Matchpool

Philip Saunders, Yonatan Ben Shimon

January 14, 2017

Abstract

Matchpool is a decentralized matchmaking protocol which uses group dynamics to help as many participants as possible to find love. It can also be used as a more generic platform for any kind of paid membership community which can clearly define two sides of an interaction or market. This paper presents the elements of the platform: the Ethereum smart contracts which handle the trust-sensitive ownership and reward structures, as well as the core game logic which makes it work. Finally we lay out a two year roadmap for launching and scaling the network, including a detailed specification of the Guppy token (GUP), which will play an integral role in its development.

Contents

1 Introduction

Most dating apps on the market today suffer from two core flaws. Firstly, there is a tendency to rely on dehumanizing profiling strategies, to the exclusion of more effective forms of matching. Secondly, eager for subscription money, many apps often allow huge demographic imbalances to occur to the detriment of user experience. In a study by the GlobalWebIndex it was estimated that over 62% of location-based dating apps such as Tinder are used by men. Where Tinder is concerned, the abundance of men and the scarcity of women, mixed with a lack of real life social cues creates an environment that is neither fun nor ultimately successful for most users.

Matchpool solves these game-theoretic problems by allowing users to create pools in their area with custom demographic specifications. Pools can be visualized as a cross between Slack channels and Meetup groups- federated, invite-only groups owned and run by devoted matchmakers. In this paper we will describe the process by which matchmakers are rewarded in the network's internal token for matching new entrants successfully. The quality or reputation of particular pools is measured against whether the joiner's particular goals have been met.

What makes Matchpool unique is the matchlock feature, which keeps a 50:50 ratio between x and y. If a certain number of x users enters the pool, then the matchlock will stop more x from entering until the same number of y's arrive. In a dating context this could apply to gender, but the it can also be parameterized to any use case where any arbitrary market polarity is required. Also there is no restriction for the x and y side of the matchlock to be opposites. For entrepreneurial networking or any other use case the x and y can be identical (or completely unset). For LGBT dating pools the "gender" configuration of both sides could be the same. It is up to the pool founder to configure the matchlock according to the kind of community they wish to create.

2 Strategies

Social networks have two strategies for connecting users: profiling and pooling.

Before the internet, a typical profiling strategy for social networking would be something like a "classified" in the local paper. This might be effective for advertising products or vacancies where both sides are clear about what they want. A good current example of where this approach works well is the job marketplace Indeed.com. Some users advertise what they are looking for, and other users compete to meet that demand.

Then there's Twitter, at the other extreme. On Twitter, users have a profile for sure, but most activity emerges spontaneously from human interaction. This is an example of a social network geared primarily towards pooling. Pooling strategies applied to dating in the real world include things like bars, nightclubs, dancing, as well deliberate forms of matchmaking like speed-dating. All of these activities have an underlying element of ritual distraction, which people need to get over themselves. This important element has been recognized and practiced throughout human history, but is missing from more atomized online approaches.

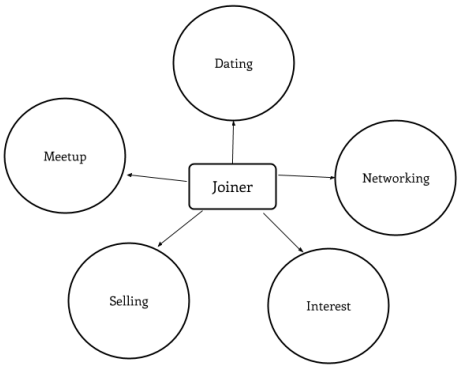
Matchpool implements a combination of the two. Joiners do have profiles, but only for sorting into pools, after which they can participate in pool chats, play dapplets and go to events (organized by the matchmaker for the group as a whole) or dates with other pool members. The basic philosophy of pools is to creating the right environment and then letting nature take its course.

While pools have a direct interest in creating as many matches as possible, we assume singles are already motivated. Once inside a pool, there is little unnecessary mediation. By creating an intentional high trust community, there is a subtle pressure to behave better and not engage in spamming or low-quality mass messaging. There is a tendency in blockchain development and systems design to lean heavily on boilerplate notions of "reputation" and "incentives", while ignoring the fact that reputation is a complex, emergent property of other people's subjective impressions and market decisions. Groups allow for this kind of unmeasured (and unmeasurable) "reputation" to emerge among their members without the need for some contrived score or karma count. Quantified reputation only relates to the quality of the pool as a whole, not to individual participants.

3 Participants

There are two kinds of participants on Matchpool: joiners and hosts.

3.1 Joiners



Joiners are a self-explanatory category: anyone who want to join a pool with the intention of meeting someone that matches their particular interest. Joiners have the following rough user journey:

1. User supplies a number of "factors", or details about themselves: name, age, gender, interests, personality type, sexual orientation, seeking relationship type, education level, income, ethnicity, religion, fitness, etc. This data is confidential and encrypted, and only used for the purpose of matching people into pools.

2. User creates a wallet and has the option to buy Guppies, either to exchange it from crypto or to purchase it directly with fiat using their credit card. This is for the purpose of paying the entry fee into pools.

3. User can see the various pools which exist within geographical proximity to their location. Depending on the values they have provided, they will see pools they are qualified for.

4. User enters the pool of their choice pays the requisite entry fee (if applicable to the business model of the pool and other policies which can be put in place).

5. New entrants have a higher reward rate for matchmakers to pair them up. During this initial period dateseekers will likely be matched with several people.

6. When users go on a date they is published on their “notebook”, which is a public log of activities. The process of logging an event involves agreement with the other person before it appears.

7. Users can go to events, message other dateseekers, play dapplets and go on dates until they are in a relationship, at which point their journey is complete. They have the option to remain a member of the pool and act as a matchmaker for other dateseekers (thereby earning divs) or leave.

Joiners are billed by the pool monthly for membership. All billing in Matchpool is denominated in the network’s own token, Guppies (GUP). Joiners can act as "helpers" to the pool host, for example as a matchmaker, admin or greeter.

3.2 Hosts

The host is the founder of the pool and sets in place the specifications regarding who can become a member. A helper on the other hand can be any member within the pool who acts as a matchmaker for other members. Helpers propose matches between two other joiners who they believe would be good partners. Depending on the kind of proposed match- a date, a relationship, a marriage- the helper will get a different number of arrows. **Table 1** shows a sample breakdown of the reward system.

This is not the exact number; this should be found with trial and error. The process for all of these “matches” is in 4 steps.

Action	Value
Date	1 arrow
Meeting	1 arrow
Relationship	20 arrows
Engagement	150 arrows

Table 1: Action-to-value list, configured by host.

1. Anyone can start a pitch involving two other users with an event category: Bob, Alice - Date.
2. Bob and Alice are notified, and can accept or reject.
3. If they both accept, the event is added to their public notebook, which is a log of activity.
4. The proposer earns 1 arrow.

The trust model of this interaction assumes a few things. Since a) both Bob and Alice are paying for membership of the pool and b) their identities have been verified, they have no incentive to accept the match fraudulently. The event is also posted to both of their public notebooks (similar to a Facebook relationship status update). In the Matchpool app there will be all the functionality for scheduling and arranging events as well as "Stories" after the event has happened.

Use section and subsections to organize your document. Simply use the section and subsection buttons in the toolbar to create them, and we'll handle all the formatting and numbering automatically.

Arrows first of all entitle the matchmaker to a certain share of divs in the pool. Divs entitle the owner to a certain share of the monthly subscription revenue.

Once matchmakers has collected a certain number of arrows, they have the ability to start new pools.

4 Pools

Love is composed of a single soul inhabiting two bodies. - Aristotle

4.1 Overview

A pool is the basic unit in Matchpool, which features a Slack-like public chat interface as well as the ability to send private messages along with other features including a wallet. Pools are established by matchmakers in two ways. The first way is to reserve in escrow a certain number of Matchpool tokens ("guppies") which is held in the pool for the duration that it is open. The second method is simply that users who have achieved a certain number of arrows from matching people in other pools. Since successful matchmaking is a vital component of the public reputation of pools and the network as a whole, it is important that new pools should be started by people who have a successful track record.

4.2 Matchlock

The matchlock allows the host to configure the demographic details of two sides of the interaction- X and Y. When starting a pool, the host is able to configure the matchlock with the required factors for each side of the interaction, including gender, age-range, education level, ethnicity, religion and geographical proximity. This can be as broad and detailed or generic.

When a certain number of type X enters, the pool becomes unavailable to users of type X until the same number of type Y enters. One of the issues with the matchlock is that people may be compelled to change their details in order to get around the matchlock, however we will develop a number of mechanisms in the platform which will allow for peer authentication and dispute of factors, including social media verification.

The population limit is set to 144 by default, which is approximately Dunbar's Number- the number of personal relationships that the human mind can handle. But the host can also configure this parameter if a larger or smaller limit is desired.

Table 2 is an example of how the host can configure the matchlock, with each field having its own set of valid factors. Users who are registered on the protocol with that list of factors can then be matched to their group of choice, as long as the matchlock is not activated due to demographic imbalance.

X	Y
Female	Male
21-35	21-35
Some College	Some College
0-20km	0-10km
10 GUP	20 GUP

Table 2: A list of example demographic configurations for the matchlock

4.3 Finding Pools

Matchpool will implement a front-end interface which will allow users to search for pools in different areas, matched with different interests. In this respect it will be somewhat similar to Meetup.com; users can search various factors to see if something fits. In the on-boarding process they will also be shown a list of pools in their area which fits their particular demographics and interests.

4.4 Fee Options

The matchlock configuration also defines the entry fee in the protocol's internal currency for different groups as well as the ongoing monthly subscription fee. In the example of a dating pool, this is important because there may be a great deal of variation between demand coming from males than females; in which case it would make sense for the price to be more expensive for one and cheaper (or even free) for others. The lock will prevent any demographic imbalances from occurring, however, when a user for whatever reason is unable to enter a locked pool, they can "register interest" in joining. Based on these interest indicators, the host will be able to see the demand level and adjust the price accordingly.

4.5 Notebooks

The notebook is a kind of log which all Matchpool users have. This log updates events such as dates, relationship status update and "stories". Whenever a user goes on a date they have the option to share a story of how it went, which is added to their public notebook.

4.6 Dapplets

Dapplets are add-ons created by developers who want add to the Matchpool protocol. In practice Dapplets work similar to Slack plugins but can enhance the experience in a number of ways. One example would be a dapplet which implements "Tinder" functionality: users can swipe through other users.

A good example of where this works well is in sites like MeetMe.com, which had a suite of mini-games like "Lunch Money", "Blind Date" etc. which helped users meet and get to know each other.

Dapplet developers are rewarded similarly to Matchmakers: if one of their apps creates a match of some kind, the dapplet developer earns arrows which translate into divs. In the same way that an individual within a group can pitch a match between x and y, so can a game. For example if a "Blind Date" game creates a real match, then the dapplet developer will earn the same number of arrows and be rewarded via the same process as manual social match within the pool.

We will develop a dapplet marketplace so that pools can continuously incorporate new functionality.

5 Ownership Logic

5.1 Arrows

Joiners pay a certain fee for continual membership of pools, which is payable monthly. This payment is made to host. The host can operate as a matchmaker themselves within the pool, but other members can also act as helpers, as mentioned earlier. Whenever a matchmaker achieves a successful pitch- defined by whether two people mutually update their Matchpool notebooks to reflect the event- they are awarded "arrows". The host, as the owner and founder of the pool automatically has a share. An automatic share can be allocated to cupids too, but for the most part this is a breakdown of what this might look like.

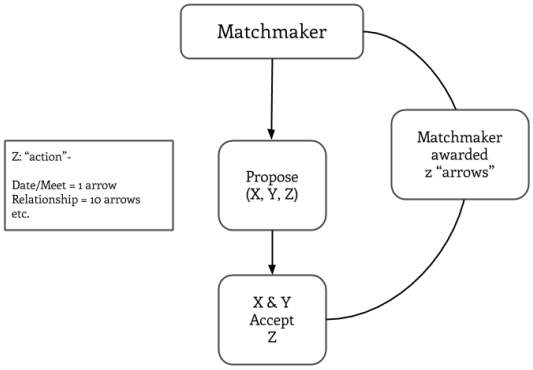


Figure 1: The process of Pitching.

5.2 Divs

Over every epoch a pool makes a certain amount of revenue, based on member subscriptions. Divs decide who owns this revenue. If a pool makes 2000 GUP over a given epoch, and a user holds at 50% of the divs in that pool, they will receive 1000 GUP at the end of the epoch. Until that point, the funds are held in escrow in the smart contract and cannot be moved.

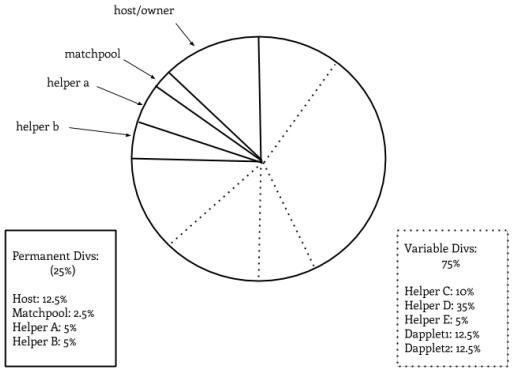


Figure 2: Perms and Vars denominated as one.

There are two categories of divs: **permanent divs** and **variable divs**. Variable divs are calculated using the following formula:

$$\frac{totalrevenue * tempdivs}{100} * \frac{earnedarrows}{totalarrows}$$

[!h]

This follows a few steps.

1. Get the total revenue (intake of Guppies) then find the amount that is allocated to variable divs.
2. Find the users divs by calculating the number of arrows accumulated as a percentage of the total arrows created during that epoch.
3. Multiply them to find the amount due to the user.

Permanent divs are pre-determined by the owner/owners of the pool, who can choose to grant divs to anyone. The share left over from permanent divs are given over to helpers who match other users on the platform, as described earlier in this paper. This is a breakdown of permanent and temporary divs considered seperately.

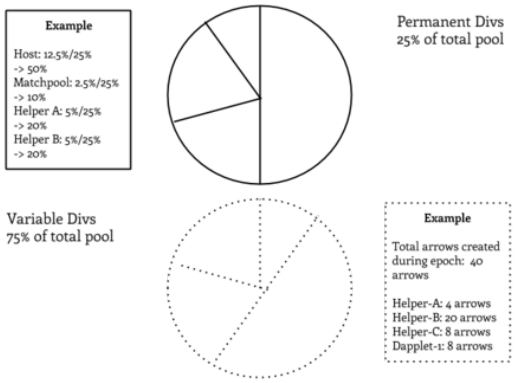


Figure 3: Perms and Vars considered separately.

5.3 Other Business Models

There are basically three business models available to hosts:

- Entry fee
- Subscription
- Packages
- Pay on delivery

Some pools may want to have alternate business models, for example to implement an escrow contract that pays half initially and half on delivery. We will facilitate these business models by implementing them in the form of dapplets: the monthly subscription facility for membership will itself be a dapplet, while a single membership with escrow will be another option. When Matchpool launches we will provide a basic suite of dapplets that hosts can configure while opening up the API to allow developers to contribute custom models.

6 Privacy

The factors that define a user's demographics are private by default, and only revealed to other users on the platform with the users explicit consent. In the last few years there has been a struggle between the interests of internet users and the interests of governments on the topic of strong cryptography. Matchpool will be incorporated in a territory which will best allow for complete privacy for users. We will also design the platform in such a way as to avoid holding centralized databases. This is important to avoiding hacking of users data.

One of the potential pitfalls is that people will use the network for socially undesirable purposes such as prostitution or selling drugs. At the end of the day Matchpool cannot control exactly how people use it. What we can do is provide the software and implement tools so that people are able to distinguish good pools from bad. One such feature is that all pools will be equipped with a

public profile so that people can review and share stories about their experience within it. If pools are used for socially undesirable purposes then it will become quickly apparent. Ultimately it is up to the community, which has a stake in its success, to create and join environments which reflects their values.

7 Trust Policy

7.1 Centralization vs Decentralization

One of the major themes of the crypto community has been on the importance of the concept of decentralization. The origin of this theme was Satoshi Nakamoto's conception of Bitcoin as a "trustless" currency- which doesn't rely on a particular authority to guarantee the truth of its record. In the context of apps, decentralization is not an unlimited good in and of itself.

Therefore trust policy of Matchpool breaks down into three categories:

- Decentralized.
- Federated.
- Centralized.

The trust-sensitive aspects of the cryptocurrency and arrows are decentralized and can't be meddled with. This is also true for the core protocol.

However apps and platforms rely on continual development to stay competitive. Continual development requires developers to have permissioned access to update the software constantly. So the front-end will be centralized and under the control of Matchpool as an organization. Here is the breakdown:

Centralized	Federated	Decentralized
Front-end	Div distrdibution	Cryptocurrency
	Pool admission	Arrows
	Pool banning	Div payments
	Arrow rewards	

Table 3: Matchpool's trust policy

7.2 Contracts

The core Matchpool protocol which deals with the "rules" of the game are decentralized using the Ethereum Virtual Machine (EVM). Ethereum is a programmable blockchain which can run small units of code called smart contracts. Blockchains have become something of a buzzword so it's a good idea to explain what they are and what they're useful for- and more importantly, what they're not useful for. A "blockchain" in general is a peer-to-peer network where every member (node) holds the same copy of a shared record or database, which can't be defrauded. In the case of Bitcoin, this shared record behaves as a simple ledger of addresses and balances of Bitcoin currency.

Blockchains are highly useful for trust-sensitive use cases where having a particular institution controlling it would lead to conflicts of interest where there is a risk of exploiting trust counter to users. In proof-of-work blockchains, new entries can only be added to the record by solving mathematical puzzles and providing cryptographic proof that the node has solved the puzzle. When this happens, the node is allowed to add a certain number of transactions to a "block", which is appended to the blockchain, making the record one block higher. Ethereum adds to the core functionality of ledger-based blockchains by adding in a "Turing-Complete" scripting language called Solidity which can compile directly into EVM opcodes. Turing-completene0ss is another buzzword which basically means a language where any imaginable problem could theoretically be expressed and solved, although it should be noted that in Ethereum there is a limit to this complexity imposed by network charges, expressed in gas.

8 Guppies

8.1 The Purpose of Having a Token

Why not simply denominate all interactions on the platform in terms of Ether or some other more well known cryptocurrency, instead of creating an appcoin from scratch? This is a fair point: there are many tokens out there which can be somewhat confusing to navigate. There are three main reasons for adopting the appcoin strategy:

1. To have a way to raise enough money to be able to develop the platform. Well-implemented crowdsales with carefully defined rights, targets and accountability structures are increasingly the best way for crypto projects to both find capital and to build a community that has a stake in its success.

2. To have a way to reward new users for joining the platform itself.

3. To provide a simple and universal measure of value across every pool.

4. To facilitate opening pools and paying subscriptions.

At the beginning, the crowdsale is an investment in Matchpool as an idea and a commitment to see its fruition. GUP is embedded in the platform on every level as the means of denominating fees and divs. Secondly one of the most essential aspects of any community-based app is being able to encourage people to join. A significant portion of the token will be reserved for the purpose of rewarding a limited number of new users in the platform at the beginning: the number will be capped to about 250,000 users. This is a similar approach that PayPal took in the early days of its existence, where new customers were awarded $20. If a pool has a fee set at 10 GUP per month, and we award new users 20 GUP (for example), then the user can be a member of the pool for two months before they will have to buy Guppies to continue being a member. If Matchpool were to use Ether as a medium of exchange this would not be possible.

8.2 Token Specification

Matchpool's platform token, the Guppy (GUP) will initially be distributed in the form of a pre-sale. Participants may acquire 1 GUP at a discounted rate by pledging a defined sum of Ether (ETH) into the token sale smart contract. The contract will define withdrawal policy for the duration of the crowdsale as well as other rules around multisignature spending and milestones.

For users coming from other currencies It will be possible to use third party conversion services like Shapeshift or Kraken to acquire Ether for the purpose of buying GUP.

The total duration of the crowdfund will be 28 days.

The first hour of the crowdsale will be Power Hour. During this period, 1 Ether will buy 110 GUP. After the first hour, the ratio will be 90:1 for the first three days. After three days it will change to 85:1 and after two weeks it will be 80:1 until close.

The crowdsale will be capped at $4.2 million USD, in terms of Ether. As soon as this amount is reached, the smart contract will stop accepting funds. At the end of the four-week period, token transfers will be locked for two months.

The total supply will be 100,000,000 GUP, with the smallest available denomination being 1000 mill per Guppy.

The following table below shows the overall specification and breakdown of the token.

Description	Amount
Total supply	100,000,000 GUP
Minimum denomination	1000 mill
Sold during crowdsale	60%
New user incentive	20%
Team share	5%
Advisory board	5%
Early stage investors	5%

8.3 Price Steps

In total, 60 million tokens will be sold during the crowdsale. The total amount that will exist at the beginning will be 80 million.

Time	Exchange rate
First hour	110 GUP for 1 ETH
First 3 days	90 GUP for 1 ETH
First 2 weeks	85 GUP for 1 ETH

Investors will be able to withdraw their investment but only at the defined steps. For example, if an investor bought 110 GUP for 1 ETH during Power Hour, if they try to withdraw after Week 2, they will get back the Ether equivalent of 80 GUP.

To ensure that incentives are aligned properly, we will set list of milestones in the smart contract, for which an external advisor (or multiple advisors) will have voting rights. This will be for releasing funds from the core contract. It will also be the case that the core team will be unable to sell their own tokens until a number of milestones have been reached or a certain timeframe has passed (for example- after 1 year). This is to avoid any sudden "dumps" which could potentially upset the price. It also means incentives are aligned for long term commitment to building Matchpool as a viable platform.

9 Roadmap

The roadmap for the launch of Matchpool.

Date	Target
December 2016	Plan out the platform and hone vision
January 2017	Start working on the MVP
Feburary 2017	Prepare for crowdsale
March 2017	Launch crowdsale and experimental pool in Paris
August 2017	Release production platform worldwide
End of year	Distribute all referral tokens (250K users)

Table 4: A roadmap for the development of the platform

10 Appendix

```
/// @title Guppy: the official Matchpool token
/// @author Philip Saunders <philip@pax.directory>
/// @dev Ropsten address: 0xEC1af65243527F0D5683bfb3E30B0C05a96a3273
 // ABI on IPFS:
 // https://ipfs.io/ipfs/QmNgCSA4DydqMEJpZxYZjCbtJ3DmPKDtVx8oKydYAkKq54


pragma solidity ^0.4.6;


contract Guppy {

    string public name = "Guppy";
    string public ticker = "GUP";

    uint256 public denom;

    uint256 public totalSupply;
    uint256 public availableSupply;
```

```
    uint256 public icoSupply;
    uint256 public referralSupply;

    uint public mill2Wei;

    uint256 public epoch;
    uint256 last;

    address[] public core;

//   "deploy tokens" allow members vote for one member to deploy a
//   particular function.

    mapping(address => uint256) public balances;

    mapping(address => mapping(address => uint256)) cheques;

mapping(bytes4 => mapping(address => uint256)) functions;

    modifier system {
            bool found;
            for(uint i = 0; i < core.length; i++) {
                if(core[i] == msg.sender){
                    found = true;
                }
            }
            if(!found) throw;
            _;
    }

    modifier interval {
            if((now - last) > epoch) throw;
            _;
    }

    modifier limited(bytes4 _deploy) {
            if(functions[_deploy][msg.sender] < (core.length - 1)) throw;
            _;
    }

    event Transfer(address _from, address _to, uint _value);
    event Mint(address _owner, uint _amount);
    event ChequeWrite(address _from, address _to, uint256 _value);

///  @dev the constructor


    function Guppy(uint256 _totalSupply,
                    uint256 _bonusSupply,
                    uint256 _coreSupply,
                    uint256 _icoSupply,
                    uint256 _denom,
                    uint256 _epoch,
                    address _core) {
// 1000- mill: the multiple in which guppies are denominated
        denom = _denom;
        last = now;
        epoch = _epoch;
```

```
// 10500000: the total supply
        totalSupply = denominate(_totalSupply);

// availableSupply
        availableSupply = denominate(_coreSupply + _icoSupply);

// supply available in the ico
icoSupply = denominate(_icoSupply);

// supply available for user referrals
        referralSupply = denominate(_bonusSupply);
        core.push(_core);

//  For every 1 ether, 10 Guppies. mill2Wei: 1 mill = 10^14 wei
        mill2Wei = 100000000000000 wei;

/* The full core supply is given to the first member of the
   core group, to transfer to others according to share */
        balances[core[0]] = denominate(_coreSupply);
}

    function denominate(uint256 _guppies) constant returns(uint256) {
            return _guppies * denom;
    }


     function millToGuppies(uint256 _mill) constant returns(uint256) {
            return _mill / denom;
    }

    function transfer(address to, uint256 value) returns (bool) {

            if(balances[msg.sender] >= value && value > 0) {
                  balances[msg.sender] -= value;
                balances[to]         += value;

                Transfer(msg.sender, to, value);
                return true;
            } else {
                return false;
            }
    }

    function createCheque(address recipient, uint256 value) returns (bool) {
     if(balances[msg.sender] >= value && value > 0) {
     cheques[msg.sender][recipient] += value;
     ChequeWrite(msg.sender, recipient, value);
     return true;
     } else {
     return false;
     }
    }

    function getChequeValue(address origin, address recipient) constant returns(uint256) {
     return cheques[origin][recipient];
    }
```

```
function claimCheque(address origin, uint256 value) returns(bool) {
 if(cheques[origin][msg.sender] > value
 && value > 0
 && (balances[origin] - value) > 0) {
 cheques[origin][msg.sender] -= value;
 balances[origin] -= value;
 balances[msg.sender] += value;
 return true;
 } else {
 return false;
 }
}

function withdraw(uint256 amount) interval returns(bool){
 if((balances[msg.sender] * mill2Wei) < amount){
 return false;
 }
 msg.sender.send(amount);
 return true;
}

function mint() payable {
        uint _mint = msg.value / mill2Wei;

        if(_mint > icoSupply)
            throw;
        balances[msg.sender] += _mint;
        icoSupply -= _mint;
        availableSupply += _mint;
        Mint(msg.sender, _mint);
}

function referral(uint _guppies, address _user) system {
        if(_guppies > referralSupply)
            throw;
        referralSupply -= denominate(_guppies);
        balances[_user] += denominate(_guppies);
        availableSupply += denominate(_guppies);
}


/* ADMIN METHODS */

function increaseSupply(bytes4 _sig, uint _amount)
                                    limited(_sig)
                                    system
                                    interval {
        totalSupply += denominate(_amount);
        last = now;
}

function newIcoSupply(bytes4 _sig, uint _amount)
                                    limited(_sig)
                                    system
                                    interval {
        if(_amount > (totalSupply - availableSupply))
            throw;
        icoSupply += _amount;
```

```
    }

    function newEpoch(bytes4 _sig, uint span) system limited(_sig) {
     epoch = span;
     last = now;
    }

    function addcore(bytes4 _sig, address _core)
                                        limited(_sig)
                                        system {
            core.push(_core);
    }

    function removecore(bytes4 _sig, address _core)
                                        limited(_sig)
                                        system {
            for(uint i = 0; i < core.length; i++) {
                if(core[i] == _core) {
                    delete core[i];
                    break;
                }
            }
    }

    function changeSupplyPrice(bytes4 _sig, uint _price)
                                        limited(_sig)
                                        system
                                        interval {
            mill2Wei = _price;
            last = now;
    }

    function corewithdraw(bytes4 _sig, uint _amount, address _to) limited(_sig)
system interval returns(bool) {
            if(this.balance > _amount) {
                return _to.send(_amount);
            } else {
                return false;
            }
    }

    function corecall(bytes4 _sig, bytes _args)
limited(_sig) system returns(bool) {
            if(this.call(_sig, _args)) {
                functions[_sig][msg.sender] = 0;
                return true;
            } else {
                return false;
            }
    }

    function coredelegate(bytes4 _sig, address _core) system returns(bool){
            if(functions[_sig][msg.sender] > 0) {
                functions[_sig][msg.sender] = 0;
                functions[_sig][msg.sender] += 1;
                return true;
            } else {
                return false;
```

```
            }
    }

    function corepropose(string _funcsignature) system {
            bytes4 sig = bytes4(sha3(_funcsignature));
            for(uint i = 0; i < core.length; i++) {
                functions[sig][core[i]] = 1;
            }
    }

    function() {
            msg.sender.send(msg.value);
    }

}
```