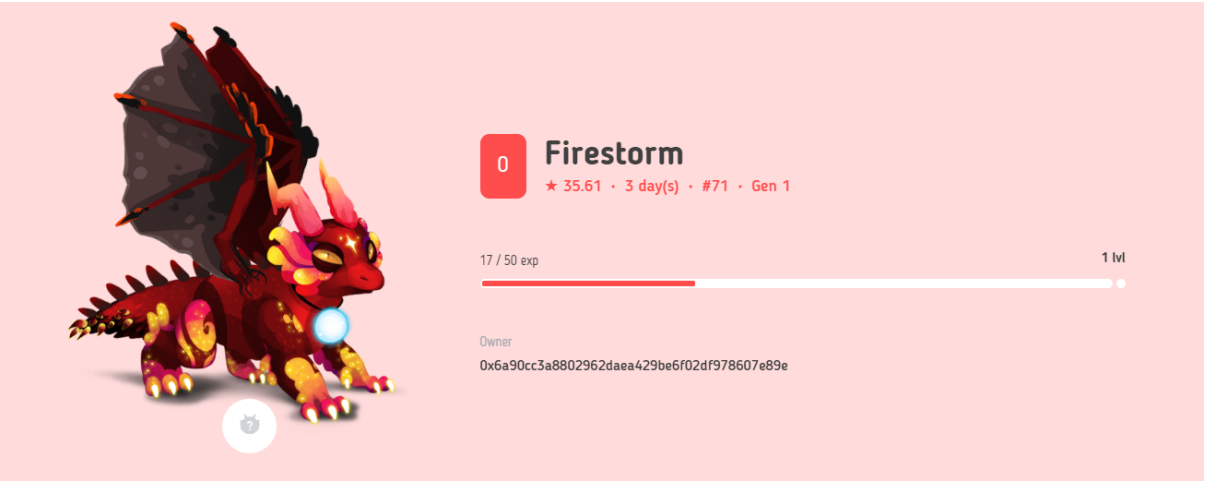

Figure 6: The dragon display, showing the image of the dragon, the dragon's name and the Dragon Skillfulness Index under the name.

$$P_g = L_g * f_{gv} * f_{dr}, \tag{5}$$

Where L_g is the gene level, f_{gv} is the gene variety factor, and f_{dr} is the dominant or recessive factor, for a dominant gene $f_{dr} = 1$ and for a recessive gene $f_{dr} = 0.7$, described in Section 5.1. The gene variety factor for each dragon variety is given in Table 5.

Gene Variety	Codename	Gene variety factor
1 (inbreeding)	Inbreeding	0.5
2 (regular)	Common 0	1.2
3 (regular)	Common 1	1.2
4 (regular)	Common 2	1.2
5 (regular)	Common 3	1.2
6 (regular)	Rare 0	1.6
7 (regular)	Rare 1	1.6
8 (regular)	Rare 2	1.6
9 (mystery)	Epic 0	2.8
10 (mystery)	Epic 1	2.8

Table 5: The gene variation multiplier depending on the different dragon variations

6.6 Special attack and special defense skills

Special attack or *special defense* is the ability to use the special powers available to each dragon family. It improves the *attack* or *defense* during a battle move. Those special

skills use Mana. In Table 6 the amount of Mana necessary for the activation of the *special attack* or the *special defense* skill depending on the type of dragon is shown as well as the probability for its activation.

For dragons with a mixed genome as a result of the breeding process the type used for special skills is determined during the incubation process by a factored random and stored in the type field in the specialAttacks/specialDefenses mappings (see Section 5.1).

Dragon type	Fire	Water	Earth	Air	Cyber
Special Attack skills details					
Mana points	$3 \times A$	$3 \times S$	$3 \times D$	$3 \times V$	$3 \times I$
Attack factor	$\sqrt{A/3} + 1$	$\sqrt{S/3} + 1$	$\sqrt{D/3} + 1$	$\sqrt{V/3} + 1$	$\sqrt{I/3} + 1$
Probability of activation	$\sqrt{I} + 10$	$\sqrt{I} + 10$	$\sqrt{I} + 10$	$\sqrt{I} + 10$	$\sqrt{I} + 10$
Special Defense skills details					
Mana points	$3 \times A$	$3 \times S$	$3 \times D$	$3 \times V$	$3 \times I$
Defense factor	$\sqrt{A/3} + 1$	$\sqrt{S/3} + 1$	$\sqrt{D/3} + 1$	$\sqrt{V/3} + 1$	$\sqrt{I/3} + 1$
Probability of activation	$\sqrt{I} + 10$	$\sqrt{I} + 10$	$\sqrt{I} + 10$	$\sqrt{I} + 10$	$\sqrt{I} + 10$

Table 6: Table of Mana costs, multipliers for a attack/defense and a probability of activation for Special Attack and Special Defense.

6.7 Special peaceful skills

The maximum Level Up for a dragon is 10. On the 10th Level Up, the player can obtain a *special peaceful skills* for their dragon.

A *special peaceful skill* allows users to earn in-game currency in exchange for offering skills to the game community on the game's marketplace[4].

A dragon can buy a *special peaceful skill* that can be used in order to receive a boost to one of its *basic skills as* shown in the first five rows of Table 7. After the boost has been applied, the *basic skill* defined in Equation 1 will become $Sk^{new}(j) = f_j \times Sk^{old}(j)$, with f_j being the Action Factor and j denoting the basic skill, $j \in$ A (attack), D (defense), S (stamina), V (speed), I (intelligence).This boost will be available to use for only one battle, thus incentivizing players to find an opponent and commence a battle as soon as they have bought the *special peaceful skill* and applied the boost to one of their *basic skills*. Alternatively, a player can buy a *special peaceful skill*, which will recharge the Health or Mana points of the dragon, bottom two rows of Table 7.

The dragon offering the *special peaceful skill* will have to use its Mana points to be able to offer the service. The amount of Mana points necessary for each different *special peaceful*

[4]The price of the *special peaceful skills*, alongside the prices of other items, such as dragons and eggs, traded on the marketplace will be decided by the market forces. The role of Dragonereum will only be to provide the internal marketplace to facilitate trade, however players will be able to interact directly with smart contracts and third-party marketplaces can be introduced.

skill is shown in column 2 of Table 7. The selling dragon will receive the in-game currency reward for the service it provided, and its Mana points will restore themselves over time.

Special peaceful skill	Action Factor	Mana points
Attack boost	$f_A = \sqrt{A/30} + 1$	$3 \times A$
Stamina boost	$f_S = \sqrt{S/30} + 1$	$3 \times S$
Defense boost	$f_D = \sqrt{D/30} + 1$	$3 \times D$
Speed boost	$f_V = \sqrt{V/30} + 1$	$3 \times V$
Intelligence boost	$f_I = \sqrt{I/30} + 1$	$3 \times I$
Healing	$H_R = 2 \times S$	$3 \times S$
Mana recharge	$M_R = 2 \times I$	$3 \times I$

Table 7: Table showing the different special skills that can be obtained (column 1), the Mana points the selling dragon needs to offer each skill (column 2) and the factor showing the increase in skill (column3).

Special peaceful skills can be used in both regular and gladiator battles. In the case of gladiator battles a dragon will be freezed once the battle is initiated and no *special peaceful skill* or body part level ups can be applied.

6.8 Leveling up dragons

A dragon will be able to breed only if it has accumulated the required number of XP for a Dragon Level Up and as a result has acquired the necessary amount of DP.

At this point, a user will have to decide whether to upgrade their dragon (level up its body parts and get a stronger dragon) or to interbreed it with another dragon.

Additionally, getting dragon offsprings will become more and more difficult when going up levels as the required amount of XP will increase from level to level. Therefore, it will be necessary to have, and to win more and more battles.

Before we describe the growing difficulty in acquiring DP we have to look at how the DS is calculated,

$$DS = 0.7 * A + 2.1 * D + 2.3 * S + 1.1 * V + 1.5 * I, \tag{6}$$

where A is the *attack*, D is the *defense*, S is the *stamina*, V is the *speed*, and I the *intelligence* of the dragon. The *basic skills* are calculated in Equation 1[5].

[5]In the equation $Sk(attack) \equiv A$, $Sk(defense) \equiv D$, $Sk(stamina) \equiv S$, $Sk(speed) \equiv V$, $Sk(intelligence) \equiv I$

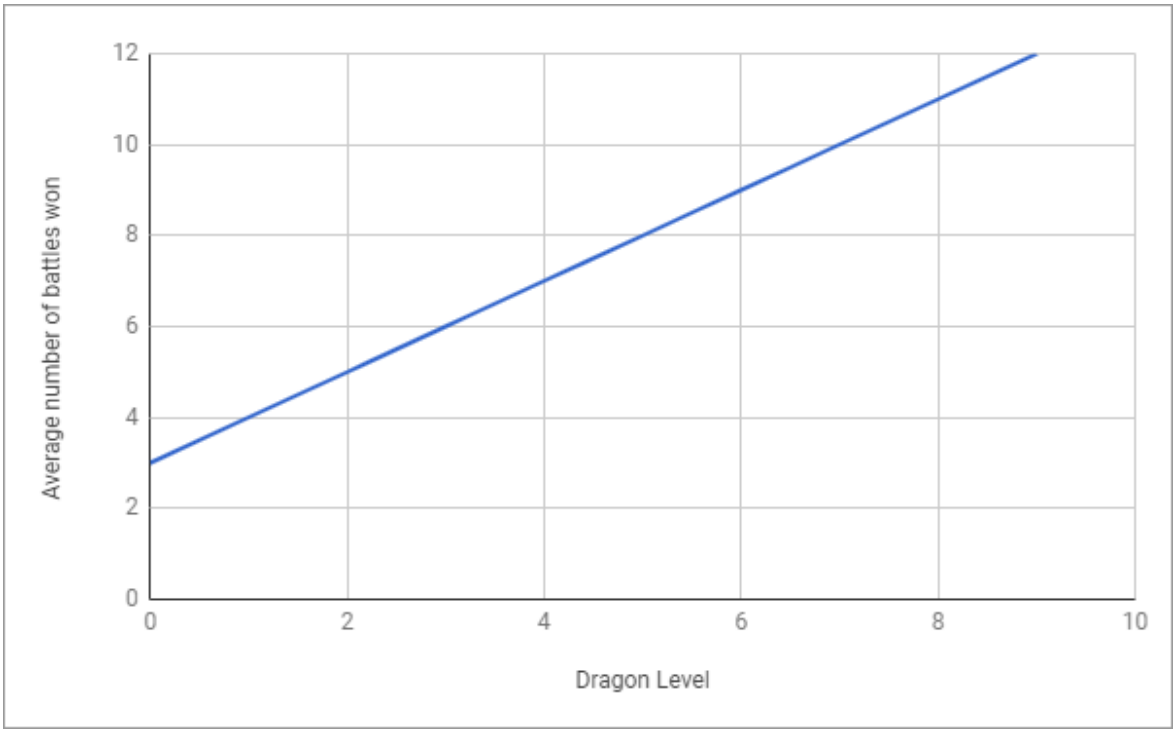

Figure 7: Average number of battles won required for a dragon Level Up

The constants in Equation 6 were carefully obtained through a rigorous simulation procedure ensuring that dragons with similar DS have an equal chance at winning the battle. Numerous battles between randomly chosen dragons with a similar amount of total basic skills were simulated and if there were battles where a dragon had an advantage, the values were adjusted in order to avoid this situation. The process was repeated until there were no dragons that had a winning advantage and every dragon, regardless of their skills or traits, has a 50% chance of winning against a dragon with a similar DS.

Based on the DS, the game's algorithms offer suitable opponents for regular battles. A short description and the corresponding code for the regular battles are given in the Appendix.

As a result of these battles the winners will receive XP. The number of XP won, in the case when the attacking dragon is the winner, is calculated according to the formulas below.

$$XP_e = \begin{cases} \left(\frac{DS_d}{DS_w}\right)^5 & \text{if } DS_d < DS_w \\ \left(\frac{DS_d}{DS_w}\right)^2 & \text{if } DS_d \geq DS_w, \end{cases} \tag{7}$$

where XP_e are the Experience Points earned after a battle, DS_d is the DS of the defeated dragon, and DS_w is the DS of the winning dragon. In the case when the attacking dragon loses, and the defending dragon wins, XP won is calculated according to the formulas below, however it is limited by the algorithm to be not more than 10.

$$XP_e = \begin{cases} \left(\frac{DS_d}{DS_w}\right)^5 /2 & \text{if } DS_d < DS_w \\ \left(\frac{DS_d}{DS_w}\right) /2 & \text{if } DS_d \geq DS_w, \end{cases} \tag{8}$$

The reduction in the number of XP won is done in order to incentivize the attacking dragons, which use gas to initiate the transaction on Ethereum.

Once the required amount of XP is accumulated the dragon will get a dragon level up. There are ten Dragon Level Ups possible for each dragon. Each next dragon level up requires more XP and therefore more battles to be won, see Table 8.

At the same time the strongest dragon in the population will likely get less XP as a result of battles as it will become more and more difficult to find an opponent of the same or higher DS, thereby limiting the number of battles it can have on the one hand and minimizing the amount of XP it can obtain by winning battles on the other.

Dragon Level	Experience Points required for a level up
0	30
1	40
2	50
3	60
4	70
5	80
6	90
7	100
8	110
9	120
10	-

Table 8: The amount of XP required for a level up at each level (0-10).

Once a dragon goes up a level it receives DNA Points which are allocated according to Table 9 below.

DP might be spent on:

- body parts level up
- breeding

Body part level ups will increase the total *basic skills* as is shown in Equation 1, since every body part level up gives a boost to the corresponding skills it is responsible for (see

Dragon Level	DNA Points received after leveling up
0	0
1	10
2	13
3	16
4	21
5	28
6	37
7	48
8	62
9	81
10	106

Table 9: The amount of DP received after leveling up at each level (0-10).

Table 1)[6]. The example of how this is calculated is given in Table 4 in Section 6.4.

The amount of DP required for a body part level up is calculated as follows:

$$C_{DP}(L_n) = Round(1.02 \times C_{DP}(L_{n-1})), \tag{9}$$

where $C_{DP}(L_n)$ is the cost of the body part level up at level L_n and $C_{DP}(L_{n-1})$ is the cost of the body part level up at level L_{n-1} correspondingly. The expression on the right hand side of the equation is rounded to the nearest hundredth, and this value is then used for the calculation of the cost at the next level and so on (the actual DP points are further rounded to the nearest integer). The first term in the series is $C_{DP}(L_0) = 10$, so 10 dP are required to level up a body part to level 1. The DP points required for leveling up a body part, calculated in Equation 9 are presented in Figure 8 as a function of the body part level.

Here we will look at the amount of DNA Points required for breeding at each level, see Table 10.

As can be noted, the amount of DP received (see Table 9) and the amount of DP required for breeding (see Table 10) are the same at each level. Therefore, a dragon has a choice between two options, it can either breed or upgrade its skills. The system will favor the spending of DP on breeding while on the same level where those DP were received, as once a dragon moves to the next level this amount of DP will only be sufficient for a body part level up.

[6]When the body part level up increases the active gene's level for that body part increases as well, since they are identical, however the levels of the other three genes in the body part set remain the same. From this arises the possibility that a dragon's offspring will inherit one of these three genes as its active gene and will have lower initial basic skills than its parents.

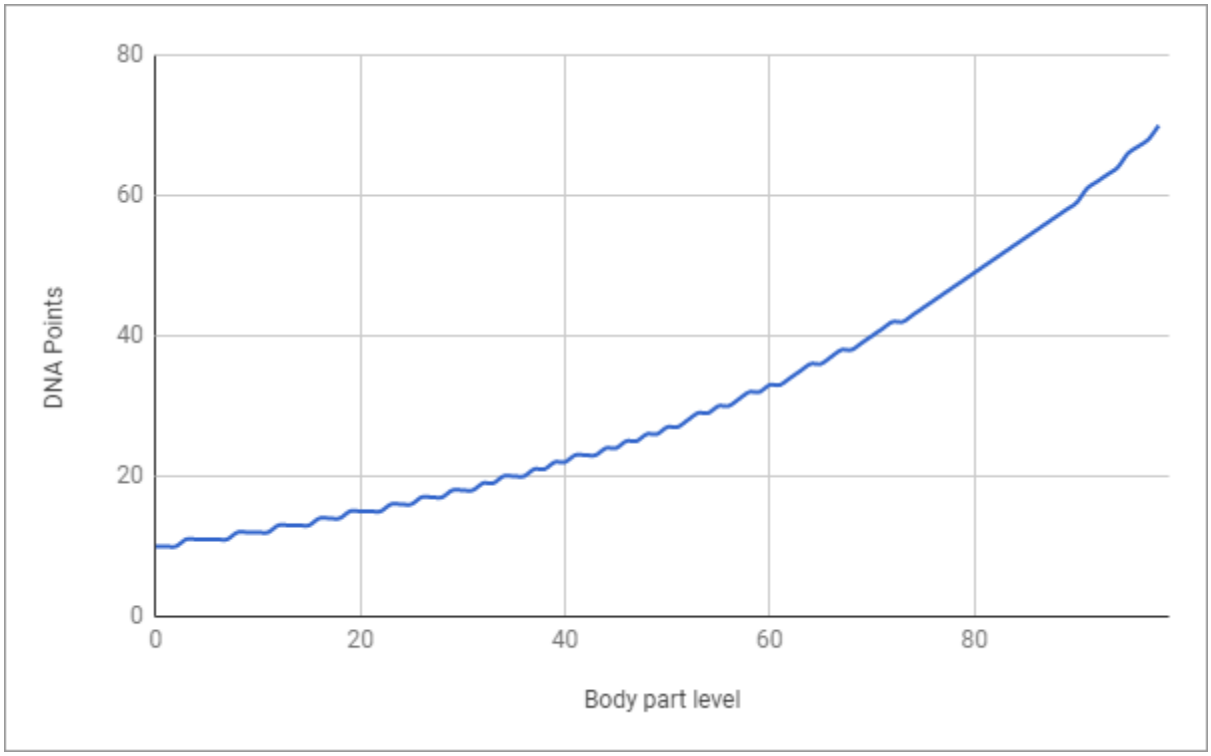

Figure 8: The amount of DNA Points necessary for a body part level up at each level (0-99)

Dragon Level	DNA Points required for breeding
0	-
1	10
2	13
3	16
4	21
5	28
6	37
7	48
8	62
9	81
10	106

Table 10: The amount of DP required for breeding at each level (0-10).

6.9 Early adopter advantage

As it was discussed in Section 2 the game has the Treasury, where all unallocated Gold is held. It is used to reward battle winners and leaderboard entrants. The *battle winner reward* is calculated using Equation 10 below,

$$R = \left(\frac{G}{N_d^2}\right) * I_{ds} * 10 * f_{DS}, \tag{10}$$

*w*here R is the *battle winner reward*, G is the Gold balance in the Treasury, I_{ds} was defined in Equation 4 above, and f_{DS} is the *dragon strength factor* defined as,

$$f_{DS} = \begin{cases} \left(\frac{DS_d}{DS_w}\right)^8 & \text{if } DS_d < DS_w \\ \\ \frac{DS_d}{DS_w} & \text{if } DS_d > DS_w. \end{cases} \tag{11}$$

However, R, is subject to the following rules:

- If there are less than 15000 dragons in the population the maximum *battle winner reward* is 200 Gold coins;
- If there is more than 15000 but less than 30000 dragons in the population, the maximum *battle winner reward* is 100 Gold coins;
- Else (more than 30000 dragons) the maximum *battle winner reward* is 25 Gold coins.

Additionally, due to the early adopter advantage, it will be easier for those players that start participating in the game from its launch to earn Gold by entering the leaderboard.

6.10 Dragon population growth

Now that we explained how dragon skills are calculated and how dragons level up, we can talk about dragon population growth.

Our simulation shows that if the dragon population will reach 100k of dragons, and even if just small part of players will play the game, it is highly unlikely that Ethereum Foundation network will be able to process those transactions even taking into account potential block gas limit growth according to the Moore's Law in the following years.

Therefore, there are several embedded mechanisms to ensure that dragon population will be within limits set by the game design (as described in Section 3).

In addition to difficulty growth while going up levels (Section 6.8) another game mechanism prevents the situation in which dragons become very valuable regardless of their level and skills and every player decides to breed their dragon once it receives the first dragon level up. It works as follows: as dragon population will grow, the number of

battles held will increase. Therefore, the amount of Gold distributed to battle winners will decrease making it more and more difficult to obtain the required amount for an egg incubation (please see Section 6.9).

On the other hand, dragons with higher DS will be able to receive Gold by being in the top of the leaderboards and by winning gladiator battles, thus creating incentive for other players to shift their playing style to produce higher valued dragons.

Additionally, once the 10000th dragon will be hatched, another game mechanism will come into play: half of Gold paid for each new incubation will be permanently destroyed by the Nest.

Given that the cost of incubation is constant and equals 1000 GOLD, 500 GOLD will be burned each time new egg is sent to the Nest, thus increasing relative price of every next incubation. This will limit theoretical supply of dragons at the level of 129999 dragons (with the remainder of 500 GOLD in the Treasury). Nonetheless we think that this theoretical cap will not likely be reached.

7 Gameplay Simulations

The following simulations were conducted, using the initial parameters shown in Table 11[7].

Gas price	5 Gwei
Egg incubation cost	1000 GOLD

Table 11: Initial parameters used in the gameplay simulations.

We study in detail three different cases. It should be noted that not all possible game features have been included in the simulations. For instance, a reward can be distributed to the top dragons on the leaderboard, and this was not incorporated as part of the cases studied below. De facto, these cases should be considered to be simplified situations, however they can be used as a very good gauge of the possible real life scenarios.

7.1 Case 1: Predominantly breeding

Number of Battles	Population	Treasury (GOLD)	Supply (GOLD)	Median battle reward (GOLD)	Median cost of Gold (ETH)	Cost of egg incubation (ETH)
500000	19100	27690000	40435000	29.92	0.000150	0.150
1000000	26800	25650000	36590000	14.32	0.000314	0.314
1500000	31100	24160000	34440000	10.05	0.000448	0.448
2000000	33900	22760000	33060000	8.01	0.000561	0.561
2500000	35900	21410000	32020000	6.70	0.000671	0.671
3000000	37600	20120000	31220000	5.76	0.000780	0.780

Table 12: Case Study 1: the number of battles held (column 1), the corresponding dragon population (column 2), the Gold in the Treasury (column 3), Gold supply (column 4), the median *battle winner reward* (column 5), the median cost of Gold in ETH (column 6), and the cost of an egg incubation in ETH (column 7).

In this scenario players breed dragons with a 90% probability and level them up with a 10% probability. The game balance decreases to around 20M GOLD at 3,000,000 battles and continue to decline slowly as a result of the high volume of egg incubations (part of Gold is burned during hatching), see Table 12 (column 3), and Figure 9 (black line). In Figure 9 the dragon population growth is shown as a function of the number of battles held. The number of battles in these case studies is a more appropriate variable than using a synthetic time, as it is problematic to predict the actual number of active players.

[7]We are using values, which were typical for September of 2018.

The initial sharp growth of the dragon population is due to the distribution of genesis eggs.

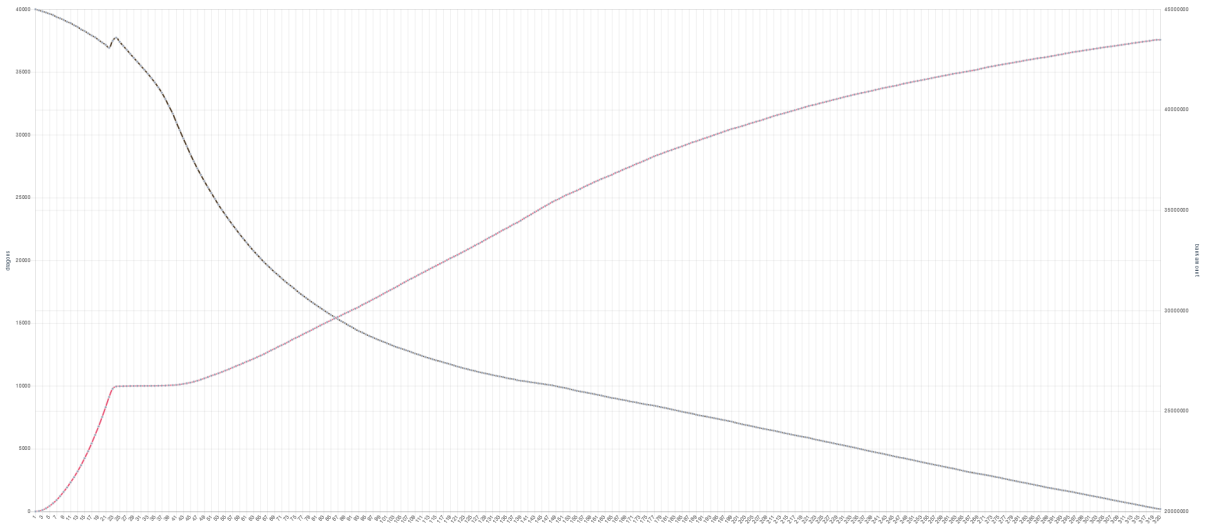

Figure 9: The number of dragons (red line), and the Treasury balance in GOLD (black line) as a function of the number of battles held.

The *battle winner reward* settles at 4.94 GOLD to 7.6 GOLD per battle depending on the DS. This small range of reward values can be explained by the fact that all dragons are of similar DS since the players do not level up dragons and spend all DP on breeding. The exact values of the median *battle winner reward* can be seen in Table 12 (column 4), and Figure 10 (dark blue line).

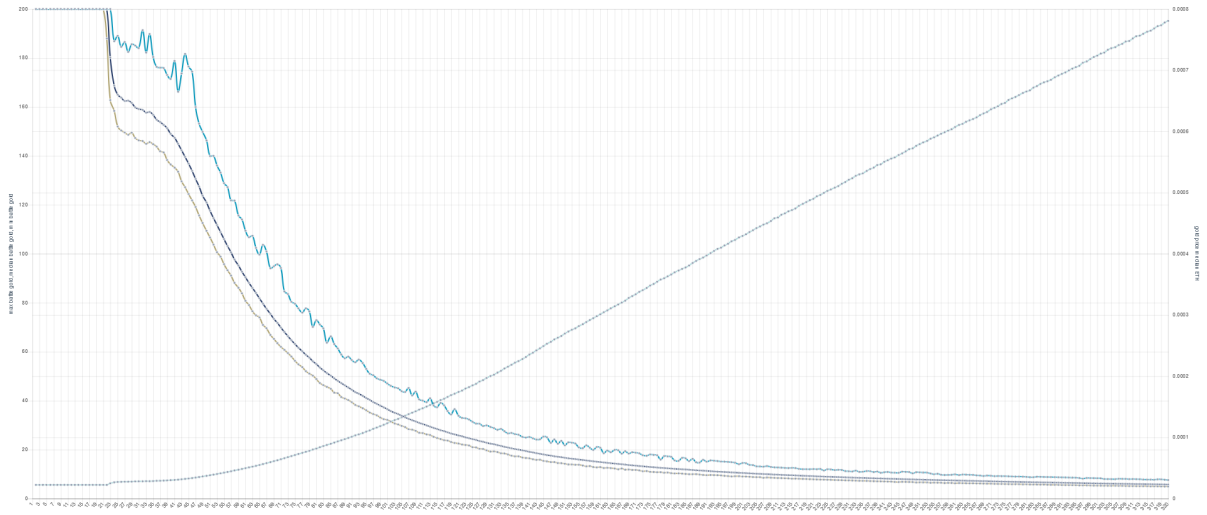

Figure 10: The cost of Gold in ETH calculated as a sum of mining fees (light blue line), and the median *battle winner reward* in Gold versus the number of battles held (dark blue line). The minimum (brown line) and the maximum (cyan) *battle winner reward* are also shown encasing the median *battle winner reward*.

In Figure 10 the cost of Gold in ETH calculated as a sum of mining fees necessary to mine the corresponding transactions is shown (light blue line) versus the number of battles held, which just like in Figure 9 is a variable that represents time in this synthetic environment. The cost of Gold grows because the amount of Gold distributed to winners reduces with the increasing number of battles held (see Section 6.9), making Gold more valuable.The median *battle winner reward* shown in the dark blue line decreases, as it depends on the amount of Gold in the Treasury, see Equation 10, which also decreases as shown in Figure 9.

The mining cost for 1 GOLD is between 0.000591 ETH and 0.000909 ETH, at 3,000,000 battles, Figure 10 (light blue line). It is calculated as: $\frac{C}{\bar{R}}$, where C is the cost of the transaction and $\bar{R}$ is the median *battle winner reward* (the median battle reward is shown as a function of the number of battles held in Figure 10).

The cost of an egg incubation in this case is between 0.59 ETH and 0.91 ETH. This range comes from the fact that depending on who the winning dragons are there will be a different *battle winner reward* associated with them depending on their DS.

Therefore, economically motivated players will start leveling up dragons in order to get dragons of higher DS and earn more Gold in battles. As such, Case 1 is not in a stable state and will eventually turn into Case 2 portrayed below.

7.2 Case 2: Breeding and Leveling Up

Number of Battles	Population	Treasury (GOLD)	Supply (GOLD)	Median battle reward (GOLD)	Median cost of Gold (ETH)	Cost of egg incubation (ETH)
500000	12500	18010000	43750000	51.18	0.000088	0.088
1000000	16000	10110000	42000000	20.50	0.000219	0.219
1500000	19600	6980000	40200000	10.72	0.000419	0.419
2000000	22900	5550000	38500000	6.70	0.000672	0.672
2500000	26400	4970000	36800000	4.75	0.000948	0.948
3000000	29300	4620000	35330000	3.69	0.001217	1.217

Table 13: Case Study 2: the number of battles held (column 1), the corresponding dragon population (column 2), the Gold in the Treasury (column 3), Gold supply (column 4), the median *battle winner reward* (column 5), the median cost of Gold in ETH (column 6), and the cost of an egg incubation in ETH (column 7).

In this scenario players randomly breed and level up dragons. In our opinion this is the most likely scenario.

The game balance drops to around 10M GOLD at 1,000,000 battles, however further decline is very slow and the ballance decreases to 4.6M GOLD at 3,000,000 battles. The *battle winner reward* settles between 1.99 GOLD and 10.21 GOLD per battle depending

on the DS of the winning dragon.

In Figure 11, just as in Figure 9 the initial sharp growth of the dragon population is due to the distribution of genesis eggs (red line). The black line shows the amount of Gold in the Treasury, which is declining due to the distribution of Gold to the battle winners and partial annihilation of Gold by the Nest.

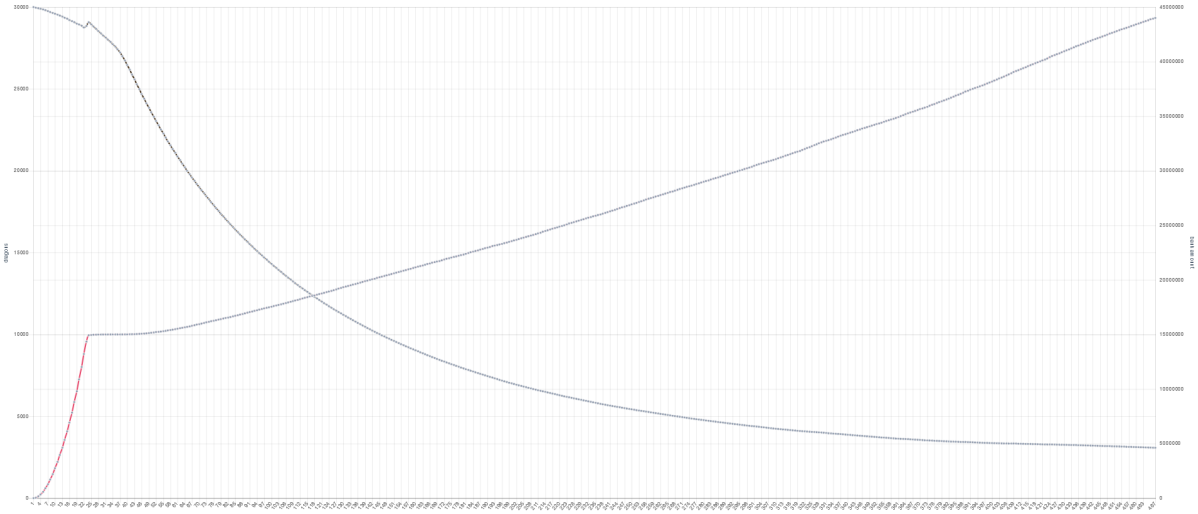

Figure 11: The number of dragons (red line), and the Treasury balance in GOLD (black line) as a function of the number of battles held for Case Study 2.

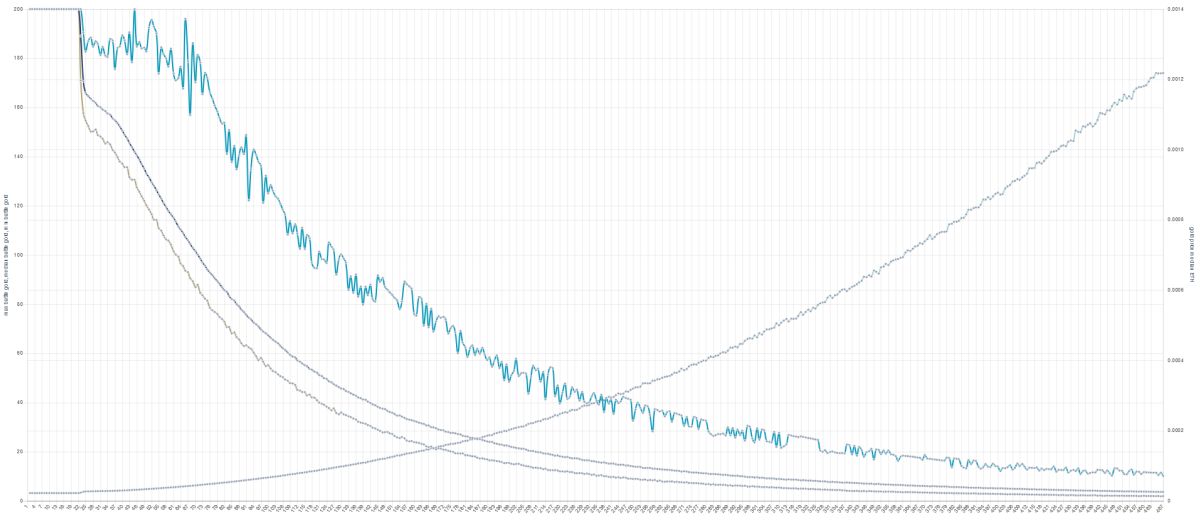

Figure 12: The cost of Gold in ETH calculated as a sum of mining fees (light blue line), and the median *battle winner reward* in Gold versus the number of battles held (dark blue line) for Case Study 2. The minimum (brown line) and the maximum (cyan) *battle winner reward* are also shown encasing the median *battle winner reward*.

The mining cost for 1 GOLD is between 0.000441 ETH and 0.002257 ETH, at 3,000,000

battles, see Table 13 (column 5), and Figure 12 (light blue line). The value of the mining cost will continue to grow with the increasing number of dragons and the *battle winner reward* will be reduced as was described in Section 6.9. The egg incubation cost in this case is between 0.44 ETH and 2.26 ETH. This variation is due to the fact that each dragon has a corresponding DS, on which the *battle winner reward* depends (see Equation 10), so the *battle winner reward* will depend on which dragons are fighting at a given time, and the possible price of the egg incubation will have a spread depending on all the different possibilities. It should also be noted that in Figure 12, the maximum *battle winner reward* fluctuates due to the fact that it depends on the DS of the participating dragons. Therefore, if certain dragons with high DS are not participating in battles at some points this will affect the value of the maximum *battle winner reward*.

7.3 Case 3: Predominantly Leveling Up

Number of Battles	Population	Treasury (GOLD)	Supply (GOLD)	Median battle reward (GOLD)	Median cost of Gold (ETH)	Cost of egg incubation (ETH)
500000	10000	14010000	45000000	66.69	0.000067	0.067
1000000	10190	2497000	44900000	16.86	0.000267	0.267
1500000	11250	0	44370000	0	Determined by the market	Determined by the market

Table 14: Case Study 3: the number of battles held (column 1), the corresponding dragon population (column 2), the Gold in the Treasury (column 3), Gold supply (column 4), the median *battle winner reward* (column 5), the median cost of Gold in ETH (column 6), and the cost of an egg incubation in ETH (column 7).

In this scenario players level up dragons on all levels with a 90% probability and breed with a 10% probability.

The game balance drops to 0 GOLD at approximately 1,430,000 battles, as a result of the low amount of fees collected for egg incubations, as is seen in Figure 13, and in column 3 of Table 14. Therefore the *battle winner reward* will not be paid. This is shown in Figure 14, where the *battle winner reward* drops sharply to zero with the number of battles held (dark blue line). Just before that moment the median *battle winner reward* will be 0.13 GOLD, and the median mining cost of Gold will be 0.022573 ETH. Once the game balance is at 0 GOLD, the price of Gold cannot be predicted due to the fact that when there is no Gold in the Treasury, the model assumes that the price of Gold is equal to the transaction fees price divided by zero (or another very small amount), thus making it too big to display on the graph.

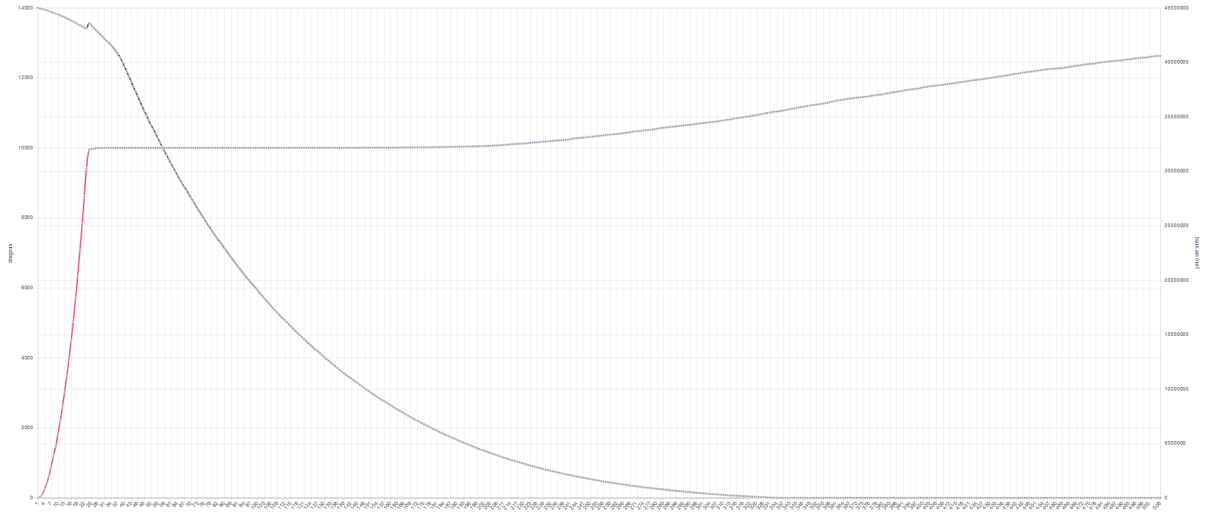

Figure 13: The number of dragons (red line), and the Treasury balance in GOLD (black line) as a function of the number of battles held for Case Study 3.

Gold can be obtained from other players and therefore its price can not be determined by this simulation. The egg incubation cost also goes up and becomes unfeasible for the majority of players.

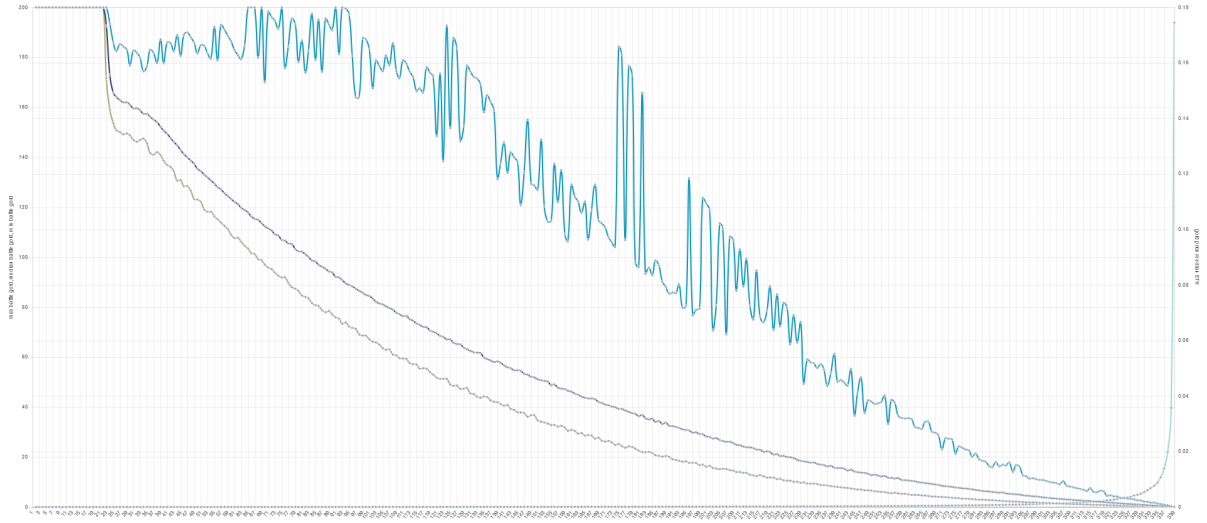

Figure 14: The cost of Gold in ETH calculated as a sum of mining fees (light blue line), and the median *battle winner reward* in Gold versus the number of battles held (dark blue line) for Case Study 3. The minimum (brown line) and the maximum (cyan) *battle winner reward* are also shown encasing the median *battle winner reward*.

Therefore, the game slows down and new dragons can only be hatched with the Gold bought on the marketplace. However, this is not critical as the initial *battle winner reward* is not a goal of the game but just a mechanism to start up the in-game economy.

Our aim is the creation of the game ecosystem where users will offer different goods and services to each other and those goods and services will be exchanged for Gold (healing, buffs) or acquired with the help of Gold (bets on dragon battles).

Additionally, as the price of dragons will become higher more players will switch to breeding dragons therefore bringing the game to the state portrayed in Case 2.

These synthetic tests confirm that the aim to create a game ecosystem which is stable in the long term was achieved. The explored scenarios might not include all the possible options for the players' behaviour, nonetheless, they offer an insight on how the game ecosystem might develop after the launch. It is shown that due to the system of feedback loops implemented in the game, that all three cases are going to end up in the equilibrium state of Case Study 2.

8 Implementation Details

8.1 Serverless application

Modern centralized games heavily rely on the entities that have created those games and that have full control over every aspect of the game: what to consider as a "bad" or "good" behaviour, private data policies, controls over the servers and all the data stored there, and determining the overall direction of the game development [8], [9], [10][8], [11], [12].

At the same time a number of technologies unrelated to the gaming industry has been emerging which let users enjoy a transparent, censorship resistant, trustless, programmable and permissionless environment where anyone has full control over their private data (this technology stack is often referred to as Web 3.0 [13], [14].)

We think that it would be great to create a full scale game built on top of those technologies, so the created game will be 100% autonomous and decentralized, and no party will have control over any element of the game or its delivery mechanisms once it is launched, while still remaining engaging and fun compared to its centralized siblings.

However, with Dragonereum our aim is more realistic and it is to create a decentralized game which will have minimal ties to our servers at launch and gradually move to decentralized solutions (storage, naming services and indexing) once the game is released.

At game's launch we will retain the possibility to update and pause smart contracts in case of a critical bug or game balance issues. However, once the smart contracts will be battle-tested and all necessary security measures will be taken, the control over deployed smart contracts will be voluntarily transferred to a new owner. This might be a DAO or just the 0x000..0001 address.

The upgrade mechanism works as follows:

- A new version of a required smart contract is deployed.
- Ownership of this contract is transfered to UpgradeController.
- Migrate function with old and new smart contracts is called by our multisig address.

In the long run and with the advancement of technology, we hope to remove any user dependency on us or on any other third party and switch to a fully autonomous and decentralized mode.

The game is currently accessible via browser with a MetaMask plugin installed (https://metamask.io/).

No sign up or any kind of registration is required as in order to offer engaging gameplay no one needs to know the names or any other personal information of the players. The

[8]Sort the table by organization type (3rd column) and see the gaming entries.

private keys which are stored in the MetaMask wallet are sufficient to serve as the players' IDs.

The data layer consists of two ERC721 [15] non-fungible tokens which will represent two states of a dragon (an egg and a dragon) and an ERC20 [16] which will represent the in-game cryptocurrency.

The business logic layer consists of several smart contracts: egg core features contract, dragon core features contract, breeding, battling and marketplace contract.

The game itself can be used without reliance on a back-end server for any purposes other than allowing users to derive the app from the server to the player's browser. The app uses client-side rendering, therefore, our server is actually just a storage which can be easily replaced.

However, to make the life of players a bit easier another back-end server is used for:

- Filtering of items offered on marketplaces.
- Opponent matchmaking for regular battles.
- Storage and delivery of notifications while a player is not online.

While the filtering server stores complete information about every item offered on the game's marketplaces, the client side application only receives the list of items in a required order. This feature can easily be turned off in the app settings so the app can be used without it.

Full scale decentralization is our long term goal and once the game has all the core functionality we hope that new solutions and services from a Web 3.0 stack will be implemented.

8.2 Off-chain solutions

In Section 3 we discussed the current limitations associated with decentralized technologies and explained the steps taken in building Dragonereum with these limitations in mind. Another possible way of resolving the effects of the transaction costs, the associated delays with transaction processing and network congestion issues on the Ethereum network is with the use of off-chain (state-chain) solutions. We studied in detail the following most promising off-chain approaches for resolving the scalability issue:

- State channels [17]
- Sharding [18]
- Plasma [19]
- TrueBit [20], [21]

- zk-SNARKS off-chain validation(ZoKrates) [22]
- Sidechains [23]
- POA network [24]

After carefully investigating the above off-chain solutions, we have concluded that at present they do not offer a viable solution for our game design, as many of them are not production ready yet and those in production do not have wide user base. As our game design was built with the current limitations of Ethereum's blockchain bandwidth in mind, at this stage there is no real necessity to go off-chain.

Furthermore, by staying on-chain we have the advantage of remaining more secure, while the above third-party solutions might have certain vulnerabilities associated with them, especially if they have not been released and tested out yet. Another benefit from not relying on third parties is the fact that our system will not become dependent on solutions that might be suspended or abandoned at any point in time and, as a result, dragons will be immortal and will live forever.

8.3 Graphics

While designing the game we considered several options for visualisation style of our dragons. Current status quo in cryptocollectible industry is use of flat, simplified images which are easily merged into an image of a game character.

However, we decided to switch to 3/4 view as it gives more realistic perspective to players and it makes the potential transition to VR or AR a bit easier.

9 Possible Integrations

9.1 Trading protocols

In addition to the in-house marketplaces for game tokens such as dragon eggs (ERC721), dragons (ERC721) and Gold (ERC20) the decentralized nature of the tokens will allow them to be traded on other global trading platforms.

We prefer this to happen in an open and decentralized manner as all the information about each token is publicly accessible and there is no need to rely on any third party. Therefore, the best option for this is the decentralized liquidity networks and exchanges.

Joining the decentralized liquidity network Bancor will provide instant liquidity for Gold and enable its convertibility to other connected tokens and vice versa with low fees and no trading spread.

Additionally, all game tokens are compliant with the 0x's protocol V2 therefore all of them can be traded on 0x relayers when V2 of 0x's protocol will be launched.

The compliance of Dragonereum's ERC721 tokens with the Wyvern Protocol (https://www.projectwyvern.com/) will enable gas-free and escrow-free auctions (sell orders) on platforms such as Opensea.io.

At game's launch there will be basic solution for in-game ERC20 trading, however we will consider implementing advanced solutions at a later date.

9.2 Prediction markets

There are many game events which can be used as a base for a prediction, for example, battle winners and leaderboard entrants. Since in Dragonereum everything is kept on-chain, third-party services such as Augur (https://www.augur.net/) can be used to place bets on events and outcomes stored in the game's smart contracts (even those that we can not predict at the moment).

With the current implementation of the Augur ecosystem we do not see the possibility to bet on small events (such as the outcome of a regular battle), however, other betting services (DiceyBit https://diceybit.com) already offer those capabilities with the use of data supplied by the game developers through an API. On the other hand, Augur can be used for larger scale events, such as predicting the maximum Dragon Skillfulness Index at a future date.

9.3 Virtual worlds

Bringing Dragonereum's dragons into the virtual reality of Decentraland (https://decentraland.org) will make the gaming experience even more engaging and immersive. For example, players will be able to witness dragon battles and find hidden dragon eggs.

Additionally, connecting Dragonereum's marketplaces to Decentraland's economy and community will allow for a higher engagement on the game's marketplaces.

At the first stage Dragonereum will have its own world in Decentraland (land in Genesis City) which will serve as a basis for future VR capabilities and features.

9.4 Messenger integrations

Easy access to the Ethereum blockchain will make the world a more open and transparent place. Therefore we advocate the use of the game in messenger platforms (e.g. Status) which will allow interaction with the game through text based contracts, interfaces and bots.

9.5 Location based protocols

Use of location based protocols will bring new dimensions into the Dragonereum's gameplay. Though core game design will much likely stay unchanged, it will be possible create new location based game mechanics in order to utilize the power of proof-of-location protocols such as control by a dragon or a thunder of dragons over countries or territories.

10 Genesis

An initial distribution of genesis eggs will be conducted during the game's launch. 10 000 genesis eggs will be distributed in total (2000 of each egg type). Types of eggs (water, fire, air, earth, and cyber) will be distributed on a sequential basis. Genesis eggs store pure types of dragons without additions from other types.

The Genesis eggs will be distributed in the following way:

- First 5000 Genesis eggs will be available for a claim once per block.
- Next 1250 Genesis eggs will be available for a claim once every 2 blocks.
- Next 1250 Genesis eggs will be available for a claim once every 4 blocks.
- Next 1250 Genesis eggs will be available for a claim once every 8 blocks.
- Next 1250 Genesis eggs will be available for a claim once every 16 blocks.

The eggs will be free to claim, however the amount of gas used by genesis transactions will be artificially increased in order to make genesis eggs and dragons more valuable.

As Gold is required for egg incubation, genesis eggs will be distributed together with the required amount of Gold. Genesis eggs do not store any information about parents, therefore, a player will not be able to predict the value and rarity of a dragon which will be hatched from a particular genesis egg.

The distribution of body parts during genesis is shown in Table 15.

Body part	Percentage
inbreeding	0
common 0	0.30
common 1	0.24
common 2	0.22
common 3	0.19
rare 0	0.025
rare 1	0.015
rare 2	0.01
epic 0	0
epic 1	0

Table 15: The percentage distribution of the different body parts according to their variety.

11 GOLD

There will be 60,000,000 GOLD minted during product deployment and no new coins will be added. Gold will be allocated as shown in Figure 15.

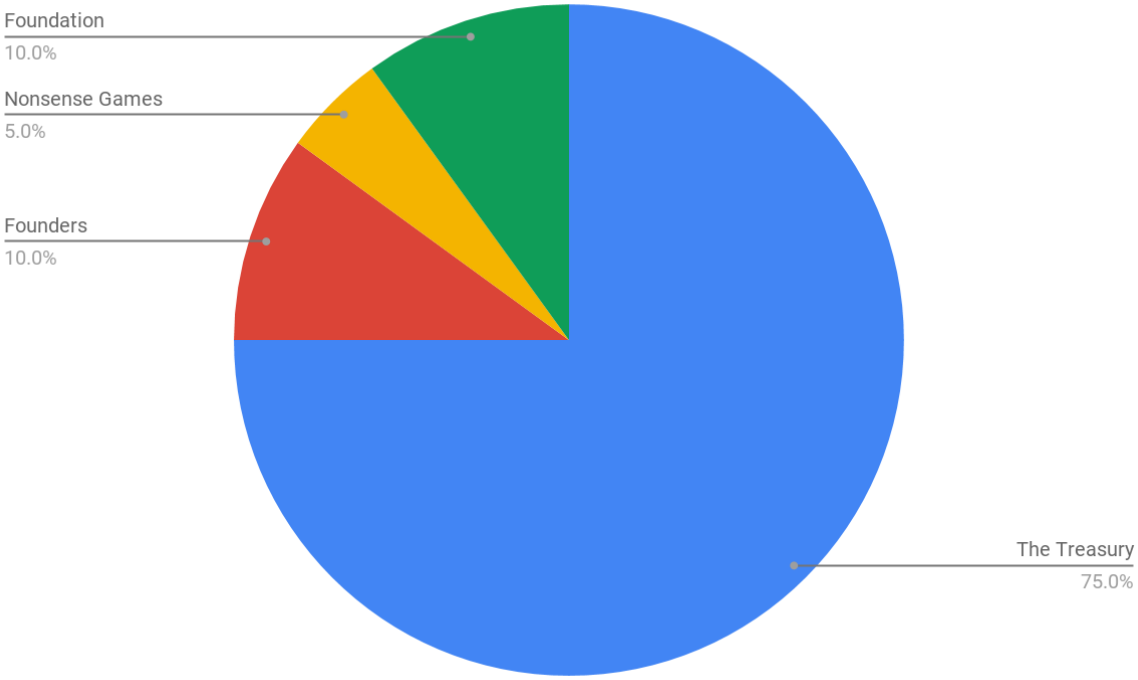

Figure 15: Chart showing the allocation of GOLD.

The Treasury

As Dragonereum will not have any kind of initial coin offerings or a sale, the majority (75%) of tokens will be allocated to the Treasury (shown in blue in Figure 15).

Cybernomics of the game is based on the willingness of players to evolve better dragons: as dragons with higher DS are likely to earn more Gold.

The more dragons are battling, the more expensive it becomes to earn Gold from the Treasury (please see 6.9). The Treasury is guaranteed to be replenished with players' Gold payments for incubation through embedded genetics mechanism. This basic feedback loop stimulates GOLD internal liquidity.

Cybernomics also determines the population specifications: dragons are very cheap to generate at the Genesis, thus stimulating aggressive population growth in order to become dominant species in the Ethereum universe.

During early stages of the game there is an incentive for a faster population growth, as available supply of Gold inflates faster than its deflation occurs (originating from the environmental costs of incubation).

Similar to the real world, evergrowing population is strictly limited by available resources.

Once environmental resources become scarcer deflation starts dominating over inflation thus securing the store of value for players who have managed to earn more Gold.

Founders

Once the smart contract is deployed, appointed team members will receive a total of 10% of tokens in their possession (shown in red in Figure 15) in accordance with individual contribution to the project in a form of funding, development, design, drawings, etc conducted before Genesis.

Nonsense Games

Nonsense Games will receive another 5% (shown in yellow in Figure 15) as a reward for the contribution of the team to the project before Genesis. The multisig address of Nonsense Games is 0x10208fb4ef202bdc49803995b0a8ca185383bba4.

The Foundation

The remaining 10% will be used for the Dragonereum ecosystem development and community building (shown in green in Figure 15). This may include (but not necessarily):

- New team members and advisors
- Liquidity pool (eg. Bancor)
- Airdrop
- Security audits
- Bounty campaigns (eg. Gitcoin).

Overall purpose of the Foundation is to stimulate liquidity of Gold by increasing its consumption by basic protocol contracts and auxiliary contracts which return Gold to the Treasury. Examples of such contracts can be name registries, trading protocols, bridges to other networks, etc.

The multisig address of the Foundation is 0x5ff8957ef7e964e8072815211c9fc3e7f820f1d4.

12 Conclusion

In this document we described a blockchain cryptocollectible PvP game built on the Ethereum network which emphasizes the advantages of the blockchain solutions for future blockchain gaming, gambling and collectible markets.

The technological and socioeconomic solutions for creation of a sustainable game ecosystem were examined and discussed.

A thorough description of the game's details and rules was provided. Further, we showed that Dragonereum is made with the current constraints of the blockchain technology in mind.

A description of the in-game cryptocurrency, Gold, was given and it was shown that its value will grow.

In addition, we presented a large variety of possible future integrations of the game.

References

[1] Tomas Draksas. *Dev. Update #6: Edgeless Dice RNG*. 2017, Aug 28. URL: https://medium.com/edgeless/dev-update-6-edgeless-dice-rng-daef755f26a0.

[2] Oraclize. *random-datasource*. 2017, May 18. URL: https://github.com/oraclize/ethereum-examples/tree/master/solidity/random-datasource.

[3] Ksenya Serova Dao Casino. *Dao Casimo White Paper*. 2017, Jun 27. URL: https://github.com/DaoCasino/Whitepaper/blob/master/DAO.Casino%20WP.md.

[4] Gluk256. *The Signidice Algorithm*. Accessed: 2018, Aug 26. URL: https://github.com/gluk256/misc/blob/master/rng4ethereum/signidice.md.

[5] Ethereum community. *Account Types, Gas, and Transactions*. Accessed: 2018, Nov 26. URL: http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html#account-types-gas-and-transactions.

[6] SECBIT. *How the winner got Fomo3D prize – A Detailed Explanation*. 2018, Aug 22. URL: https://medium.com/coinmonks/how-the-winner-got-fomo3d-prize-a-detailed-explanation-b30a69b7813f.

[7] Dragonereum. *How to breed Dragons on the blockchain*. 2018, March 26. URL: https://medium.com/@dragonereum/how-to-breed-dragons-on-the-blockchain-e7b4c8cad2c0.

[8] Gogo. *PLAYERUNKNOWN'S BATTLEGROUNDS*. 2018, Jun 16. URL: https://steamcommunity.com/app/578080/discussions/1/2788173147744268307/.

[9] Steam. *Banned by Game Developer (Game Ban)*. 2017. URL: https://support.steampowered.com/kb_article.php?ref=6899-IOSK-9514.

[10] WikipediA. *List of data breaches*. 2018. URL: https://en.wikipedia.org/wiki/List_of_data_breaches.

[11] Ben Gilbert. *Major changes are coming to 'Fortnite' and the most popular player in the world isn't happy about it*. 2018, Jun 22. URL: https://www.businessinsider.com/fortnite-changes-building-resource-ninja-2018-6.

[12] Aaron Mamiit. *Blizzard Secretly Made 'World Of Warcraft' Enemies More Powerful, And Players Are Not Happy About It*. 2017, Mar 30. URL: https://www.techtimes.com/articles/203499/20170330/blizzard-secretly-made-world-of-warcraft-enemies-more-powerful-and-players-are-not-happy-about-it.htm.

[13] Josh Stark. *Making Sense of Web 3*. 2018, Jun 6. URL: https://medium.com/l4-media/making-sense-of-web-3-c1a9e74dcae.

[14] Stephan Tual. *Web 3.0 Revisited - Part One: "Across Chains and Across Protocols"*. 2017, May 26. URL: https://blog.stephantual.com/web-3-0-revisited-part-one-across-chains-and-across-protocols-4282b01054c5.

[15] Ethereum. *ERC: Non-fungible Token Standard #721*. 2017, Sep 26. URL: https://github.com/ethereum/eips/issues/721.

[16] Ethereum. *ERC: Token standard #20*. 2015, Nov 19. URL: https://github.com/ethereum/eips/issues/20.

[17] Joseph Poon et al. *The Bitcoin Lightning Network: Scalable On-Chain Instant Payments*. 2016, Jan 14. URL: https://lightning.network/lightning-network-paper.pdf.

[18] Vitalik Buterin. *Ethereum 2.0 Mauve Paper*. 2016. URL: https://cdn.hackaday.io/files/10879465447136/Mauve%20Paper%20Vitalik.pdf.

[19] Josh Stark. *Making Sense of Ethereum's Layer 2 Scaling Solutions: State Channels, Plasma, and Truebit*. 2018, February 12. URL: https://medium.com/l4-media/making-sense-of-ethereums-layer-2-scaling-solutions-state-channels-plasma-and-truebit-22cb40dcc2f4.

[20] Jason Teutsch et al. *A scalable veri
cation solution for blockchains*. 2017, Nov 16. URL: https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf.

[21] Julia Koch et al. *A Predictable Incentive Mechanism for TrueBit*. 2018, Jul 2. URL: https://arxiv.org/pdf/1806.11476.pdf.

[22] Jacob Eberhardt. *ZoKrates*. 2018. URL: https://github.com/JacobEberhardt/ZoKrates.

[23] Adam Back et al. *Enabling Blockchain Innovations with Pegged Sidechains*. 2014, Oct 22. URL: https://blockstream.com/sidechains.pdf.

[24] Stephen Arsenault. *POA Network Whitepaper*. 2018, Jan 7. URL: https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper.

Appendix

Regular Battles

Below is the code describing the execution of the regular battles.

```
   function _resetBlocking(Dragon dragon) internal pure returns (Dragon) {
        dragon.blocking = false;
        dragon.specialBlocking = false;
        return dragon;
    }

    function _attack(uint8 turnId, bool isMelee, Dragon attacker, Dragon opponent, uint8
_random) internal pure returns (Dragon, Dragon) {
        uint8 _turnModificator = 10; // multiplied by 10
        if (turnId > 30) {
            // _turnModificator = uint8(1 * 10 + uint16((turnId - 30) * 10 * 5) / 40);
            // hack "stack too deep" error
            uint256 _modif = uint256(turnId).sub(30);
            _modif = _modif.mul(50);
            _modif = _modif.div(40);
            _modif = _modif.add(10);
            _turnModificator = _modif.toUint8();
        }

        bool isSpecial = _random < _multiplyByFloatNumber(attacker.specialAttackChance,
_turnModificator);
        uint32 damage = _multiplyByFloatNumber(attacker.attack, _turnModificator);
        if (isSpecial && attacker.mana >= attacker.specialAttackCost) {
            attacker.mana = attacker.mana.sub(attacker.specialAttackCost);
            damage = _multiplyByFloatNumber(damage, attacker.specialAttackFactor);
        }
        if (!isMelee) {
            damage = _multiplyByFloatNumber(damage, DISTANCE_ATTACK_WEAK__);
        }
        uint32 defense = opponent.defense;
        if (opponent.blocking) {
            defense = _multiplyByFloatNumber(defense, DEFENSE_SUCCESS_MULTIPLY__);
            if (opponent.specialBlocking) {
                defense = _multiplyByFloatNumber(defense, opponent.specialDefenseFactor);
            }
        } else {
            defense = _multiplyByFloatNumber(defense, DEFENSE_FAIL_MULTIPLY__);
        }
        if (damage > defense) {
            opponent.health = _safeSub(opponent.health, damage.sub(defense));
        } else if (isMelee) {
            attacker.health = _safeSub(attacker.health, defense.sub(damage));
        }

        return (attacker, opponent);
    }

    function _defense(Dragon attacker, uint256 initialSeed, uint256 currentSeed) internal
pure returns (Dragon, uint256) {
        uint8 specialRandom;
```

```
        (specialRandom, currentSeed) = _getRandomNumber(initialSeed, currentSeed);
        bool isSpecial = specialRandom < attacker.specialDefenseChance;
        if (isSpecial && attacker.mana >= attacker.specialDefenseCost) {
            attacker.mana = attacker.mana.sub(attacker.specialDefenseCost);
            attacker.specialBlocking = true;
        }
        attacker.blocking = true;
        return (attacker, currentSeed);
    }

    /* solium-disable-next-line security/no-assign-params */
    function _turn(
        uint8 turnId,
        uint256 initialSeed,
        uint256 currentSeed,
        uint32 distance,
        Dragon currentDragon,
        Dragon currentEnemy
    ) internal view returns (
        Dragon winner,
        Dragon looser
    ) {
        uint8 rand;

        (rand, currentSeed) = _getRandomNumber(initialSeed, currentSeed);
        bool isAttack = rand < currentDragon.attackChance;

        if (isAttack) {
            (rand, currentSeed) = _getRandomNumber(initialSeed, currentSeed);
            bool isMelee = rand < currentDragon.meleeChance;

            if (isMelee && distance > MAX_MELEE_ATTACK_DISTANCE) {
                distance = _safeSub(distance, currentDragon.speed);
            } else if (!isMelee && distance < MIN_RANGE_ATTACK_DISTANCE) {
                distance = distance.add(_multiplyByFloatNumber(currentDragon.speed, FALLBACK_SPEED_FA
            } else {
                (rand, currentSeed) = _getRandomNumber(initialSeed, currentSeed);
                (currentDragon, currentEnemy) = _attack(turnId, isMelee, currentDragon,
currentEnemy, rand);
            }
        } else {
            (currentDragon, currentSeed) = _defense(currentDragon, initialSeed, currentSeed);
        }

        currentEnemy = _resetBlocking(currentEnemy);
        if (currentDragon.health == 0) {
            return (currentEnemy, currentDragon);
        } else if (currentEnemy.health == 0) {
            return (currentDragon, currentEnemy);
        } else if (turnId < MAX_TURNS) {
            return _turn(turnId.add(1), initialSeed, currentSeed, distance, currentEnemy,
currentDragon);
        } else {
            uint32 _dragonMaxHealth;
            uint32 _enemyMaxHealth;
            (_dragonMaxHealth, ) = getter.getDragonMaxHealthAndMana(currentDragon.id);
            (_enemyMaxHealth, ) = getter.getDragonMaxHealthAndMana(currentEnemy.id);
            if (_calcPercentage(currentDragon.health, _dragonMaxHealth) >=
```

```
_calcPercentage(currentEnemy.health, _enemyMaxHealth)) {
                return (currentDragon, currentEnemy);
            } else {
                return (currentEnemy, currentDragon);
            }
        }
    }
```