Etheraffle White Paper – Draft

Greg Kapka

Founder & Head Developer

❍

Acknowledgements

With special thanks to the endlessly supportive Abigail Whalley without whom the first step could never have been started. And to Diego Cardenas Ibanez & Stefania Palmisano without whom it would have never ended.

❍ Dedicated to Richard Kapka ❍

Contents

1.0 Mission

1.01 Vision_____2
1.02 Goals _____2
1.03 Why_____2

2.0 Platform

2.01 Etheraffle_____4
2.02 Current Format_____6
2.03 Player Procedure _____8
2.04 Smart Contract Procedure_____12
2.05 External Control _____17
2.06 Ether Ingress_____18
2.07 Affiliate Scheme_____18
2.08 Ether Egress_____19
2.09 Disbursal _____21
2.10 EthRelief _____24

3.0 The LOT Token

3.1 Particulars_____25
3.2 Features_____25
3.3 Smart Contract_____26

4.0 ICO

4.1 Overview_____27
4.2 Key Dates_____27
4.3 Tier System_____28
4.4 Bonus Structure_____29
4.5 Smart Contract_____31
4.6 Fund Usages_____33

5.0 Post ICO

5.1 Near Term_____36
5.2 Further Ahead_____37

6.0 FreeLOT

6.1 Particulars_____38
6.2 Features_____38

7.0 Bibliography

6.1 Notes_____40
6.2 Sources_____40
6.3 Contract Addresses_____40

1.0 Mission

1.1 Vision

Etheraffle, the world's first charitable blockchain lottery. Decentralized and powered entirely by smart-contracts on the blockchain. Built to provide funding for EthRelief: A DAO controlled platform for charitable giving and largest source of decentralized altruism in the world.

1.2 Goals

Etheraffle is an already functional, ethereum based, blockchain lottery platform whose goals going forward are threefold:

❍ To fully decentralise all platform logic by migrating onto the Swarm content distribution service as it develops whilst the same time dissolving the ĐApp ownership and contract-control to a voting-based, smart-contract powered, permission-less, democratic autonomous organisation formed of and run by the Etheraffle LOT token holders.

❍ To continue providing players with a simple and intuitive user experience that is designed to be navigable by anyone including those completely new to the ethereum ecosystem – giving opportunities to win large prizes whilst their ticket sales go on to provide a host of benefits for both the Etheraffle LOT token holders and for Etheraffle's sister company, EthRelief.

❍ To continue growing the ĐApp's market-share, profitability and usage worldwide in order to drive more funding to the EthRelief portal and effectively incentivize the LOT DAO running it.

The nascent EthRelief platform's goal going forward is singular:

❍ To create and provide the largest source of decentralised altruism on the blockchain, funded by Etheraffle, controlled and managed entirely by LOT token holders' votes as part of the LOT DAO, in order to ultimately dispense continuous and democratically-decided charitable donations worldwide.

1.3 Why

Lotteries are an extremely profitable industry, with a market cap in excess of $279.9BN[1] c.2015. So profitable in fact that many provide a stream of revenue that

is given over to funding of various sorts. However the size, allocation, usage and these funds is controlled entirely by and at the discretion of companies whose internal policies and decision making are private and opaque.

Charitable entities share similar opacity concerns – their not-for-profit nature belie the issue of a donor not knowing whether their contributed funds helped their intended recipient or pad the salaries of managerial staff instead.

Both of these concerns are solved by the transparency and immutability afforded by blockchain technology, and are at the core of what Etheraffle & EthRelief intend to address.

Running such enterprises on the blockchain also manifest a host of other benefits:

- **Provably Fair** – Everything occurs in the public domain.
- **Auditable** – Smart-contracts are open-source by default.
- **Logistically Scalable** – Ethereum knows no borders.
- **Low Overheads** – No physical infrastructure required.
- **Worldwide Market** – Ethereum's network spans the globe.
- **Impartial** – Code execution is objective, always.
- **No downtime** – It is impossible for the ethereum network to go offline.
- **No censorship** – Extreme transparency
- **Democratic** – There are no privileged ethereum addresses.

2.0 Platform

2.01 Etheraffle

Etheraffle is an already existing, fully working and richly featured blockchain lottery ĐApp powered by ethereum. It is the only ethereum blockchain lottery to mimic the format of real world lotteries in order to provide familiarity to players, larger payouts due to the week long accumulation of prize pools, and security against the miner—centric attack vectors.

Etheraffle's UI is designed to provide users with as friendly and seamless an experience as possible, allow them to interact with the blockchain in layman's terms, simply, and without the confusion. Those visiting the ĐApp via browsers unable to communicate with the blockchain are detected by the ĐApp itself and offered methods by which to get connected. Via large prize pools and an easy-to-use interface Etheraffle aims to entice non-ethereum users into cryptocurrency ecosystem.

All user interaction with the ĐApp is facilitated via ether, including the ticket prices and prize payouts. All prize payouts are available to the user in full regardless of size, and in a single transaction, directly into the withdrawer's wallet.

The UI provides simple and intuitive means to both enter the draw and withdraw winnings in as simple a way as possible.

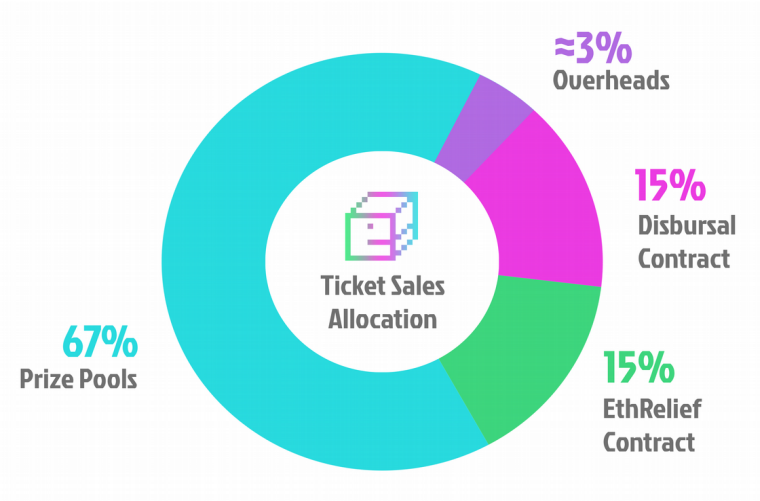
The entire lottery process is fully automated by the Etheraffle smart-contract and needs no outside interaction. Fail-safes within the contract ensure that in the event of unforeseen errors, players are able to enact ticket refunds. For technical details see sections 2.3 and 2.4.

Rigorous testing has been undertaken during its development. This testing is likely to continue by the community itself once the contract's prize pools grow and incentivize novel attack methods. We encourage this and going forward. A portion of ether raised during crowd-funding will be used to provide bug-bounties on the Etheraffle contract and any further contracts as and when they are published.

All Etheraffle smart-contracts are publicly viewable and auditable in their entirety by anyone via etherscan[2], at each contract's respective address in section 7.3. The source code is readily available and future contract's source will be made similarly accessible.

Play on Etheraffle requires no sign-up, no user-name and no email addresses. Etheraffle stores no data beyond what its smart contracts hold: The ethereum

address and corresponding selected numbers. The process is entirely anonymous and impartial and fair.



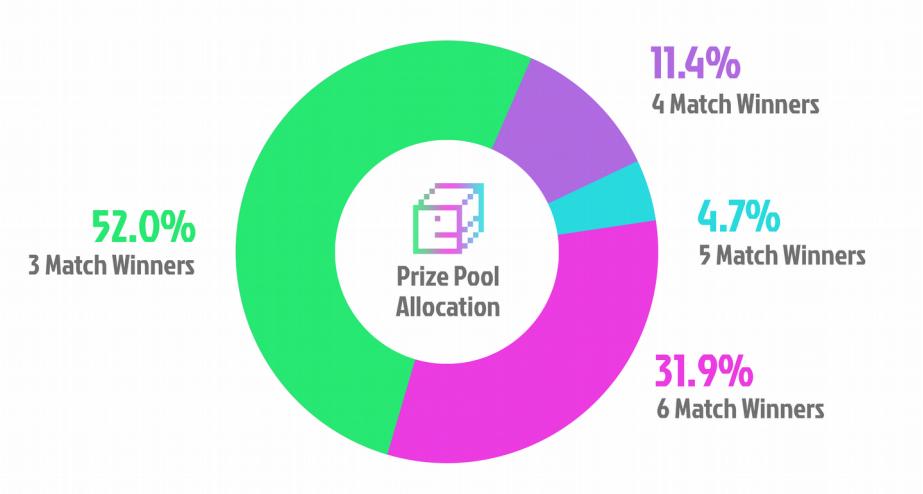
At the close of each week's draw, the ticket sales are tallied and split per the above chart by the smart contract. These figures are the currently split, as set during contract creation. They are governed by variables in the smart-contract that can be controlled externally, and which will eventually migrate to DAO control, and are therefore subject to change. See section 2.05 for more information.

2.02 Current Format

The current flagship format includes a single weekly lottery where six numbers are selected from a possible 49, and whose draw takes place on Saturday evenings at 8p.m. U.T.C. Players are able to enter the lottery during the week long period between Sundays 00:00 and Saturdays 19:00, as governed by the smart-contract. All prizes won are available for withdrawal from Sunday 00:00 following the draw. Users have a period of sixty days within which to make their withdrawal, beyond which any unclaimed prizes are automatically rolled back into available prize pool.

Lottery Odds	
Matches	Odds
0	1 in 2.29
1	1 in 2.42
2	1 in 7.55
3	1 in 56.6
4	1 in 1032
5	1 in 54200
6	1 in 13983816

Etheraffle's odds are combinatoric, meaning that the order of selected numbers is not relevant when matching them against winning numbers drawn. This provides more favourable odds to the user than the permutational methods seen elsewhere. Prizes are awarded to users matching three or more of their chosen numbers to the six randomly drawn winning numbers. After the draw, the prize pool is split into four fractions representing three, four, five and six match wins. Those fractions are then shared amongst the number of winners in that tier.



The draw itself is powered by Random.org as the source of randomness, and Oraclize[3] who facilitate the off-chain API calls in a decentralized and provable manner. This method is in lieu of a secure and reliable source of randomness on the blockchain itself, which area is undergoing active research in which Etheraffle hopes to take part post-ICO.

Oraclize power two necessary API calls in the Etheraffle contract: The first retrieves the six random numbers from Random.org and the latter calls Etheraffle's own endpoint which provides the contract with a count of the three, four, five and six match winners for that particular draw. Fault-tolerance is engineered into both of these API calls leaving the contract in a paused state in the event of failure. Should the contract not be revived after a period of two weeks, a function becomes available allowing users to enact refunds.

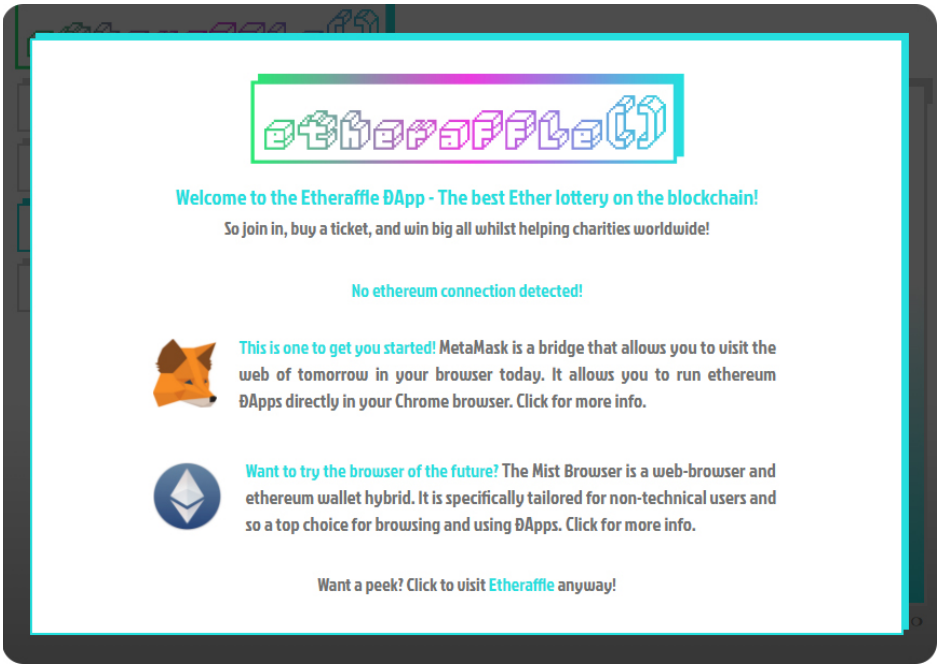
At a raffle's end, funds are first allocated to the gas needed for the Oraclize methods that power the raffle turnover. Next, the portion going to the disbursal contract for LOT-holder participation incentives is calculated and sent to the disbursal smart-contract. Then the portion going to EthRelief (see section 2.10) is similarly calculated (section 2.08) and sent to its smart-contract.

The second API call back then allows calculation of the three, four, five and six match win amounts which are then are set into the raffle's struct per section 2.04 and the variables explained in section 2.05. Finally, and the withdraw period for that raffle is opened.

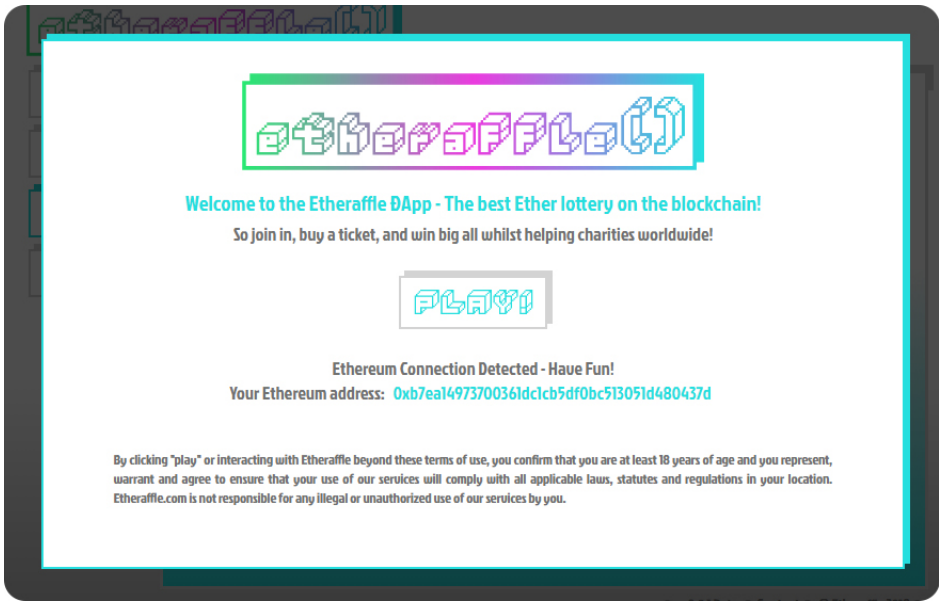
Fail-safes are included at every critical step of the raffle turnover. Should one trigger, the contract is paused arresting all methods. If the contract is not revived after a period of two weeks, the contract's refund method becomes automatically available for use.

2.03 Player Procedure

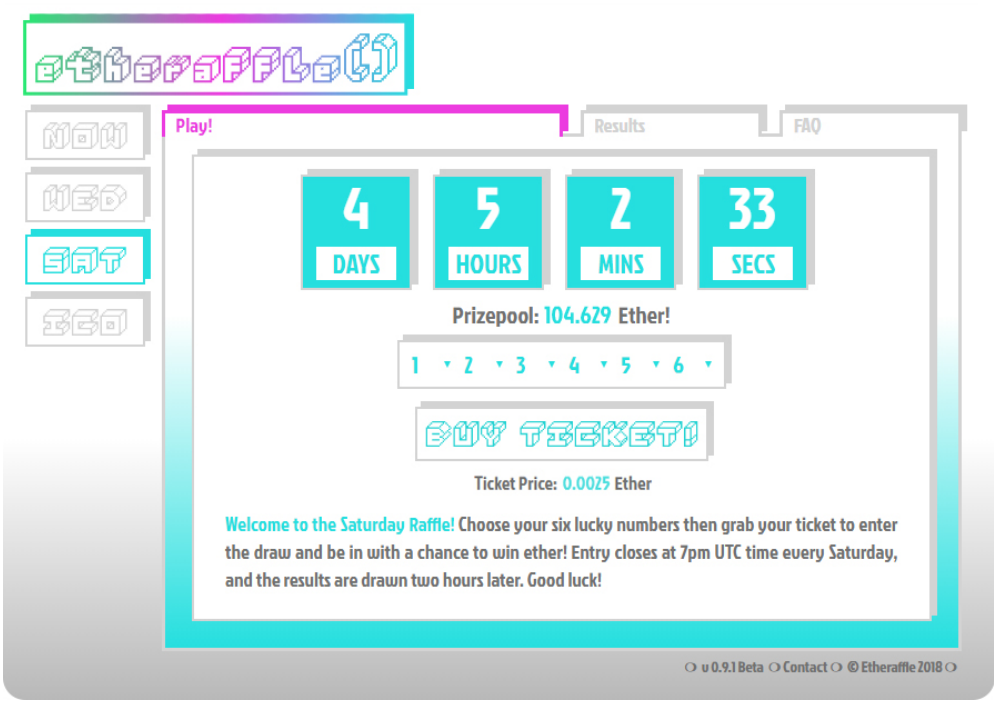
On visiting Etheraffle, the platforms detects the state of the user's ethereum connection. Should none be found, options are offered:



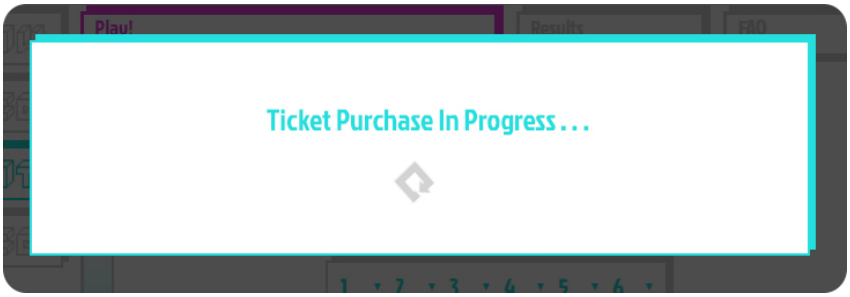
In the case of an active connection:



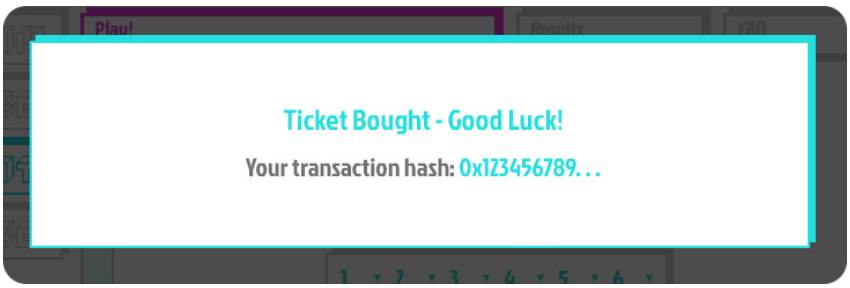
Once connected, the user is presented with a timer counting down until the close of the current raffle, and the raffle entry form:



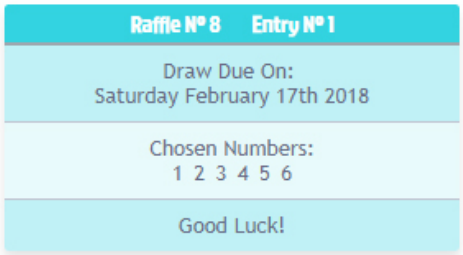
Using the drop down menus to select their six chosen numbers, the user then purchases the ticket via the 'Buy Ticket' button:



At which point their ethereum client of choice takes over. Once the user has signed the transaction:



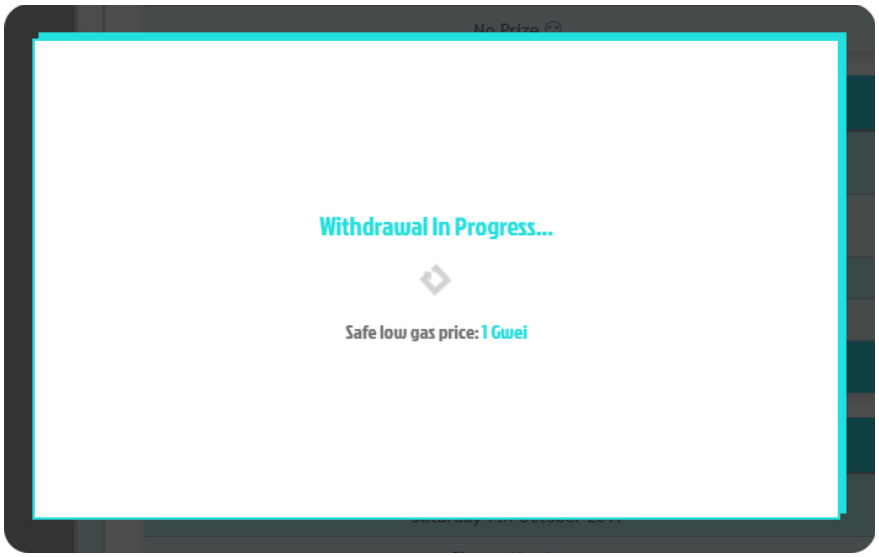
Once the transaction is mined, the user is successfully entered into the pending draw. Their ticket purchase is detected via an event logged by the smart-contract, after which it appears in the user's results tab:



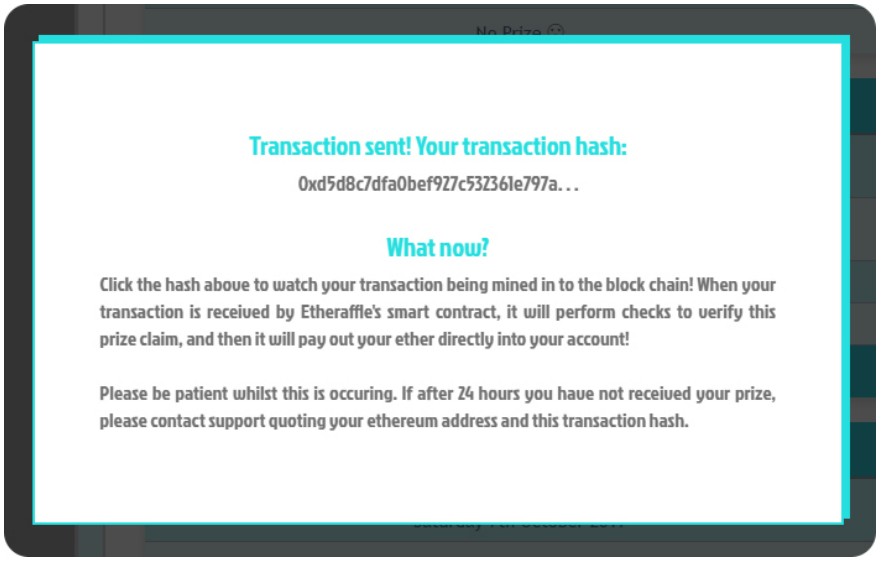
Tickets can no longer be purchased after 19:00 UTC on Saturdays. At 20:00 UTC the Oraclize methods are executed, performing the draw. Once complete, the results tab is updated to show the outcome. In the event of a non–winning ticket:

And of a winning ticket:

When claiming their prize via the button seen in the above figure, the user undertakes a similar transaction:



After which:



2.04 Smart Contract Procedure

The Etheraffle smart-contract tracks the prize pool by incrementing a global variable whenever tickets are purchased. It also tracks the week number based on the current block's timestamp and a constant integer defined as Etheraffle's birthday in epoch time. This becomes the raffle ID. Each week's raffle data is stored in a struct accessed via a mapping of that ID to it:

```
mapping (uint => rafStruct) public raffle;
struct rafStruct {
    mapping (address => uint[][]) entries;
    uint unclaimed;
    uint[] winNums;
    uint[] winAmts;
    uint timeStamp;
    bool wdrawOpen;
    uint numEntries;
    uint freeEntries;
}
```

The struct contains variables to track the number of entries, the number of free entries, the timestamp of that raffle's beginning, two arrays to hold the eventual winning numbers and corresponding prizes, and a second mapping of ethereum addresses to an array of integer arrays of the user's chosen numbers.

With that in place, section 2.02's user journey through the ĐApp occurs correspondingly as follows. On purchasing a ticket the enterRaffle method is called:

```
function enterRaffle(uint[] _cNums, uint _affID) payable external onlyIfNotPaused {
    require(msg.value >= tktPrice);
    buyTicket(_cNums, msg.sender, msg.value, _affID);
}
```

The first function parameter is an ordered array of the users six chosen numbers, which ordering is not enforced in the GUI but is executed before the calling of this method. User number's apparent ordering is thus maintained in the GUI despite said ordering having no bearing on the combinatoric lottery odds. The second parameter is an affiliate identification number to be used with the proposed affiliate scheme outlined in section 2.07. The function requires the ether sent with

the transaction to be greater than or equal to the current ticket price. The buyTicket function is then called.

```
function buyTicket(uint[]  _cNums, address _entrant, uint _value, uint _affID)
  internal
{
    require
    (
        _cNums.length == 6 &&
        raffle[week].timeStamp > 0 &&
        now < raffle[week].timeStamp + rafEnd &&
        0         < _cNums[0] &&
        _cNums[0] < _cNums[1] &&
        _cNums[1] < _cNums[2] &&
        _cNums[2] < _cNums[3] &&
        _cNums[3] < _cNums[4] &&
        _cNums[4] < _cNums[5] &&
        _cNums[5] <= 49
    );
    raffle[week].numEntries++;
    prizePool += _value;
    raffle[week].entries[_entrant].push(_cNums);
    LogTicketBought(/* Params... */);
}
```

The function parameters take in the array of chosen numbers, the address of the entrant, the ticket price paid and the affiliate ID. The functions requirements ensures that the chosen number array contains only six numbers, ensures that the current raffleID's struct contains a timestamp, and that the raffle has not yet closed as defined by the variable rafEnd. It then ensures that the chosen number array's integers are bound between 1 and 49, and that there are no duplicates. The function body goes on to increment the number of entries in the raffle struct, increment the prize pool by the price paid for the entry, and store the user's chosen numbers in the entries mapping of the raffle struct. Finally an event is triggered logging the pertinent data.

The closing of a raffle is defined by the rafEnd integer which is the number of seconds between the raffle's timestamp and the desired end time of the raffle. Raffle timestamps are generated upon the raffle struct initialization:

```
raffle[weekNo].timeStamp = birthday + (weekNo * weekDur)
```

Resulting in timestamps of that week's Monday at 00:00 UTC. The rafEnd variable of 500400 ∴ defines the raffle end time as Saturdays at 19:00.

The first raffle struct was initialized upon the mining of the Etheraffle smart-contract, which mining also began the recursive chain of Oraclize queries which power the raffle turnovers. The queries are set to return each Saturday at 20:00

UTC ensuring no overlap between the user's ability to enter a raffle and that raffle's results being known.

The Oraclize callback function executed when an API call returns is what enacts the raffle turnover. The function has two sections called depending on whether the API call was to Random.org or to Etheraffle's API.

```
function __callback(bytes32 _myID, string _result) onlyIfNotPaused {
    require(msg.sender == oraclize_cbAddress());
    LogOraclizeCallback(/* Params... */);
    reclaimUnclaimed();
    disburseFunds(qID[_myID].weekNo);
    setWinningNumbers(qID[_myID].weekNo, _result);
    //Prepare & send next Oraclize query...
}
```

The function is only callable by the Oraclize address, and only if the contract is not paused. It first reclaims any unclaimed prizes from the raffle ten weeks prior to the current one (∴ forming the sixty day withdrawal period), then executes the disburseFunds function (about which see section 2.09). The winning numbers are then set from the results of the callback. Finally a second Oraclize query is prepared and sent, this time to the Etheraffle API.

```
function __callback(bytes32 _myID, string _result) onlyIfNotPaused {
    require(msg.sender == oraclize_cbAddress());
    LogOraclizeCallback(/*  Params... */);
    newRaffle();
    setPayOuts(qID[_myID].weekNo, _result);
    //Prepare & send next Oraclize query...
}
```

This callback first initializes the next raffle struct, then sets the payouts in this raffle's struct as calculated from the number of matches received from the callback result, and finally prepares the next recursive query set due to return the following Saturday at 19:00 UTC, completing the iteration.

Payouts are set as follows:

```
function setPayOuts(uint _week, string _result) internal {
    /* Code converting results  string to uint array... */
   uint[] memory payOuts = new uint[](4);
    uint total;
    for(i = 0; i < 4; i++) {
        if(numWinnersInt[i] != 0) {
            payOuts[i] = (prizePool * pctOfPool[i]) / (numWinnersInt[i] * 1000);
            total += payOuts[i] * numWinnersInt[i];
        }
    }
    raffle[_week].unclaimed = total;
    prizePool -= raffle[_week].unclaimed;
    for(i = 0; i < payOuts.length; i++) {
        raffle[_week].winAmts.push(payOuts[i]);
    }
    raffle[_week].wdrawOpen = true;
    LogPrizePoolsUpdated(/* Params... */);
}
```

Validation checks resulting in contract pauses have been omitted for clarity. The function takes the string result from the API call and converts to an integer array consisting of the number of three match wins, four match wins, five and six match wins. It then calculates the prize per tier using the pctOfPool array as described in section 2.05 via:

```
payOut = (prizePool * pctOfPool) / (numWinnersInt * 1000);
```

The function sums the total winnings for that raffle and stores the amount in the 'unclaimed' variable, before subtracting it from the contract's global prize pool, thereby sequestering the raffle's winnings in its struct. Once stored, the withdraw is opened via the struct's boolean, completing the raffle process and allowing winner withdrawals. Upon winning, our section 2.03 user creates a withdraw transaction via the GUI which calls the following method:

```
function withdrawWinnings(uint _week, uint _entryNum) onlyIfNotPaused external {
    require
    (
        raffle[_week].timeStamp > 0 &&
        now - raffle[_week].timeStamp > weekDur - (weekDur / 7) &&
        now - raffle[_week].timeStamp < wdrawBfr &&
        raffle[_week].wdrawOpen == true &&
        raffle[_week].entries[msg.sender][_entryNum - 1].length == 6
    );
    // ...
```

The function takes in two parameters: The week number for the prize claim in question and the entry number of the user's entries for that week. The Etheraffle GUI encodes these automatically making the user's process as seamless as possible.

The function cannot be called if the contract is paused, nor if any of the requirements are not met. Those requirements demand the raffle to have been correctly timestamped, the current time to be beyond the sum of the end time of the raffle and its timestamp, the current time to be before the withdrawal deadline of sixty days, the raffles withdraw boolean to be true and that the entry in question's array is exactly six in length. The function continues:

```
    //...
    uint matches = getMatches(_week, msg.sender, _entryNum);
    require
    (
        matches >= 3 &&
        raffle[_week].winAmts[matches - 3] > 0 &&
        raffle[_week].winAmts[matches - 3] <= this.balance
    );
    raffle[_week].entries[msg.sender][_entryNum - 1].push(0);
    raffle[_week].unclaimed -= raffle[_week].winAmts[matches - 3];
    msg.sender.transfer(raffle[_week].winAmts[matches - 3]);
    LogWithdraw(/* Params... */);
}
```

It first compares the user's entry in question against that raffle's winning numbers as stored in the struct, returning the number of matches. Next it requires that the matches be greater than or equal to three, and that the prize amount is covered by the contract's current balance. A zero is then pushed into the user's entry array rendering a second withdrawal impossible. Finally it subtracts the amount won from that raffle's sequestered funds before making the ether transfer to the caller's address, completing the withdrawal.

2.05 External Control

Many of the variables pertaining to the running of Etheraffle have setter methods associated with them allowing external contract currently via the Etheraffle core's multi-signature wallet. They are as follows:

```
bool    public paused;
address public ethRelief;
address public etheraffle;
address public disburseAddr;
uint    public take         = 150;
uint    public gasAmt       = 500000;
uint    public rafEnd       = 500400;
uint    public wdrawBfr     = 6048000;
uint    public gasPrc       = 20000000000;
uint    public tktPrice     = 2000000000000000;
uint    public oracCost     = 1500000000000000;
uint[]  public pctOfPool    = [520, 114, 47, 319];
```

- **paused** – A boolean toggling the paused state of the contract. Entering raffles, withdrawing winnings and the Oraclize callback functions are all rendered unusable when the contract is paused.

- **ethRelief** – The address of the EthRelief contract

- **etheraffle** – The address of the Etheraffle multi-signature wallet.

- **disburseAddr** – The address of the disbursal contract or ether distribution to LOT holders.

- **take** – The percentage take in parts per thousand.

- **gasAmt** – The amount of gas provided to power the Oraclize callback function.

- **rafEnd** – End time in seconds after a raffle's initial timestamp.

- **wdrawBfr** – Period of time in seconds after which prize withdrawals are no longer possible.

- **gasPrc** – Price of gas in wei provided for the Oraclize callback.

- **tktPrc** – Price in wei of raffle entry.

- **oracCost** – Price in wei of the Oraclize service

- **pctOfPool** – Percentage split of prize pool allocated to each winning tier, in parts per thousand.

As development continues and Etheraffle dissolves its leadership to the DAO of LOT token holders, the integer variables will come under DAO control via voting methods being currently developed to allow incremental changes of them during the quarterly disbursal periods.

2.06 Ether Ingress

There are two paths for ether to enter the Etheraffle smart-contract and increase the global prize pool variable. The first is via ticket purchases for play in the raffles, as outlined in section 2.03. The second is via an add to prize pool function, which method allows for manual incrementing of the variable corresponding to the amount of ether sent along with the contract call:

```
function addToPrizePool() payable external {
    require(msg.value > 0);
    prizePool += msg.value;
    LogPrizePoolAddition(msg.sender, msg.value, now);
}
```

It is via this that prize pools will be increased following the LOT token sale in order to enhance platform attraction (see section 4.6). This method also provides forward-compatibility with future upgrades as discussed in section 2.08.

The Etheraffle smart-contract's fallback function cannot receive ether, and instead reverts should any be sent via a transaction with an absent/unrecognised function signature. Similarly, Etheraffle also has no token fallback function, making the mistaken sending of LOT tokens (and indeed any other ERC223 compliant tokens) to the contract impossible.

2.07 Affiliate Scheme

Present in the current beta of the Etheraffle smart-contract is the means to facilitate an affiliate scheme. Encouraging multiple skins and/or versions of 'Etheraffle' being created via other developers/interested parties will be incentivized by both access to the large Etheraffle prize pools and by the offering of entries at a rate allowing for mark-up.

It is proposed that such a scheme make use of a flexible-entry system whereby an approved list of addresses mapped to approved discounted entry rates is created, giving a method via which affiliates can be tracked and have their negotiated at-cost

rates bestowed upon an address under their control. Proposals by external parties for such privileges will be via the LOT DAO and thus could be scrutinised and voted for/against during the voting cycles.

```
function affiliateEntry(uint[] cNums, address _entrant) external {
    require(
        isAffiliate[msg.sender] &&
        msg.value >= affiliateRate[msg.sender]
      );
    buyTicket(_cNums, _entrant, msg.value, _affID);
}
```

In this way, such a scheme would help to increase the revenue flowing through Etheraffle for all the beneficiaries of the Etheraffle contract, including EthRelief, LOT holders via the voting DAO incentives, and the affiliates themselves.

The scheme currently tracks an affiliate ID per entry in to a raffle by whichever means, as one of the parameters of the entry function. Nothing further is done with the ID beyond an event logging its existence in the current contract, and development regarding the expansion and implementation of the scheme is ongoing as outlined in the roadmap in section 5.1.

2.08 Ether Egress

There are three paths for ether to exit the contract. The first is via the withdraw winnings function explained in section 2.03. Security in this method is achieved via the set heavy requirements gating the method, and the manner in which the payout is calculated from only the raffle ID, entry number and the ethereum address of the caller of the function. Since all winning and chosen number arrays are stored in the contract, and no methods exist by which they can be modified thereafter, there is no way to achieve a successful withdrawWinnings function execution and ∴ ether transfer except by satisfying fully the conditions of that function – i.e. by holding a winning ticket. Further, a successful execution extends the winning ticket's chosen number array thereby causing a second attempt at withdrawal execution to fail the requirements.

The second path is via the upgrade function:

```
function upgradeContract(address _newAddr) onlyEtheraffle external {
    require(upgraded == 0 && upgradeAddr == address(0));
    week        = 0;
    gasAmt      = 0;
    apiStr1     = "";
    prizePool   = 0;
    randomStr1  = "";
    upgraded    = now;
    upgradeAddr = _newAddr;
    uint amt    = prizePool;
    require(this.balance >= amt);
    etheraffleUpgrade newCont = etheraffleUpgrade(_newAddr);
    newCont.addToPrizePool.value(amt)();
    LogUpgrade(_newAddr, amt, upgraded);
}
```

By which method the Etheraffle contract is upgradeable without prejudicing unclaimed prizes by previous raffle winners. The function can only be called once, and only by the Etheraffle multi-signature wallet and will therefore form part of the suite of methods to be controlled by the future DAO. The function requires the upgrade contract to have the addToPrizePool method available in it, via which it transfers an amount of ether corresponding to the global prize pool variable in the contract. This the leaves the unclaimed prize pools sequestered in each raffle's structs still available to and correctly apportioned for future claimants. The function sets a timestamp upon completion which works in tandem with the contract's self destruct method:

```
function selfDestruct() onlyEtheraffle external {
    require(now - upgraded > weekDur * 10);
    selfdestruct(ethRelief);
}
```

The execution of which cannot be performed sooner than ten weeks beyond the upgrading event, preserving the sixty day deadline prize claimants have to withdraw. A successful self destruction clears the contract data and forwards any remaining ether to the EthRelief contract.

The final ether exit point is found in the disburse funds function:

```
function disburseFunds(uint _week) internal {
    uint oracTot = 2 * ((gasAmt * gasPrc) + oracCost);
    prizePool -= oracTot;
    uint profit;
    if(raffle[_week].numEntries > 0) {
        profit = ((raffle[_week].numEntries - raffle[_week].freeEntries) * tktPrice * take) / 1000;
        prizePool -= profit;
        uint half = profit / 2;
        disburseAddr.transfer(half);
        ethRelief.transfer(profit - half);
        LogFundsDisbursed(/* Params... */);
        LogFundsDisbursed(/* Params... */);
        return;
    }
    LogFundsDisbursed(/* Params... */);
    return;
}
```

Security checks resulting in contract pauses have again been omitted for clarity. Run each week and called by the Oraclize callback function, the disburse funds method first calculates the cost of those Oraclize calls using the variables discussed in section 2.5 and subtracts the cost from the global prize pool. Profit is then calculated, also subtracted from the prize pool and then halved before transferring each half to the disbursal contract and EthRelief contract in turn.

2.09 Disbursal

The Etheraffle disbursal contract is an endpoint for half of the ether profit generated by the Etheraffle platform. The contract will form the lynchpin of the LOT token holder DAO and the ether this contract receives will be used as reward incentives for the active participation in DAO activities by the LOT holders. It is through these activities that Etheraffle decentralization will be realised.

The disbursal contract currently published for the launch of the Etheraffle platform is a placeholder that has two pertinent functions. The first is a standard ether receiver method called by the Etheraffle contract when disbursing funds. The function accepts ether and logs and event upon arrival capturing the amount and the senders address.

```
event LogEtherReceived(address fromWhere, uint howMuch, uint atTime);

function receiveEther() payable external {
    LogEtherReceived(msg.sender, msg.value, now);
}
```

The second method is the future-compatible upgrade function which forwards any ether in the contract to a supplied future disbursal contract address by calling the method outlined above. The function is only callable by the Etheraffle multi-signature wallet and will be enacted once the development of the disbursal proposal outline below has been satisfactorily completed.

```
function upgrade(address _addr) onlyEtheraffle external {
    upgraded = true;
    LogUpgrade(_addr, this.balance, now);
    ReceiverInterface(_addr).receiveEther.value(this.balance)();
}
```

The disbursal contract upgrade is under active development with a goal deployment of the end of Q2 2018. The proposed mechanics for the next disbursal contract follow, and include the first example of an Etheraffle contract variable coming under control of the LOT DAO, as driven by an ether incentive.

First, the LOT token's transfer functions are freezable, and the contract has within it a method by which to set the frozen status of those transfers (see section 3.2). That method is only callable by an approved list of addresses, of which the future disbursal contract will be made one.

There will be a hypothetical voting function whereby after voting, an ether reward is given per the voter's LOT holdings. In order for those ether rewards to be correctly apportioned out, the movement of the LOT token needs to be temporarily halted. With that achieved, ether rewards per address can be calculated accurately and fairly via:

$$\frac{L_n}{L_{tS}} \times Q_x \, \Xi$$

$$\begin{aligned} L_n &= \text{number of LOT} \\ L_{tS} &= \text{LOT total supply} \\ Q_x &= \text{year quarter} \end{aligned}$$

However, should token transfer not be arrested, a user can artificially inflate their apparent LOT holdings by moving them to a new address and calling the voting function again, and successfully receive a further reward. With the token transfer disabled, this would not be possible.

And so, by endowing the disbursal contract's address with the ability to freeze or unfreeze the LOT token transfers, and then including the function calls to do so in the disbursing contract's voting method, we incentivize the behaviour of the DAO.

Next we make the voting method public and callable by any one. Assuming the voting cycles occur in the last week of each quarter, the voting method can be rendered only executable in the last week of each quarter via checking the time of the call against a valid week where the valid week is equal to:

$$\left(\left(\left(\frac{b_t - s}{w \times 13}\right) \bmod 4\right) \times 13\right) + 12$$

where:

b_t = block timestamp
s = any year beginning in UNIX time where s < b_t
w = week duration in seconds

When called in a valid week the function can go on to inspect the token contract's frozen status, freezing it if necessary, before carrying out the voting logic and transferring the ether reward.

```
function castVote() external {
    if(LOT.frozen() && isValid()) {
        logVote(/* Params... */);
        sendReward(msg.sender);
        return;
    } else if(!LOT.frozen() && isValid()) {
        LOT.setFrozen(true);
        LogFreezing(/* Params... */w);
        logVote(/* Params... */);
        sendReward(msg.sender);
        return;
    }
    //...
```

Because the Etheraffle contract transfers ether each week to the disbursal contract, the incentives accumulate until a valid week is reached, at which point LOT holders are able to execute the function, cast their vote, and receive their reward.

Additionally, the voting method should also contain a pathway via which if called in a non-valid week and finding the LOT token still frozen, would unfreeze it. This needs no ether incentive since any requirement by any one of the DAO members at any point between valid weeks to move their LOT will motivate their calling of this method. Further, in this way any ether incentive remaining at the close of a valid voting week can be rewarded to the EthRelief contract via the same pathway:

```
//...
else if(LOT.frozen() && !isValid()) {
    LOT.setFrozen(false);
    LogFreezing/* Params... */);
    FreeLOT.mint(msg.sender, free);
    uint y = getQuarter() > 0 ? getYear() : getYear() - 1;
    uint q = getQuarter() > 0 ? getQuarter() - 1 : 4;
    uint amt = redeemed[y][q];
    if(amt > 0) {
        redeemed[y][q] = 0;
        EthRelief.receiveEther.value(amt);
        LogDonation(/* Params... */;
        return;
    }
    LogDonation(/* Params... */);
    return;
}
```

And so whilst no further incentive is required for the above, it is proposed to use the FreeLOT coupon (see section 6.0) to add an additional small bounty to the user calling this function, to go some way in offsetting their gas outlay.

2.10 EthRelief

The EthRelief contract is second recipient of the Etheraffle contract's disburse ether function, and will form the locus from which charitable donations are proposed, handled and emitted.

The voting framework for such events is under active, current research and development, using DAO participatory mechanisms similar to those proposed in the previous section.

In the early stages of the Etheraffle enterprise, funds for EthRelief will be allowed to accumulated in a placeholder identical to the disbursal placeholder described in section 2.09, whilst the development of the voting mechanisms and methodologies takes place. Meanwhile the disbursal contract will be live-testing proposed schema for said voting protocols.

Modular roll-outs are planned in order to slowly implement the voting blueprints found in section 2.09. These will be made available to beta-testers comprised of early ICO participants, referenced from the tier system of the ICO outlined in section 4.3.

The EthRelief white paper is due end of Q2 and will describe the project in detail and provide the roadmap for the initial implementations for gradual platform launch beginning the end of Q3.

3.0 The LOT Token

3.1 Particulars

The Etheraffle LOT token is an ethereum based ERC223 compliant token, and is granular to six decimal places. It is non–mintable and non–destroyable. The total supply is fixed at 3,400,000,000. The ERC223 standard is an upgrade to the more common ERC20 standard and improves security of the token transfer process whilst retaining backwards-compatibility with all ERC20 transfer methods. Moving an ERC223 token requires the destination, if a contract, to have a token fallback method. Absent this method, the token transfer fails thus preventing accidental loss of tokens by inadvertent transfer to contracts not designed to receive them. The total supply of LOT is derived from the ICO structure as outlined in section 4.0 and is calculated via:

$$\sum_{m=0}^{3} \left((tC_m \times tR_m) + \sum_{n=(m+1)}^{3} tC_n \times 1500 \right)$$

where:

tC_x = tier x ether cap
tR_x = tier x LOT exchange rate

3.2 Features

The Etheraffle LOT token's transfer functions are able to be frozen by the switching of a boolean. The switching function is callable only by designated freezers, whose addresses can only be added or removed from the role by the Etheraffle multi-signature wallet at the time of writing. Said functionality will subsequently migrate to DAO control per the roadmap in section 6.0. Currently, only the Etheraffle multi-signature wallet and the disbursal contract currently in beta-testing outlined in section 2.09 have freeze privileges.

Token freezing is necessary for and driven by the disbursal contract's voting events. Temporarily removing the ability to move the LOT token to different addresses during the disbursal period avoids the artificial inflating of apparent holdings and ensures fair distribution of ether incentives per LOT holdings. The token being frozen will not affect its ability to be traded on the exchanges, except those running on the ethereum blockchain itself.

Holding the LOT token endows upon its owner the ability and right to participate in any and all of the elements of the DAO that will eventually run and control the

entirety of the Etheraffle and EthRelief platforms. Whilst incentivized by ether, active participation is in any and all aspects of the DAO is not a requirement and is entirely at the holder's discretion.

3.3 Smart Contract

The ERC223 compliant transfer function is as follows:

```
function transfer(address _to, uint _value, bytes _data) onlyIfNotFrozen external {
    uint codeLength;
    assembly {
        codeLength := extcodesize(_to)
    }
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to]        = balances[_to].add(_value);
    if(codeLength > 0) {
        ERC223Compliant receiver = ERC223Compliant(_to);
        receiver.tokenFallback(msg.sender, _value, _data);
    }
    LogTransfer(msg.sender, _to, _value, _data);
}
```

Of note is its checking the destination address for code, and calling the token fallback method should it find such. The method is also governed by the modifier function that conveys the ability to freeze and unfreeze token transfers:

```
modifier onlyIfNotFrozen() {
    require(frozen == false);
    _;
}
```

The following method allows the frozen status to be set to the desired boolean, and is gated by the modifier above it requiring function callers to exist amongst the list of approved freezers:

```
modifier onlyFreezers() {
    require(canFreeze[msg.sender]);
    _;
}

function setFrozen(bool _status) external onlyFreezers returns (bool) {
    frozen = _status;
    LogFrozenStatus(frozen, now);
    return frozen;
}
```

4.0 ICO

4.1 Overview

The Etheraffle ICO involves the exchange of ether for the ERC223 LOT token. ICO participants, and ∴ LOT holders, will eventually form the Etheraffle DAO – the decentralized custodians of the ownerless and trustless Etheraffle and EthRelief platforms. Owning LOT guarantees membership to the LOT DAO. Any and all participation in the DAO is entirely voluntary. Active participation is rewarded in ether, relative to LOT holdings (see section disbursal 2.09).

The ICO structure consists of three tiers with corresponding exchange rates. The spread of the exchanges rates is narrower than is typical of past ICOs, so as not to prejudice final tier participants. Instead, the Etheraffle LOT token offers a unique, tiered, stacking bonus system, about more which see section 4.4.

Purchasers may purchase LOT tokens from as many tiers as they wish, and so can become eligible for multiple bonuses. Any LOT purchased via the ICO contract are transferred to the purchasers wallet as soon as the transaction is mined. Any bonus LOT becomes available for redemption upon completion of the crowd sale.

The Etheraffle ICO has a minimum entry of 0.025 ether. This lower threshold is to encourage as wide a cross section of ethereum users as possible, enhancing the quality of the DAO.

The ICO contract's address:

0x00E52182bE36A55161Fe40b19a4C29eeF0017019

4.2 Key Dates

Pre–ICO Begins:_____Monday 19th of February 2018 at 00:00

Tier 1 Begins: _____Saturday 10th of March 2018 at 00:00

Tier 2 Begins: _____Saturday 24th of March 2018 at 00:00

Tier 3 Begins: _____Saturday 14th of April 2018 at 00:00

ICO Deadline: _____Friday 11th of May 2018 at 23:59

Redeem Bonus Deadline: ___Friday 25th of May 2018 at 23:59

4.3 Tier System

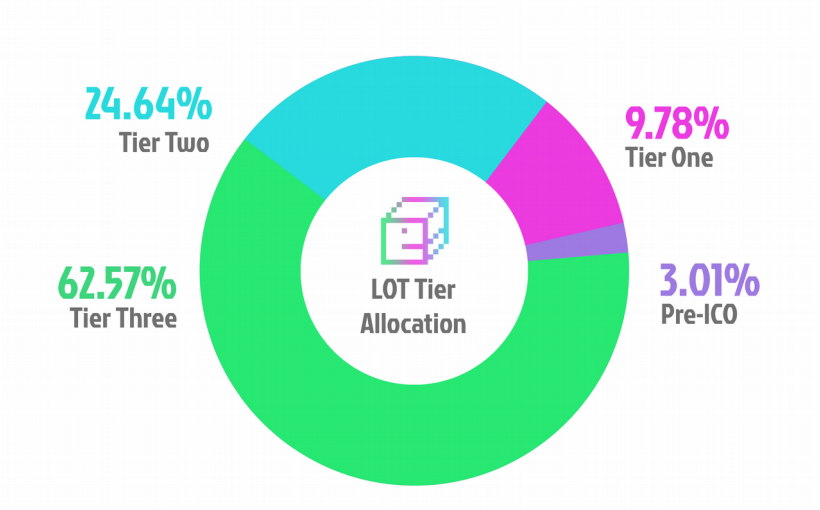
Etheraffle ICO Particulars			
Tier	Exchange Rate (LOT/ETH)	Length (Weeks)	Cap (ETH)
0	110,000	1	700
1	100,000	2	2,500
2	90,000	3	7,000
3	80,000	4	20,000
All	1,500 Bonus LOT per ETH per Tier	n/a	n/a

The Etheraffle ICO has 3 tiers and a pre-ICO tier zero. As tiers progress, both the ether cap and the tier's duration increases. This is to to allow for the growth of larger bonus LOT pools and the wider dispersion of the LOT token itself for the benefit of the DAO.

Entry into the ICO involves sending ether to the smart contract. Etheraffle.com/ICO provides a GUI to facilitate participation if desired. Sending ether by any manner to the ICO contract address in section 4.1 during the ICO period results in the purchase of LOT tokens, which are sent to the purchaser's wallet as soon as the transaction is mined. The ICO contract fallback function undertakes this task:

```
function () public payable onlyIfRunning {
    require(now <= tier3End && msg.value >= minWei);
    uint numLOT = 0;
    if (now <= ICOStart) {// :. tier zero...
        require(tier0Total + msg.value <= maxWeiTier0);
        tier0[msg.sender] += msg.value;
        tier0Total += msg.value;
        numLOT = (msg.value * tier0LOT) / (1 * 10 ** 18);
        LOT.transfer(msg.sender, numLOT);
        LogLOTTransfer(msg.sender, 0, msg.value, numLOT, now);
        return;
    } else if (now <= tier1End) {
        //Tier 1 version of code block above...
    } else if (now <= tier2End) {
        //Tier 2 ibid
    } else {
        //Tier 3 ibid
    }
}
```

Transferring ether to the ICO contract invokes this function which sifts purchasers into the correct tier based on the block timestamp, and transfers LOT at the corresponding tier's exchange rate. Requirements throughout the method ensure the ether caps are adhered to, and preserve the minimum purchase level as outlined prior. The function also maps the addresses of participants to the amount of ether they have sent to whichever tier, in order to correctly calculate bonuses available at the close of the ICO. Once that ICO deadline has passed, the function reverts any ether transfers enacted thereafter, arresting any further attempts at LOT purchasing, securely ending the ICO.



The tier structure results in the LOT being split amongst the tiers as in the preceding figure. 20% of the total minted LOT token are held in reserve for the Etheraffle team

4.4 Bonus Structure

Etheraffle's ICO design follows the tiered structure and adds in a stacking bonus system. For every ether raised in each tier following the pre-ICO stage, an additional 1500 LOT tokens are created and accumulate as that tier's bonus pool. This bonus pool is then shared out equitably amongst the previous tier's purchasers, per their percentage share of the ether raised in that previous tier. Purchasers receive a share of all bonus pools of each tier subsequent to the tier in which they purchased LOT tokens. Bonus LOT can ∴ be summed:

$$\sum_{m=0}^{3} \left(\sum_{n=(m+1)}^{3} \left(\frac{t\Xi_m}{\sum t\Xi_m} \right) \times \sum t\Xi_n \times 1500 \right)$$

where:

$$t\Xi_x = \text{tier x ether}$$

Bonuses are available for redemption during the period:

Redemption Period: _____Saturday 12th of May 2018 at 00:00

Redemption Deadline: _____Friday 25th of May 2018 at 23:59

The ICO contract provides a single function for bonus redemption, a GUI for which will be provided at Etheraffle.com/ICO as of the redemption period beginning. An abridged version of this function:

```
function redeemBonusLot() external onlyIfRunning {
    require(now > tier3End && now < wdBefore);
    require
    (
        tier0[msg.sender] > 1 ||
        tier1[msg.sender] > 1 ||
        tier2[msg.sender] > 1 ||
        tier3[msg.sender] > 1
    );
    uint bonusNumLOT;
    if(tier0[msg.sender] > 1) {
        bonusNumLOT +=
        ((tier1Total * bonusLOT * tier0[msg.sender]) / (tier0Total * (1 * 10 ** 18))) +
        ((tier2Total * bonusLOT * tier0[msg.sender]) / (tier0Total * (1 * 10 ** 18))) +
        ((tier3Total * bonusLOT * tier0[msg.sender]) / (tier0Total * (1 * 10 ** 18)));
        tier0[msg.sender] = 1;
    }
    if(tier1[msg.sender] > 1) {
        bonusNumLOT +=
        ((tier2Total * bonusLOT * tier1[msg.sender]) / (tier1Total * (1 * 10 ** 18))) +
        ((tier3Total * bonusLOT * tier1[msg.sender]) / (tier1Total * (1 * 10 ** 18)));
        tier1[msg.sender] = 1;
    }
    if(tier2[msg.sender] > 1) {
        bonusNumLOT +=
        ((tier3Total * bonusLOT * tier2[msg.sender]) / (tier2Total * (1 * 10 ** 18)));
        tier2[msg.sender] = 1;
    }
    if(tier3[msg.sender] > 1) {
        tier3[msg.sender] = 1;
    }
    if(bonusNumLOT > 0) {
        LOT.transfer(msg.sender, bonusNumLOT);
    }
    FreeLOT.mint(msg.sender, bonusFreeLOT);
    LogBonusLOTRedemption(msg.sender, bonusNumLOT, now);
}
```

Purchaser's ether amounts are defaulted to 1 wei once the bonus LOT calculations are performed for two reasons. The first provides the security mechanism whereby the method requires the caller to have greater than 1 wei associated with their

address in one or more structs, making second bonus redemptions impossible. The second is that via this the ICO contract upon completion maintains a ledger of addresses of purchasers in each tier. This ledger will later be referenced when rolling out developmental updates with respect to the DAO and the EthRelief platform, in order to offer beta-access to each tier in turn. This will not affect the Etheraffle disbursal contract's usage however, the features and ether rewards of which every LOT holder will have guaranteed access to at the close of the ICO.

As a final bonus, all LOT purchasers receive on execution of this function ten FreeLOT tokens. For more information on these, see section 6.1.1.

4.5 Smart Contract

Other functions in the ICO smart-contract include a transfer ether method callable only by the Etheraffle multi-signature wallet:

```
function transferEther(uint _tier) external onlyIfRunning onlyEtheraffle {
    if(_tier == 0) {
        require(now > ICOStart && tier0Total > 0);
        etheraffle.transfer(tier0Total);
        LogEtherTransfer(msg.sender, tier0Total, now);
        return;
    } else if(_tier == 1) {
      //Tier 1 version of previous code block...
    } else if(_tier == 2) {
      //Tier 2 version of previous code block...
    } else if(_tier == 3) {
      //Tier 3 version of previous code block...
    } else if(_tier == 4) {
        require(now > tier3End && this.balance > 0);
        etheraffle.transfer(this.balance);
        LogEtherTransfer(msg.sender, this.balance, now);
        return;
    }
}
```

The function ensures funds are available immediately after a tier for development purposes in order to meet the Q2 disbursal deadline. See sections 4.6 and 5.0.

A similar function for the transferral of LOT tokens also exists:

```
function transferLOT() onlyEtheraffle onlyIfRunning external {
    require(now > wdBefore);
    uint amt = LOT.balanceOf(this);
    LOT.transfer(etheraffle, amt);
    LogLOTTransfer(msg.sender, 5, 0, amt, now);
}
```

Again, only callable by the Etheraffle multi-signature wallet and only after the bonus redemption and ∴ the ICO is complete. It returns any unsold LOT tokens to the control of Etheraffle.

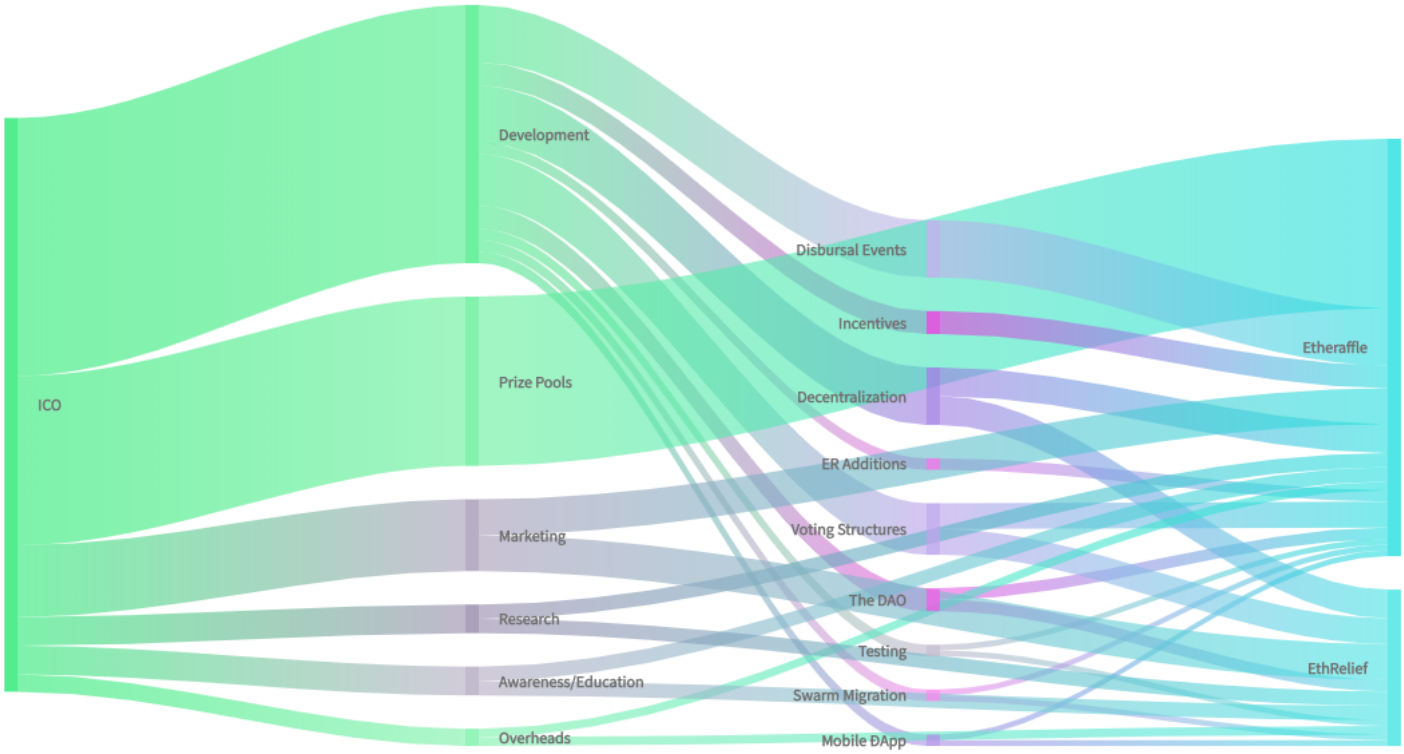
Also included is a refund mechanism for user security, callable in the event of the ICO being cancelled for any unforeseen circumstances. A GUI has been provided should the necessity arise and will be found at Etheraffle.com/ICO along with any pertinent information.

```
function refundEther() external onlyIfNotRunning {
    uint amount;
    if(tier0[msg.sender] > 1) {
        amount += tier0[msg.sender];
        tier0[msg.sender] = 0;
    }
    if(tier1[msg.sender] > 1) {
        amount += tier1[msg.sender];
        tier1[msg.sender] = 0;
    }
    if(tier2[msg.sender] > 1) {
        amount += tier2[msg.sender];
        tier2[msg.sender] = 0;
    }
    if(tier3[msg.sender] > 1) {
        amount += tier3[msg.sender];
        tier3[msg.sender] = 0;
    }
    msg.sender.transfer(amount);
    LogRefund(msg.sender, amount, now);
    return;
}
```

The method automatically sums any ether the caller may have in any tier before making the refund.

4.6 Fund Usage

The following figure provides an overview of the flow of ether post-ICO.



An interactive version of this diagram can be found at http://etheraffle.com/ico

Prize Pools ≈ 29.5%:

Funds will be used to seed the Etheraffle flagship raffle's prize pool per a time released schedule so as to increase attractiveness and engagement. The seeding will be accompanied by ongoing testing including the offering of bug-bounties to galvanise the Etheraffle contract ensuring its security and functionality in the longer term.

Research ≈ 5%:

Focusing on both blockchain randomness and the manifold technological solutions with respect to the decentralized voting in the EthRelief specification. Etheraffle also intends to partake in the ICO tradition of placing bounties on desired core ethereum components to help drive platform architecture growth.

Marketing ≈ 12.5%:

A continuation and ramping up of ongoing marketing and advertising strategies with focus shifting to Etheraffle platform use and engagement alongside generating public consciousness of the new paradigms herein. Also for drawing attention to the nascent EthRelief platform, and the use-cases thereof.

Awareness / Education ≈ 5%:

Allocated for the educating of donor recipients with respect to their abilities/understanding of how to receive funding via ether. Driving cryptocurrency awareness via the Etheraffle platform, serving as a jumping-off point to understanding how to interact with the blockchain.

Overheads ≈ 3%:

For the ongoing procedural costs of running the enterprise. Subject to change pending requirements for an increase in team size/structure going forward.

Development ≈ 45%:

- **Disbursal Events** – Expansion and manifestation of the ether reward incentive schema outlined in section 2.09 powering and incentivizing the active participation by LOT holders in the voting DAO

- **Voting Structures** – Design and implementation of the voting protocols to will govern the platform contract variables at the heart of the Etheraffle DAO

- **Incentives** – The design and implementation of the incentive schema powering the DAO

- **Decentralization** – To develop the smart-contracts and decentralized managerial frameworks necessary for the eventual full migration of control to the Etheraffle DAO.

- **The DAO** – DAO format, functionalities and implementation of methods to transfer increasing amounts of smart-contract control and responsibilities to the LOT token holders.

- **Etheraffle Additions** – To expand the suite on offer at the Etheraffle platform, in order to maximise engagement and increase funding for EthRelief and the DAO participation rewards. To develop and implement the affiliate scheme as outlined in section 2.07, and in tandem, develop the make available software development kits for future affiliates to tie into the main Etheraffle smart-contract(s).

- **Swarm Migration –** To decentralize and open source the current front-end framework via migration of Etheraffle codebase to the Swarm network for further dissolution of ownership to the voting DAO.

- **Testing –** Continual platform stress testing and test-driven future contract creation.

- **Mobile ĐApp –** Etheraffle's existing ĐApp is fully functional on mobile devices when using the ethereum-enabled Cipher Browser or similar, however a full, stand-alone app implementation is in development removing reliance on third party software.

- **EthRelief –** The creation of the future EthRelief blockchain protocol. The worlds largest source of decentralized altruism. About more which please refer to the EthRelief white paper.

5.0 Post ICO/Roadmap

Post-ICO, the developmental road map will have particular focus on the disbursal contract schema from section 2.09. A live version of the implementation proposed will be in place for the end of the Q2, to function as a real-world test and proof of concept for the DAO participation ether reward mechanisms.

5.1 Near-Term

Q1 2018:

circa February:
- Etheraffle launch.
- ICO launch.

Q2 2018:

circa May:
- Complete beta-testing of proposed disbursal mechanisms.

circa June:
- Deploy v1.0 of disbursal contract.
- EthRelief white paper.

Q3 2018:

circa July:
- LOT listing on exchanges.
- Roadmap update.

circa Sept:
- Modular roll-out of EthRelief functionality begins.

5.2 Further Ahead

After which near-term and mission critical elements, developmental foci will shift to a broader set of goals whose exact timeline will be elaborated on in future roadmap updates. They include but are not limited to:

Growth of the Etheraffle platform via:

- Affiliate scheme implementation
- Additional language support
- SDK Development
- Swarm DAO & Etheraffle ĐApps

Development of the DAO use cases and range of abilities via:

- Disbursal contract upgrades
- Further incentivization schema for DAO
- Voting framework for passing existing contract control to DAO

And finally development of the EthRelief platform via:

- Voting theory research
- Voting format and relevant architecture.
- EthRelief ĐApp development.
- EthRelief platform/chain developments.

6.0 The FreeLOT Token

6.1 Particulars

The FreeLOT token is an ethereum–based, ERC223 compliant token (See section 3.1 for more details). It is indivisible, of flexible supply and controlled wholly by the Etheraffle multi–signature wallet, and so in future by the LOT DAO. The purpose of the FreeLOT token is to act as a coupon via which free tickets to Etheraffle can be given out for promotional and like purposes.

6.2 Features

The token can be both minted and destroyed. Both abilities are executable only by approved addresses, which approval can only be granted and forfeited by the Etheraffle multi–signature wallet. Current minters and destroyer's addresses are stored in the token contract's publicly scrutable mappings:

```
mapping (address => bool) public isMinter;
mapping (address => bool) public isDestroyer;
```

To redeem the coupon, the Etheraffle contract has in it a second entry method:

```
function enterFreeRaffle(uint[] _cNums, uint _affID) payable external onlyIfNotPaused {
    freeLOT.destroy(msg.sender, 1);
    raffle[week].freeEntries++;
    buyTicket(_cNums, msg.sender, msg.value, _affID);
}
```

Which method takes in the same parameters as the standard entry function explained in section 2.4, however it has no ether requirement for execution. Instead, it attempts to destroy one of the caller's FreeLOT coupons. Failure to do so reverts the function and no free entry is achieved. In this way, one FreeLOT coupon grants the owner one and only one free entry into an Etheraffle raffle of their choosing.

```
uint public redeemed;
uint public totalSupply;
```

Regarding supply, the FreeLOT contract tracks two global and public balances; the total supply being incremented upon every FreeLOT minting; and the redeemed

amount which is decremented upon every destruction event. The difference between the two then reflecting the number of FreeLOT coupons currently in existence, awaiting redemption.

FreeLOT coupons can be exchanged freely between addresses by their owners. Once minted, and before being used, the FreeLOT has no restrictions on movement and can be passed between addresses at will via a standard token transfer transaction. There are no time limits on coupon redemption.

As of writing, the FreeLOT contract has two approved minters – the Etheraffle multi-signature wallet, and the Etheraffle ICO address, and two approved destroyers – the Etheraffle multi-signature wallet and the main Etheraffle contract.

7.0 Bibliography

7.1 Sources:

1. http://totallygaming.com/news/lottery/global-lottery-industry-maintains-growth-trajectory

2. https//etherscan.io

3. https://www.oraclize.it/

7.2 Notes

a. https://www.etheroll.com pioneered this mechanic, and still serve as proof of concept today.

7.3 Contract Addresses

All of Etheraffle's deployed contracts may be examined in full at etherscan[2] via their respective addresses:

Etheraffle:
0x4251139bf01d46884c95b27666c9e317df68b876

The Etheraffle ICO:
0x00e52182be36a55161fe40b19a4c29eef0017019

The LOT Token:
0xd70b659ae2c61fc52a31723af84a1922747feab7

The FreeLOT Token:
0x4c388dce25665ea602b92f15718ca278bba45a9a

The Disbursal Contract (placeholder):
0x3bfb12ed112aB833F275Dbf622b7CacC4CBF092b

EthRelief (placeholder):
0x63E14Dc47003d87d152eD507DF3F50C20bD88c3e