Etherbots:
Strategic Battling and Reward-Based Economies in Decentralised Multiplayer Games

Alex Connolly

February 15, 2018

Contents

1 Introduction

This whitepaper is a draft: for questions, queries, and concerns please email alex@etherbots.io or contact the Etherbots team on discord.

CryptoKitties, released in November 2017, demonstrated that blockchain games based on the collection, trading and ownership of virtual objects are commercially viable and are able to attain significant popularity. However, such games have so far been unable to present sophisticated gameplay mechanics, or effectively incorporate direct player-to-player interaction, apart from marketplace functionality. To some degree, this is the result of the present inability of the Ethereum network to scale beyond limited numbers of transactions, which necessarily causes games to become more and more expensive to play as they increase in popularity. This mechanic favours games which require the fewest number of transactions, but as these issues are gradually resolved or circumvented, future blockchain games may be able to implement progressively more complicated gaming logic.

Nevertheless, these simpler games are necessary to prove the viability of this new ecosystem, and as new games or extended functionality can be built on top of existing game states, these collectibles may retain value after such changes take place (provided users remain willing to ascribe value to them). Further, player-to-player applications provide a valuable proving ground for concepts which will be utilised by broader peer-to-peer systems. The creation of artificial, logically constrained economies on the blockchain (such as those maintained by games) is a valuable trial of the future effectiveness of these platforms.

Etherbots presents a new vision of these 'collectible' games - one where the value of a token is not only defined by its visual attractiveness or its ability to appreciate in price, but by its relative utility with regard to interactions between users (the battle system). This paper describes the rationale for the design of Etherbots and outlines its first implemented battling system.

2 Rationale

2.1 Collectibility

The average transaction on the Ethereum network currently costs around $0.70. At the height of Cryptokitties' popularity, transaction fees surged, leading to a backlog of tens of thousands of unconfirmed transactions which

had been sent without accounting for this increase. The fact that all transactions are costly forces games, particularly those which require large volumes of transactions between large numbers of users, to create financial value for players as compensation for both their initial investment and ongoing costs. This is also the product of the player base - many players are searching for a return on their initial investment, which in most cases will be derived from the collectible nature of the parts and their relative scarcity within the game's ecosystem and economy. There must therefore be predictable scarcity, which in the case of Etherbots is achieved by a hard-capped presale, a small promotional allowance and a well-defined process of part inflation. While there is a limited number of possible part types, parts will be able to achieve differentiation from each other by virtue of their accumulated experience and their battle record, as well as by having particular rare characteristics.

This is one of the fundamental differences between Etherbots and other Ethereum games: value is created through playing time and strategy (much like regular games), rather than mere speculation or chance.

2.2 Longevity

CryptoKitties still remains popular, despite community interest having waned significantly since it dominated the Ethereum network in late 2017 and early 2018. This is largely due to its strong community, its position as a market leader, and the lack of non-ponzi scheme alternatives. However, in order for CryptoKitties - or any other collectible - to be desirable in the long term, games must offer the promise of continuing functionality: new tournaments, upgrades, and improved user experience. This can be achieved through the willingness of community members or developers to create extensions to the original contract, as well as the ability of the contract to interact with future systems (one of the key goals of ERC721 and standardisation more broadly).

Games must also negotiate the tension between immutable functionality and the fact that one mistake in an implementation can destroy the balance of the game. This is particularly important to consider when adding new battle types, as significant amounts of user investment will be put at risk during this process (for example, if it is easy to collude to gain experience). In general, we have adopted the following philosophy: where a mistake in the value would cause one party to be at a clear disadvantage, such that they would not engage in battle, the correct solution is to add a wholly new battle contract. Users can use the old one if they wish, but would have no incentive to do so if it were truly imbalanced. This is an excellent differentiator for well-designed Ethereum games - players do not have to

rely on the developers creating constantly successful and enjoyable updates, but are able to perform their own 'system restores' by calling a previous contract. However, where a mistake in the contract could enable colluding parties to gain an unfair advantage over other players, then developers must have enough governance ability to fix - or limit - the bug, through centrally controlled fields and functions.

3 Battling

The most fundamental mechanic of Etherbots, besides the ownership and transfer of parts/tokens, is the ability for robots to engage in battle. The results of these battles are determined primarily by user input, where users select fixed-length combinations of moves known as 'move strings', and these moves are executed in sequence. The possible moves are defined in the following relation:

$$\text{defence} > \text{attack} > \text{body} > \text{turret} > \text{defence}$$

Therefore, for any moves a and b which are part of a possible move set C, and given a predicate *defeats*(move, move) which may be implemented as $(a+1) == b \mod |C|$ for sequentially numbered moves, the damage done by both the attacker and the defender (with the total base round damage being td, the loser split l and winner split w) is as follows:

$$(\text{aDmg}, \text{dDmg}) = \begin{cases} (\frac{w \times td}{w+l}, \frac{l \times td}{w+l}), & \text{if defeats(a, b)} \\ (\frac{l \times td}{w+l}, \frac{w \times td}{w+l}), & \text{if defeats(b, a)} \\ (\frac{td}{2}, \frac{td}{2}), & \text{otherwise} \end{cases} \tag{1}$$

After base damage for the move has been calculated, the following modifiers are applied:

1. Part Level/Experience
2. Part Rarity Bonuses
3. User-based Perks
4. Element Combination Bonuses
5. Randomness

These modifiers each calculate a percentage bonus to the underlying damage. In order to prevent bias in the ordering of percentage application, the total percentage bonus is applied as a whole. However, in order for any battle to be *potentially* winnable by a player who has selected the optimal move sequence when compared to their opponent, the following constraint on the loser's perk bonus lpb is maintained for each winning move doing base damage wd and losing move doing base damage ld:

$$(lpb + 100)\% \times ld \leq (wpb + 100)\% \times wd \times (\frac{rr \times (wpb + 100)\% \times wd}{2 \times 100}) \quad (2)$$

where wpb is the perk bonus of the winner and rr is the random range discussed later. This ensures that, even amongst robots with vastly different levels of experience and part rarity, playing optimally will always give you at least a chance of success.

3.1 Part Level/Experience

For each level a part attains, it gains a damage bonus. This is represented by the equation below, where a is the 'level boost' and b is the number of levels between boosts.

$$bonus = \frac{a \times level}{b} \quad (3)$$

This bonus is currently set at half a point per level, with $a = 1$ and $b = 2$. This ensures that part levels play a significant role in determining the outcome of battles, without high level parts being at an immediate and significant advantage. There is currently no need for the level boost to be logarithmically scaled - we want to reward higher-leveled players and encourage them to play against each other, as well as creating a thriving market for pre-loved parts.

3.2 Part Rarity Bonuses

When using a part with a non-standard rarity (such as gold or shadow), there is a $s+n$ bonus for each part of that rarity in the current robot, where s is a fixed constant and n is the number of rarity tiers below the current part rarity. Currently s is fixed at 4, giving full shadow a 16% bonus and full gold a 20% bonus.

$$bonus = (s + n) \sum_{p=0}^{c} \begin{cases} 1 & \text{if rarity}(p) == a \text{ and rarity}(p) \text{ != } 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Where c is the number of parts, and a is the rarity of the 'moving' part as defined above.

3.3 User-based Perks

Perks are tied to user accounts, and may be unlocked through the acquisition of experience. Perks only apply where the 'moving part' activates those perks - for instance, the electricity perk will only apply when using electricity-based moves. For each active perk, the user receives a fixed bonus f. In addition to this, for every 'prestige' obtained by the user, this fixed bonus is increased by the current prestige level p.

$$bonus = (f + p) \times \text{activePerks(user, move)} \tag{5}$$

Users can prestige after filling the perk tree. However, 'prestiging' removes your current perk bonuses, and you must start completing the tree again from scratch, but with the benefit of higher bonuses once the perks are unlocked again. The perk levels are below:

1. Offensive/Defensive
2. Melee/Body/Defence/Turret
3. Android/Mech
4. Steel/Water/Fire/Electric

3.4 Element Combination Bonuses

Elemental bonuses provide a fixed percentage bonus of e based on how many elemental parts align with the 'moving' part.

$$bonus = (e - 1) \sum_{p=0}^{c} \begin{cases} 1 & \text{if element}(p) == a \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Where c is the number of parts, and a is the element of the 'moving' part. Note that $e-1$ is used to avoid granting a bonus to robots without elemental combinations. Currently, elements all do equal damage against each other, as a Pokemon-style system would have the effect of making all elemental bots very vulnerable when defending. However, it is being considered for future battle implementations, which may rely on a double commit and thus avoid this issue.

3.5 Randomness

Randomness is chosen by performing a bitwise XOR operation on the move strings. While this randomness is predictable where the parties are colluding, colluding parties could control the result of the battle in any case by selecting move strings which guarantee a particular victor. (A separate, and independently secure source of randomness generates battle rewards). Where both parties intend to defeat the other, it is an adequate way of creating variability in the outcome of any given duel.

Randomness is applied to the damage done by each player after each move, through the application of a percentage-based range of randomness (if $rr = 40$, values can range by 20% on either side of calculated damage).

$$rDmg = \begin{cases} dmg + \frac{dmg \times rand}{100} & \text{if rand} > \frac{rr}{2} \\ dmg - \frac{dmg \times rand}{100}, & \text{otherwise} \end{cases} \tag{7}$$

where $rand$ is the random seed mod rr. Randomness is applied after all of the part and perk bonuses are used to modify the base damage.

3.6 Outcome

The winner of each battle is therefore defined as the player with the highest total score over c moves:

$$score = \sum_{m=0}^{c} \text{applyRandom}(((\text{base}(a[m], d[m]) \times (100 + \text{bonus}(a[m])\%)))) \tag{8}$$

Where a is the move string of the attacker and d is the move string of the defender. The battle is reported in stages to allow for a suspenseful UI, but the entire calculation happens in one *executeMoves* internal call.

4 Rewards

Etherbots aims to reward players for both engaging in large amounts of gameplay and for playing skilfully. While grinding is still viable, investing effort into your battle strategies may result in significant long-term savings on both time and battle fees.

4.1 Cooldowns

One of the central features of CryptoKitties' gameplay is the cooldown which limits the ability of cats to breed. However, this mechanic (limiting the primary functionality of the game) can be frustrating for users - particularly in a game which tries to promote player activity and emphasise the ownership of parts. It is also a blunt and arbitrary restriction, which feels out of place in a game whose economy should be inherently secure without resorting to time-locks. To avoid these cooldowns, but prevent the system from being abused by those who have enough money to continually pay battle fees, Etherbots uses experience and reward cooldowns. These cooldowns, which are implemented as scaled fall-off functions, increase the incentive of players to focus on winning duels rather than playing volume. These cooldowns expire after a 24 hour period, and will never prevent a user from participating in a battle or tournament. They are not rolling, and will reset at 24 hour intervals (a necessary compromise to save significant amounts of storage and gas).

4.2 Experience

Players are rewarded with more experience for playing more successfully - in particular, for defeating robots which have a higher total level than their own and are therefore presumed to have a base combat advantage. Experience gained will apply to all parts used in the current battle, as well as the user's account, where it can be used to purchase perks. A crucial goal of the experience system is preventing colluding parties from using one high-leveled robot to 'boost' other parts, devaluing the effort of other players. It is thus necessary for users who lose battles, despite having a significant combat advantage, to lose experience. However, such losses must be rare where the parties have not colluded to pre-empt the outcome of a duel. Levels are linear, and each currently requires 1000 experience.

The primary formulas used to calculate the experience gained by both parties in battle are as follows, where w and l are the winner and loser's total

level, WS and LS are the winner's and loser's split, and WE and LE are abbreviations for WinExperience and LoseExperience respectively. $bMax$ and $bMin$ are the maximum and minimum amount of experience which can be awarded in a single battle:

$$totalExp = \max(\text{bMin}, \frac{\text{bMax} - \max(\frac{\text{-bMax}}{\text{bMin}}(w-l), \frac{\text{bMax}}{\text{bMin}}(w-l))}{\text{slowing factor}}) \tag{9}$$

$$winExp = \max(minWE, min(maxWE, \frac{(totalExp \times WS)(\frac{l}{w})}{WS+LS})) \tag{10}$$

$$loseExp = totalExp - winExp \tag{11}$$

At launch, the following constants will be set: $minWE = 75$, $maxWE = 1000$, $minLE = 250$, $maxLE = -900$, slowing factor $= 3$.

Each part's experience is related to the number of battles it has participated over a 24-hour period through the following quadratic. It begins at a vertex with y-coordinate v and then declines until it reaches an artificial asymptote a, with the constant c determining the rate of the falloff. However, if the part has a negative base experience, this function will not be applied, and the full cost of the experience will be deducted.

$$exp = \begin{cases} \max(a, \frac{-1}{c}(games)^2 + v) & \text{if base} > 0 \\ base & \text{otherwise} \end{cases} \tag{12}$$

At launch, the following constants will be set: $v = maxWE$, $a = minWE$, $c = 8$.

This also disincentivises very small bots from constantly attacking high-leveled defenders. They will consume the battle fee of both parties, but will be acting entirely against their own self-interest by selecting a target against whom (if they are not colluding) they have almost no chance of succeeding, and thus will have very low expected experience rewards. Further, they will be unable to reduce their own exp reward.

4.3 Part Rewards

After every battle, there is a chance of a reward being generated for each participant. These parts are dependent on performance in battle, and are

directly correlated to experience received, so that the part reward system is equally resistant to collusion. To allow for negative experience to impact this process, the distribution of randomness is between the maximum experience lost (-900) and the maximum gained in a single battle (1000). These rewards may be either parts, with rarity distributed in the same manner as in the presale crates, albeit with 1% of the parts being gold, and 2% being shadow, or part shards, which can be forged together to create a part. These shards are simply represented as a per-user count, and are not ERC721 tokens nor transferrable. The number of shards per part s is fixed at 500, and cannot be changed.

The drop distribution from reward crates in terms of parts is as follows, where a part will drop every c crates and the average number of shards in non-part reward crates is a:

$$\text{expected reward} = \frac{\text{part}}{c} + \frac{a}{s(1-c)} \tag{13}$$

This will be further modified by experience scaling.

At launch, a will be set to 1, but in order to ensure that the system remains responsive to market demand, this value will be centrally alterable. Some market-based solutions were considered but seem open to manipulation, particularly during periods of low transaction volume, and a highly variable battling fee may make core game functionality too expensive. A centralised solution, while undesirable from a principled perspective, protects the market from these challenges, and will be made transparent to allow battlers to always have full information about their current chance of reward. a should be initially low and increased only where required, rather than vice-versa in order to placate legitimate community concerns about developer manipulation.

The following function parts(t) estimates the number of parts in the game after t battles, and with ps presale parts and a small number of promotional parts pp (which will be given away (not sold) as advertising, or to start partnerships).

$$\text{parts}(t) = ps + pp + \frac{t}{\text{c}} + \frac{at(100-c)}{s} \tag{14}$$

This does not consider the deflation caused by users breaking parts down into shards (for $s \times 70\%$).

As the number of parts in the game increases, the number of battles being held will also increase (as users will be able to form more robots).

This creates a slow but steady linear increase in the number of parts relative to the number of battles, and should allow for the potential player base to increase without compromising the value of the parts purchased by early-adopters.