Concepts for a Blockchain-Based Gaming Platform

Keir Finlow-Bates

keir@chainfrog.com

18 January, 2019

Abstract

Blockchains are potentially useful for in-game currency and player identity, but are capable of providing far more than that. In the following paper I present some further concepts and specifications for implementing games enabled by blockchain.

Contents

Introduction

I start with a brief discussion of some of the issues and problems that are faced in connecting a blockchain to a game, due to the limitations and restrictions that blockchains impose on programming systems, namely:

- latency
- block confirmations
- permissioned and open blockchains
- decentralization and surrendering control

Then I examine the concept of an asset bank node, with which the game provider can manage the blockchain system to prevent gamer abuse of the system, without having to assume a totalitarian dictatorial role.

Subsequently I present examples of how the virtual objects and concepts that blockchain enables can be adapted to provide significant value and improved engagement to the gaming experience, including character definition, how consensus systems can allow character initiated “mining”, mapping entities in the blockchain to virtual space, as well as the standard and obvious use of cryptocurrency for in-game money.

In the conclusion I summarize how blockchain can add these further dimensions to the online multiplayer world.

Finally, there is a glossary defining the various terms used in this paper.

1 Latency and Confirmation Issues

At first view, it may seem unfeasible to match the slow rate of blockchain updates with the rapid rate at which game activity can progress, therefore only making blockchain suitable for recording assets after a game is over, for the authentication process of logging in, or for slow turn-based games like chess. However, this does not need to be the case.

As blockchain blocks are added to the chain at a set rate (for example one block every ten seconds), waiting for data in the blocks to be confirmed causes a delay in re-rendering the current game view. This is the blockchain latency issue.

This is further compounded by partitioning in the blockchain network and the rebasing of the blockchain that is required when a disconnected partition reconnects. A losing chain segment is scrapped and replaced with a winning chain segment. This is the confirmation issue.

In order to discuss latency and confirmation, it is important to consider the two basic types of game state:

- Ephemeral game states, which are soon forgotten, even if something went wrong. These include things like the exact character position or distant object rendering.
- Essential game states, which will aggravate players and cause them to quit if they are not accurately tracked. These include things like character level, assets gained, amount of gold owned and other records of personal achievement.

The upshot is that players complain about glitches, but they quit over personal loss.

Therefore it is worth noting that transactions on the blockchain network can be seen almost as soon as they are submitted, even if they have not yet been packaged into a block and recorded on the canonical blockchain hash-linked list.

As a result, almost instantaneous events related to ephemeral game states can be acted on by the game engine when the relevant transaction is first observed. For example, a character changes their name and submits a transaction to record this. The name label for the character can immediately be updated. If later it turns out the transaction is not accepted, the name can be reverted with no real loss to game play.

Essential game states, on the other hand, should only be acted on when the relevant transaction is packaged into a block and there have been a number of subsequent block confirmations. For example, a character transfers 100 gold coins to a second character. The second character should not be able to spend that gold until it is confirmed, as it is difficult to roll back actions taken with the coins. Similarly an experience point raise should not be displayed until it is locked down in the blockchain.

In summary: a well-structured game will update ephemeral states almost instantly based on transactions, and essential states after a suitable confirmation period.

2 Permissioned versus Open

For blockchain to have a valid place in a game, the game designers have to surrender some of the control and rights that game companies have traditionally reserved for themselves, by not being the sole administrators of the servers on which the game runs and the databases that hold player and game data. In blockchain there is a spectrum of openness, based on the level of permission required to submit certain actions to the blockchain. The key permissions include the right to:

- Admit other addresses to the blockchain
- Create new addresses
- Create digital assets

- Issue more of a digital asset
- Submit digital asset transfers (transactions)
- Create a data stream
- Read from a data stream
- Write to a data stream
- Grant one or more of the above permissions to an address
- Revoke one or more of the above permissions from an address
- Run a data re-transmission node on the blockchain
- Run a block-generating node on the blockchain

As a blockchain-based game designer you have to ask yourself which of the above rights you are willing to cede to your players, and which you want to keep for the game administrator. In general, the more rights you are able to pass on to your players, the more valuable your blockchain and hence your game will be.

The Asset Bank

One approach is for a master node run by the game supplier to run an asset bank to set the structure of the blockchain and ensure a suitable supply of assets.

For example, the asset bank may issue a digital asset for experience points (XP) and a digital asset for currency (CR). The asset bank transfers these assets to players as they earn them, and can issue more of both as supplies run low. The asset bank therefore has the role of central bank.

In our example, the game may allow players to transfer CR to other players, but may restrict players from transferring XP.

A Possible Architecture

In the diagram below, an exemplary game system architecture is presented. A game server provides low latency communication between game engines, with the game blockchain enabling digital assets and further blockchain functionality.

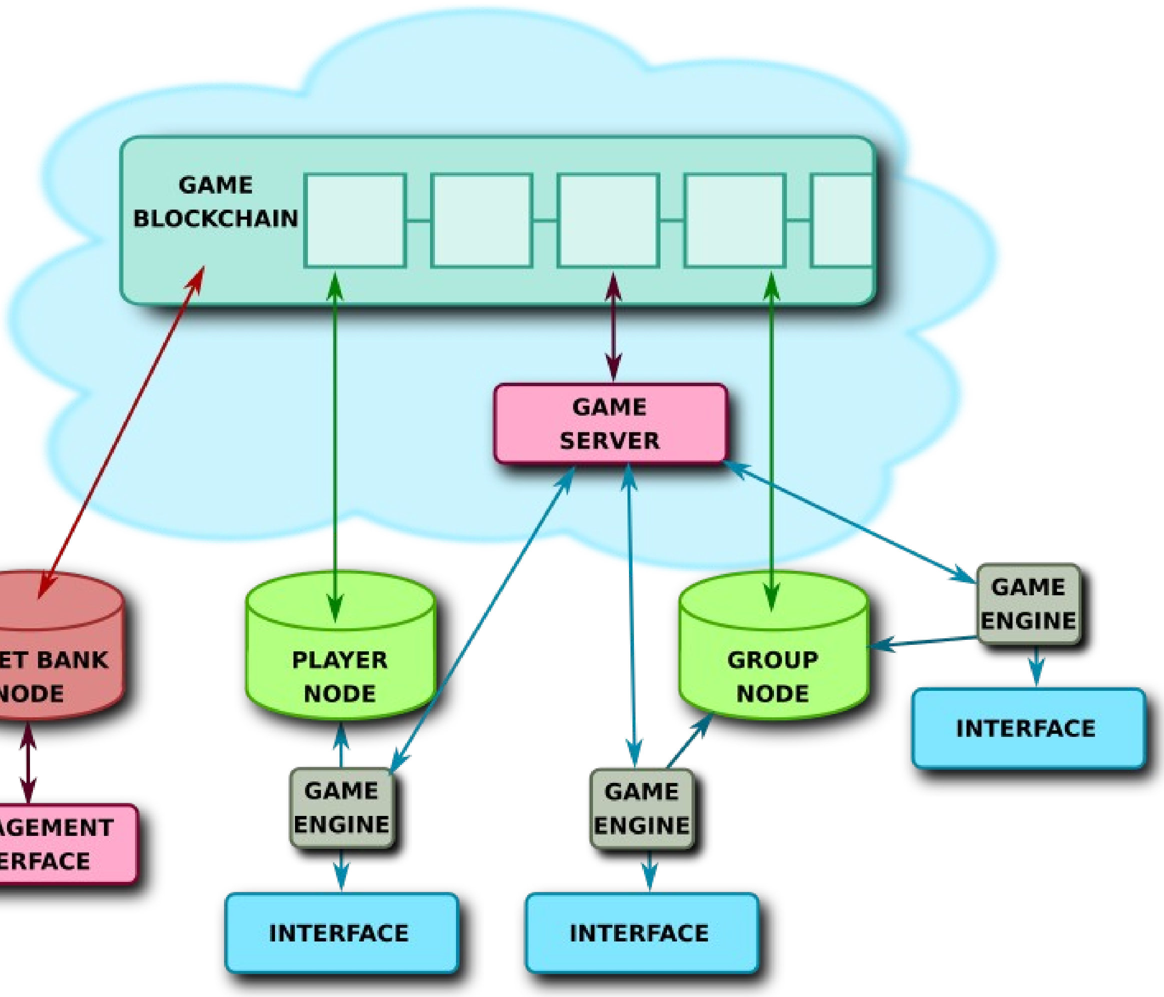
Game players can either run their own node on the blockchain, or connect to a shared group node.

The interface receives game data and renders the game, and passes back instructions from the player.

The asset bank note is a separate master node which handles issuing of digital assets and player authentication.

The game blockchain, as well as providing a tamper proof record of player character identities and asset ownership, can allow for linking assets and ownership of areas within the game.

All these parts are discussed in further detail below.



3 Mining Space

Typically a game involves a game field. In a computer game this is usually a two-dimensional or three-dimensional space representing the playing area. Blockchain allows for the concept of decentralized ownership, for example by using a non-fungible token to represent each area. However, further possibilities are available that do not involve allocating a token to each location.

For example, consider a plane (2-dimensional space), divided into a grid. Each square with the grid has a unique identifier in terms of its position on the X and Y axis, and permits mined ownership, either through a modified proof-of-work or proof-of-stake algorithm. The following example should be illustrative, and uses proof-of-work:

The aim is to produce a payload plus nonce that meets a proof-of-work difficulty, that is, a hash of the payload plus nonce specifies the square and satisfies some other restriction.

The player must announce in advance on the blockchain that they intend to mine in order to own a specific square (otherwise they could mine at random and hope to claim a good location that just pops out of the repeated hashing). For example, let us assume that the grid consists of 65535 by 65535 squares. In hexadecimal this is `0xFFFF` by `0xFFFF`.

The payload plus nonce, when hashed with the chosen hash function (in my examples I use BLAKE because there are no ASICs to assist in the proof-of-work effort) must therefore return a 32 byte number ending in `FFFFFFFF` in order to be considered successful.

Furthermore, the more zeroes the returned value starts with, the more successful the result is. This allows different players to take over a square by expending more effort on mining. Additional restrictions can be added in the take-over protocol, for example, by requiring the competing player to significantly exceed the previous successful mining by more leading zero bits, or similarly by requiring more leading zero bits if the square owner also owns neighboring squares.

The payload can be set to be a header or content from a previous recent block. This prevents secretly starting to mine a specific square for days or months and then announcing it. For example, the restriction can be that the header of a block that is at most five blocks before the mining announcement must be used as a payload for the mining success to be valid.

The announcement that a player is mining a specific square can be made public in the game interface to inform other players so they can take appropriate action.

It may be the case that finding a hash that ends in `FFFFFFFF` is too difficult for the gaming devices. Given that the specific square was announced beforehand, this can therefore be relaxed, for example only requiring that the hash ends in `FFXXFFXX`, where `X` is any hexadecimal digit.

Similarly if the difficulty level is too low, additional leading zero bits can be imposed. The difficulty can be adjusted dynamically in order to compensate for technological advances in computers, or even on an individual basis, player by player.

Obviously it is not compulsory that the lower bits are used for data specification, and the higher bits for difficulty, but it does clearly separate the two.

4 Identity

In a traditional blockchain such as Bitcoin, pseudo-anonymous identity is provided by addresses. Anyone can download wallet software and generate as many addresses as they like, and each address can be credited with a cryptocurrency balances through transfer transactions.

In a gaming environment an address can be used as a player account. In MultiChain an address can be credited with many different digital assets. As an example, a game may come with wallet software that generates an account address for each of a player's characters to present and manage the character profiles.

Character Representation Schemes

As an player account address is a hash of an ECDSA public key, there are two approaches that can be used to derive a character representation from the character. One is to use the pseudo-random data that the account address generates, and the other is to publish the public key on an open stream and to use the public key. For the examples in this paper we are going to use the 33 byte compressed key, but the address or a 65 byte uncompressed version would work equally well.

The public key and address are pseudo-random, in that you cannot predict in advance what digits or characters they are going to contain. As a result, random character appearances can be created by using various parts of the address to specify features and colours.

For example, start with a raw private key:

`283D01856115B7970B622EAA6DAFF2B9ECE30F1B66927592F6EA70325929102B`

Then generate the 33 byte compressed public key:

`0284E5235E299AF81EBE1653AC5F06B60E13A3A81F918018CBD10CE695095B3E24`

Use the last six digits as a hex representation of the color of the character:

`#5B3E24`

Use the seventh-to-last digit, let's call it D, to determine the number of sides the character has, using the formula 3 + x mod 4. For D = 9, this is `0x4`. So the character is a square.

Given the address above, the character ends up being a brown square:

Let's pick another address, and use the same character definition scheme. Start with a raw private key:

`94A7D6B0C89B2691768D318F0DDAB882F8937480B6825090480EF7139E21A15B`

The 33 byte compressed public key is:

`026BE1F6B8591DF613DADF207953B4A09E9AB0CB7053322B4BC10D76F73BB6ECAC`

So the color is:

`#B6ECAC`

And z is `0xB`, so 3 + z mod 4 is 6.

Hence this character is a pale green hexagon:

Green hexagons and brown squares may not look that exciting, but other parts of the address could be used to specify: gender, clothing, character class, eye and hair color, the structure of a spacecraft, or the color of segments of a worm. It all depends on the game you are designing.

Some colors may be reserved, for example for background colors and elements of the game. This can be overcome by either rejecting keys that produce such reserved colors or color ranges, by using modulo arithmetic to prevent their selection, or using a color lookup or translation table. A simple solution is to use dark colors for the background (RGB hex values less than `0x55`, for example) and bright colors for characters (RGB hex values greater than `0x99`).

If a player is not happy with the way their character looks, they can always scrap the key and generate a new one. There is even the possibility of transferring all the digital assets to the new key, if such assets have been amassed before the player decides to change character.

Achievements

Character achievements such as level, experience and skills can be represented by digital assets. As described previously, the asset bank creates a digital asset, and monitors a data stream on the blockchain to determine when a player has earned some of the assets. A transaction is then initiated by the asset bank.

A second approach is to implement smart contracts on the blockchain which automate the award of assets based on achievement. At the time of writing MultiChain does not support smart contracts and so the logic for them needs to be managed off-chain, but achievement related transfers are an excellent use case for smart contracts.

5 Token Usage

There are three fundamental token types that can be defined:

1. fungible divisible tokens, for example for representing currency or energy, that can be divided down into fractional amounts.
2. Fungible indivisible tokens, for example for experience points or ammunition, that cannot be divided into fractions.
3. Non-fungible tokens that are single and unique, for example to represent ownership of unique items in a game such as a vorpal sword or a specific space craft.

Traditional token usage is well-covered elsewhere and so I won't go into it in depth here.

6 Conclusion

Hopefully I have shown that blockchain can be used for much more than just in-game digital assets. In particular the pseudo-random nature of hash functions and asymmetric keys generated in blockchains provide data input for adding extra color to a game. To summarize:

- Tokens can be used for in-game currency, artifact representation, and player statistics, and allow players to conduct disintermediated trades.
- Randomness provided by keys and hash functions of blockchain data can be used to generate unpredictable yet repeatable game structures.
- The consensus system can be linked to game space or object ownership, and players can actually work to own objects or locations in the game.

Glossography

The following definitions are provided to clarify the different blockchain terms, in order to be able to talk about them in a consistent manner:

Address: a hash of a public key usually represented in a base58 encoding to turn it into a readable string, and to hide the public key until it is actually used.

Digital asset: a cryptocurrency representing objects or classes in the game. In MultiChain multiple assets can be defined and a given number of each asset can be instantiated. Assets may also be divisible into fractions. Some assets have a fixed quantity on issuance, others have the possibility of further issues if the supply is running low.

Node: a node is peer-to-peer software that receives transactions and re-transmits them to other nodes, maintains a copy of the blockchain and transmits blocks to other peer nodes. In MultiChain a node also maintains balance information for addresses that are being monitored (or “watched” in MultiChain terminology).

Non-fungible token: a token that is unique, and can therefore be used to identify a specific item in a game.

Nonce: number used only one. Look it up on Wikipedia.

Private key: a secret large number that allows the wallet in which the keys are held to sign transactions. Similar to a PIN code for your bank card. As Multichain uses ECDSA, this number is 32 bytes long.

Public key: a number derived from the private key (however the private key cannot be derived from the public key), and is represented either in uncompressed form (65 bytes) or compressed form (33 bytes). Similar to a credit card number.

Stream: data published on the blockchain with a stream identifier. As a result, if the blockchain is parsed for the stream identifier, the data (including timestamps for each entry) can be retrieved. You can imagine this as a channel in a chat program, but stored on a blockchain.

Token: an indivisible digital asset, i.e. a digital asset that cannot be divided down into fractional quantities.

Wallet: a collection of keys.