
V1.0
polygon labs
Blueprint
A Web3 games guide by Polygon Labs

Who is this document for?

Game developers, product managers, and software architects beginning their Web3 journey. The guide is structured for developers with Web2 game development experience, and intended to accelerate their learning and game development process.

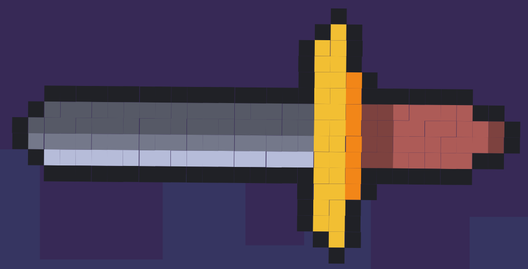
What can I expect to learn?

An overview of game systems and how to design blockchain integrations. You will receive a primer on the methods behind putting game services on-chain complete with considerations, advantages, drawbacks, and links to additional resources.

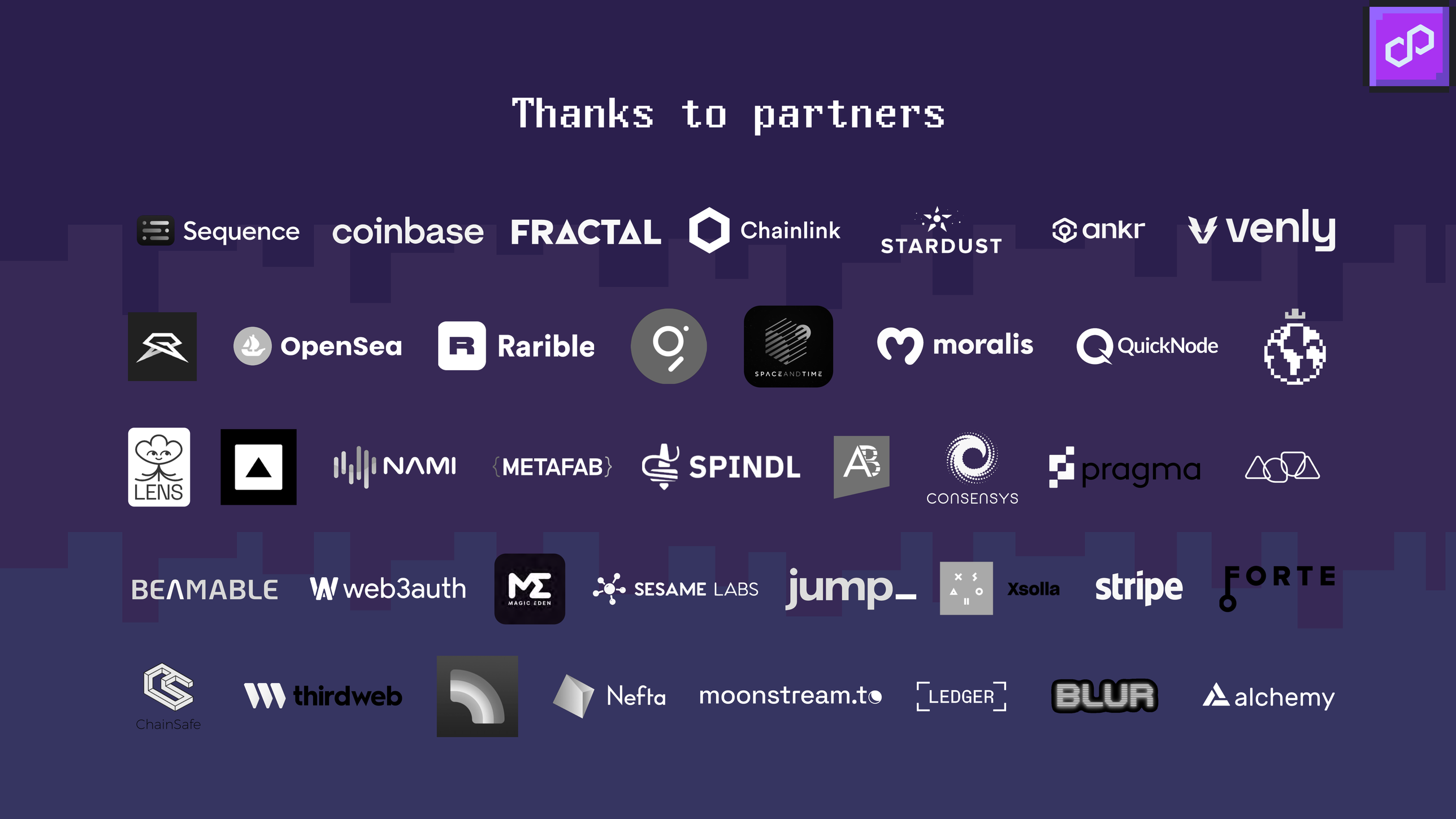
Thanks to: Will Matteson, Owen Wahlgren, Sameer Chauhan, Charnjit Bansi, Joshua Dhamanigi, Nirmal Ravisyam, Diana Chimes, and Michael Salk.

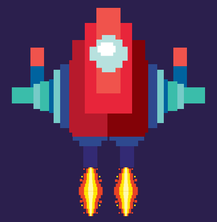
For questions, complaints, or to learn more about Polygon you can reach out to blueprint@polygon.technology or Will Matteson on Twitter @0xlegofan

It's dangerous to go alone! Take this.

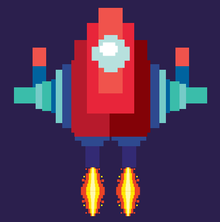


Thanks to partners





Welcome Player1!



You have unlocked the Web3 Continent — vast plains of digital ownership and snow-capped peaks of decentralization. In this realm, building in public is the norm, rules are just suggestions, and we're all gonna make it. You're at the forefront of game design, and maybe even the invention of an entirely new economy. But first, let's take a moment to unpack the objectives of this level.

Gaming is winning. With a population of over three billion gamers, and a $200 billion global market, what was a once cottage industry is now a sleeping giant. If software is eating the world, then games are getting a few bites at least. Games require talent from across creative disciplines: like storytelling, design, engineering – and yield uniquely engaging, and meaningful, ways to play.

Web3 has the same trajectory. The games industry briefly flirted with blockchains in the past, but when NFTs took the world by storm in 2021, Web3 gaming was thrust into the limelight.

Many of the "new" ideas for games are actually well-tread, and have been capturing the interest of gamers and developers for decades. MMORPGs and TCGs experimented with digital assets in the past (e.g. Diablo III auction house, WoTC), but didn't have the technology frameworks or consumer interest to succeed.

The first iteration of Web3 games showed promise but were limited in scale and scope. It's already hard to build games on top of vanilla data structures, and when you add in blockchains without established frameworks, tooling, and best practices – developers have had a tough time getting to market.

But it won't be like this for long. The Web3 Continent is expanding, and development continues at a torrid pace.

Player1, there is a way forward. Polygon Labs will show you the path. Blueprint will show you how to build on-chain-games, all the way up from the foundation.
Before we get started, do you have any questions, Player1?

Table of contents

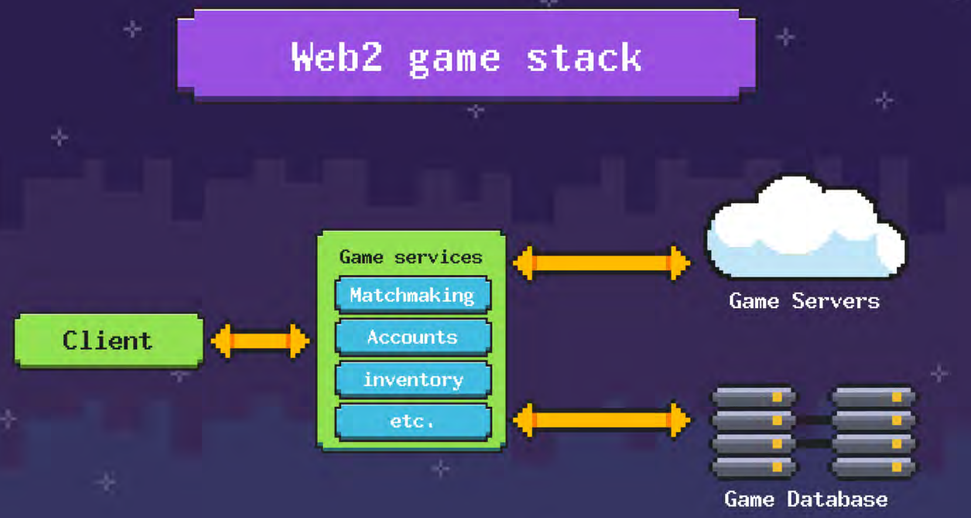
High level architecture

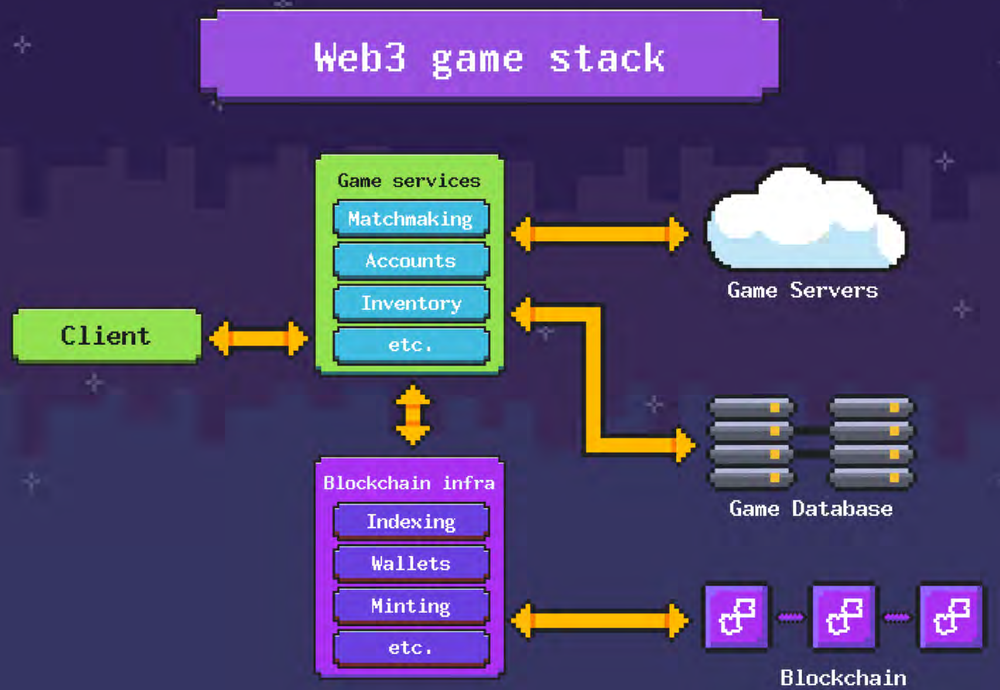
There isn't a shared definition of Web3 games – which is part of the problem. Here are some commonly understood definitions:

- Games with new P2P economic models and "real ownership"
- Games that use blockchains for cross-title interoperability
- Decentralized games that use blockchain consensus to accept community patches
- P2E and other variants of real money gaming
- And many, many more

We will adopt a purposefully broad and tautological definition: that Web3 games are just games that utilize blockchains for game features. What's consistent across 99% of Web3 games is that they'll need a way to verify player ownership (wallets), keep track of and parse on-chain events (indexing), and reflect game events on-chain (minting and metadata). Web2 game services are well documented, and there are all sorts of frameworks and back-ends-as-a-service that make it easy to create a game. With Web3, you will have to rebuild some (if not reconfigure all) game services to be able to interface with the blockchain.

But never fear, that's why we are here!





From a technical standpoint, you should think of blockchains as a new type of backend for games. The new functionality that Web3 unlocks (ownership, decentralized game building) are similarly often extensions of Web2 functionality (item leasing models, UGC, and so on).

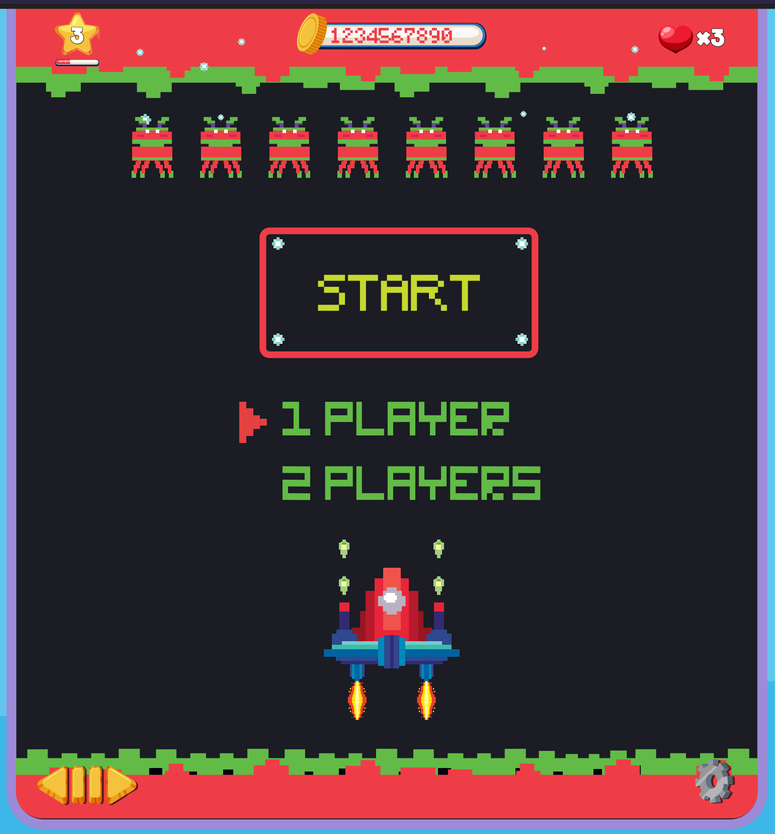
You can also think of the shift from Web2 -> Web3 gaming as similar to the progression from internal combustion -> electric cars. When you change the energy source of a vehicle (e.g. games backend), there will be other changes to make to car components and engineering tolerances (like game systems).

For example, you'll still need to deliver ample torque to the wheels, but instead of being generated by tiny explosions, it's through an electric motor, which impacts how the drivetrain needs to function. Similarly, using decentralized compute and storage can impact downstream game systems like player inventory, login, and more. Developers need to design and implement these systems with blockchains in mind. That being said, we're cryptographers and game developers, not automotive engineers, so don't follow our advice for building cars.

When starting out on your journey, it may be wise to first think about player experience and functionality, setting aside blockchain components for a moment to consider the experience you want to create. When that's done, you can add blockchain back in. Think of blockchains and token standards as a technology, an ends to a means (player experience or a product), rather than as the product itself. Many Web3 game developers struggle with overengineering the blockchain components, to the ultimate detriment of their development speed and ultimate player experience.

Processing design in this way mirrors the framework of this paper: start from the known / no-regrets game needs, and journey toward the unknown.

Onboarding



First time user experiences changes with Web3, due mostly to on-chain authorization and fiat on-ramps (covered later) . In all games, players need to be authenticated and brought into the game ecosystem by creating or importing a platform account. The authentication process can either be internal, enabling a username/password, or handled using social auth (OAuth) that links the social account with an internal identifier representing the player account. Player accounts encapsulate what is required to participate in a game ecosystem, as all other systems key off account identifiers. The fundamental concepts are the same in Web3, the building blocks (wallet vs Web2 credentials) are just different.

The de-facto standard now is frictionless auth, where registration is deferred until later. This can be done by creating a player account of their own accord, storing the access token on their device for use. Optionally, the token can also be preserved on the native cloud identity service of the device if available, such as iCloud for Apple devices. Account abstraction enables native session key functionality -- but we're still early on that front.

Wallets

Wallets are fundamental to blockchains. Players will need a wallet associated with their account. In simple terms, a wallet is the portal to managing digital objects and tokens on a blockchain. They are composed of a public-private key pair (just like a username and password); the user has the private keys, allowing them to use a wallet to manage digital objects and assets, while the public key is visible on the blockchain, and others can send transactions to a user. In other words, wallets are dApps built on top of this pairing. Authenticating or signing a transaction pushes it to the blockchain through RPC calls on a blockchain node.

A later section will cover game-specific wallets in depth.

Introducing blockchain elements to games means tackling this additional point of friction. From a UX perspective, game developers must imagine frictionless processes to help users create and securely manage their own wallets. Developers can integrate wallets into gaming in a number of ways:

- Automatically create and link a game-specific wallet for a player upon signup
- Ask players to link an existing external wallet to their account
- Create a new player account where the wallet is used as the primary method of authentication
- Manage game NFTs until the player wants to take control of them

These scenarios are not mutually exclusive. Different kinds of wallets can be deployed to achieve the desired results, but each comes with its own integration costs and tradeoffs. Integrating external wallets into account setup immediately lays the groundwork for digital objects and liquidity, but from a UX perspective, it is best to allow users to create a game account and start their gaming experience without requiring wallet setup. You should always try to minimize the amount of steps a player must take onboarding, and make sure that the flow is appropriate for your expected audience (e.g. Web3 noobs or Gamefi degens).

Launcher

Many traditional game launchers and publishers are placing restrictions on blockchain activity (e.g. Apple) or outright prohibiting them (e.g. Playstation). Adding to this problem, players are averse to downloading first party launchers and .exe files, as they are attack vectors for malicious activity.

Still, there are a number of game launchers available for getting your games into the hands of players, and more categories may come online soon. Here is what the landscape looks like right now:

- Web3 launchers: Web3 games only, likely NFT trading functionality emphasis
  - Fractal
  - Magic Eden
  - Aqua.xyz
  - Gamefi.org

- Web2 launchers: Built for Web2 games, no native NFT trading functionality
  - Epic Games Store
  - Steam (albeit with restrictions)

Access

Once account is authenticated, game studios may want to use an entitlements system to determine what players can access. Entitlement systems can be extended to include blockchain assets, specifically NFTs.

Beyond PFPs, NFTs can have significant functionality in game systems, paving the way for new design spaces. An NFT can represent access to map areas, digital objects, the game itself, really any arbitrary content.

From an implementation perspective, access is often a “simple” blockchain read (ideally server-sided). However, it is important to note that the rest of a player experience relies on the result of the call. For example, If an NFT is used to grant access to special features within the game, like a new game mode, developers need to account for the experience when access may be lost if the NFT is traded away. Checking for continuous ownership may be important, especially when NFTs have an impact on competitive play or rewards.

Know Your Customer (KYC)

Depending on your locality and game, users might be required to KYC. While we cannot provide legal advice about this, we can give you some tips and best practices. It is generally best to delay KYC or other frictional loops until they are necessary for the user. Since these user steps will induce churn, it is best to only make users KYC if/when they need to, instead of doing so preemptively.

You may need to KYC users for them to take assets out of the game ecosystem or take custody of those assets. In this scenario, you could use a standalone KYC provider. You might opt for one of the many Web2 options, or even explore using blockchain based identity solutions, e.g. the Polygon ID open-source toolkit, with Web3-native tooling that can support KYC attestations.

Most of the time, we have been approached about KYC vendors in the context of fiat on-ramps. Fiat on-ramps are covered in the “Store” section below, and most (if not all) on-ramps also have integrated KYC procedures.

Web3 UX

UX can make or break a gaming journey. When players interact with the game, whether buying content or claiming rewards, good game design executes requests without asking for additional input, and the outcome of an action is consistent with player expectation.
But as we noted above, introducing blockchain into a game inherently demands gas costs for transactions posted on-chain. Polygon Labs abstracts this payment by enabling meta-transactions, using relayers. Doing so gives developers more choice over who pays for the gas.

An overview of gas payments:

- Noncustodial – Only users
- SC wallet – Developer or the user
- Custodial – Developer or the user
- Relayer – Developer or the user

With account abstraction live (see EIP-4337), the feature gap between custodial (e.g. Fireblocks) and noncustodial wallets (e.g. Metamask) has shrunk immensely. Before this EIP, noncustodial wallets didn't natively support social recovery, gasless meta-transactions, session keys. Account abstraction will allow developers to provide Web2-like experiences, without sacrificing decentralization. At a high level, the way account abstraction works is by bundling player interactions into a transaction that is validated by a smart contract wallet and paid for by another entity called a paymaster. It builds upon the meta-transaction model and gives smart contract wallets native support for separating the transaction sender and payer.

Devs (especially near term) may also look to relayers that submit payloads on behalf of third parties, allowing game developers to pay network fees on behalf of users. Doing so optimizes gas efficiency without having to use a custodial wallet.

A relayer is a service that allows users to make transactions on a blockchain via signing messages and API calls without having to submit transactions to the blockchain themselves. Relayers typically have to be supported on the client level to enable gasless/meta-transactions. Smart contract wallets can have native relay support. This means gasless and meta-transactions can be supported without a specific game client requiring relayer support.

Examples:

- Account abstraction repository
- Biconomy relayer
- Gas Station Network
- Build your own relayer with Oxsequence


Gameplay and progression
1UP
Victory
1UP

Instead of thinking about Web3 games as a monolithic construct, let's start to get specific about how blockchains can show up in a game, and the design implications.

First, "game loops" or "core gameplay": the main sequence of player inputs and engine feedback, like dueling someone in a match of Skyweaver, can be represented on-chain. Second, the "metagame," abstractions on the core gameplay like ranked ladder placement, like the rewards in Dookey Dash. Lastly, we will touch on Social Systems, although this area is more nascent. Let's dive into how the mechanics of these work in detail.

Game Loop

There are two common ways to introduce blockchain into game loops:

- **Simple interactions:** On-chain assets do not affect core gameplay; only updated out of gameplay
  - Cosmetics or characters that are brought into the loop (i.e. pre-match character selections)
- **Complex interactions:** Assets on-chain require frequent interaction during gameplay
  - Items that are picked up or held represented on-chain
  - In-game events simultaneously published on-chain

Example simple blockchain-game interactions (High latency tolerance)

- Buying or selling on-chain content that also exists within a game like skins, characters, weapons, and so forth
- Crafting items: burning owned content and creating (minting) new ones or modifying metadata (modification may be done on or off-chain)
- Trading earned or purchased content on a 3rd party or white-label marketplace outside of the game
- On-chain data only used for gating or validation of services, e.g. read-only, auths
- Rewards for winning and penalties for losing

Complex blockchain-game interactions (Less latency tolerance)

- Game events result in frequent blockchain interactions or modifications to metadata
  - Picking up an item
  - In-game achievements
  - Real time movement
    - Turn based games have greater tolerance
- Game events are triggers by on-chain events
  - Updates to the blockchain need to be represented in real-time within the game
    - If game events are intended to be random / not predictable, latency tolerance increases

Metagame & Progression systems

Most games have “metagame” progression systems, like achievements, battlepasses, ranks or tiers, and more. These systems track gameplay events, reward the player when pre-defined goals are met, and can help position players within a broader context of their community.
Later in this document, we break down NFTs in granular detail; for now, we’ll think about them as “non-fungible tokens” in the most basic sense. Here are some examples of progression systems that could be represented on-chain:

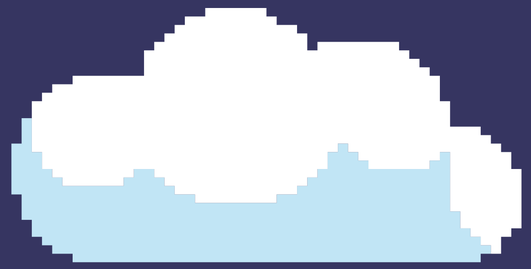
- An NFT whose metadata is updated as the player completes in-game requirements
  - Examples include leveling up a character, or reaching in-game mastery
  - Other milestones of character development
- Soulbound NFTs that represent access to a battle pass
  - Progress for the battle pass can be stored off-chain for ease of updating, or stored on-chain as part of metadata
  - Progress rewards can also be NFT rewards
- Progression that is tracked off-chain, but triggers on-chain rewards for ranked ladder progression

Game progression data can be quite rich. Some design considerations for developers include how much progression data should be reasonably stored on-chain, and what kind of tradeoffs a developer can expect, in terms of costs and performance.

Ultimately, there should be two game systems that:

1. **Manage progression; and**
2. **Handles granting rewards**

There doesn’t exist any native on-chain solutions or EIPs that enable these systems (yet). However, existing NFT minting and content management platforms are a good bet at fulfilling requirements for your game.



On-chain progression

On-chain progression can be considered from a number of angles. Here are some key considerations considerations when thinking about on-chain progression:

- Blockchains are **really good at storing things, but bad at frequently updating** what has already been stored. This is by design.
  - While updatability is often fundamental to game progression, it violates immutable cryptographic primitives
  - Design your progression system to around the strengths of blockchains (permissionless, player can reference / take with them anywhere) but avoid downsides (scalability, immutability)
    - e.g. Ranked season performance that players will want to be able to share easily but is only updated annually
- **Picking where to track progression, and how to increment**
  - A “step” for progressing a battle pass might be minting an incremental soulbound NFT badge, instead of modifying metadata which requires tradeoffs (See the metadata section)
  - **Don’t pick a too-small unit of progression to track on-chain**
- It is bad UX to require XP to increment on-chain after a battle or achievement, which would also require paying gas to confirm earned XP!
  - The benefits of putting that on-chain are eclipsed by the scale and ux drawbacks
- **Consider tracking large, socially significant events on-chain,** like leveling up, while keeping smaller achievements (XP accumulation) off-chain

- Another system of progression could be **session-oriented,** rather than achievement-oriented: settle to the chain at the end of each session.

These are all suggestions; this design space is ripe for innovation.

Social Systems

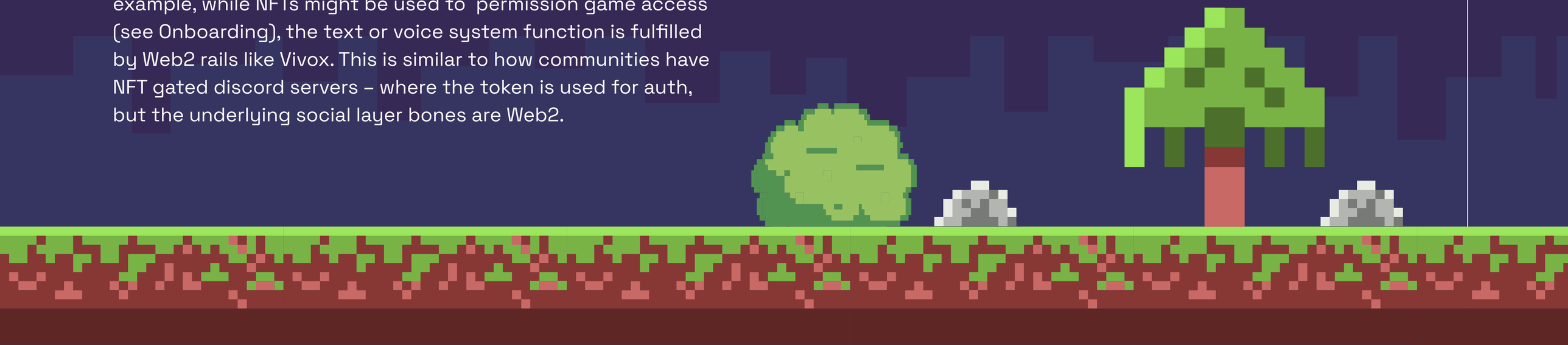
A core function of social systems is player identity, which opens up a number of different socio-gaming dynamics: bragging rights, community visibility, leadership, innovative gameplay, and so on. Public ledgers are a perfect social system for players to share accomplishments with one another, or showcase special items.

Multiplayer online games require social systems with features like a friends list, voice comms, game lobbies, and guilds or community building tools, among others. For the most part, these systems still exist within the traditional Web2 stack. For example, while NFTs might be used to permission game access (see Onboarding), the text or voice system function is fulfilled by Web2 rails like Vivox. This is similar to how communities have NFT gated discord servers – where the token is used for auth, but the underlying social layer bones are Web2.

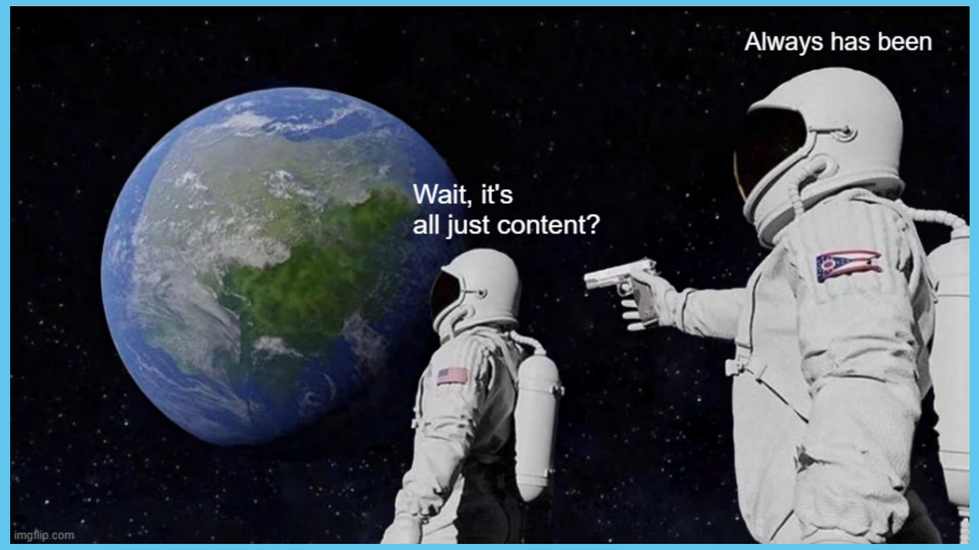
Even so, some developers are experimenting with blockchain use cases for systems that build robust communities, mostly in genres that introduce novel player experiences. What does that look like? Developers (like Crypto Unicorns) might use Lens Protocol as the base social graph, or leverage DAO tooling to build guilds, where tokens are used as voting rights.

Permissionless but deterministic aspects of smart contracts mean that guilds can make their own rules and voting structures without core game developers explicitly implementing these features.

Web3 social systems create dynamic on-chain/off-chain, in-game/ out-of-game experiences that compliment or transcend traditional hubs of gamers.




Content
&
Entitlements



Ownership means something different in Web3, and entitlements and content can look radically different. But in the end, content as a core concept remains consistent with Web2, it is mainly entitlements and ownership that are impacted by blockchain implementations. When thinking about content systems, you should start with traditional Web2 systems, and then innovate on architecture, based on your features/needs.

A non-exhaustive list of content types includes:

- Characters
- Cosmetics
- Achievements
- In-game items
- XP Boosts
- Hard and soft currency
- Resources
- Consumables
- Stackable items

Games can choose to have a variety of different content types; how they are implemented doesn't have to be uniform (i.e. content can be a mix of Web2 and Web3). All the content above might exist on-chain, but even so, Web3 studios must make design decisions that take into account performance, player needs, scalability, and other variables that affect the experience of their game. See the game loop section for more commentary on this.

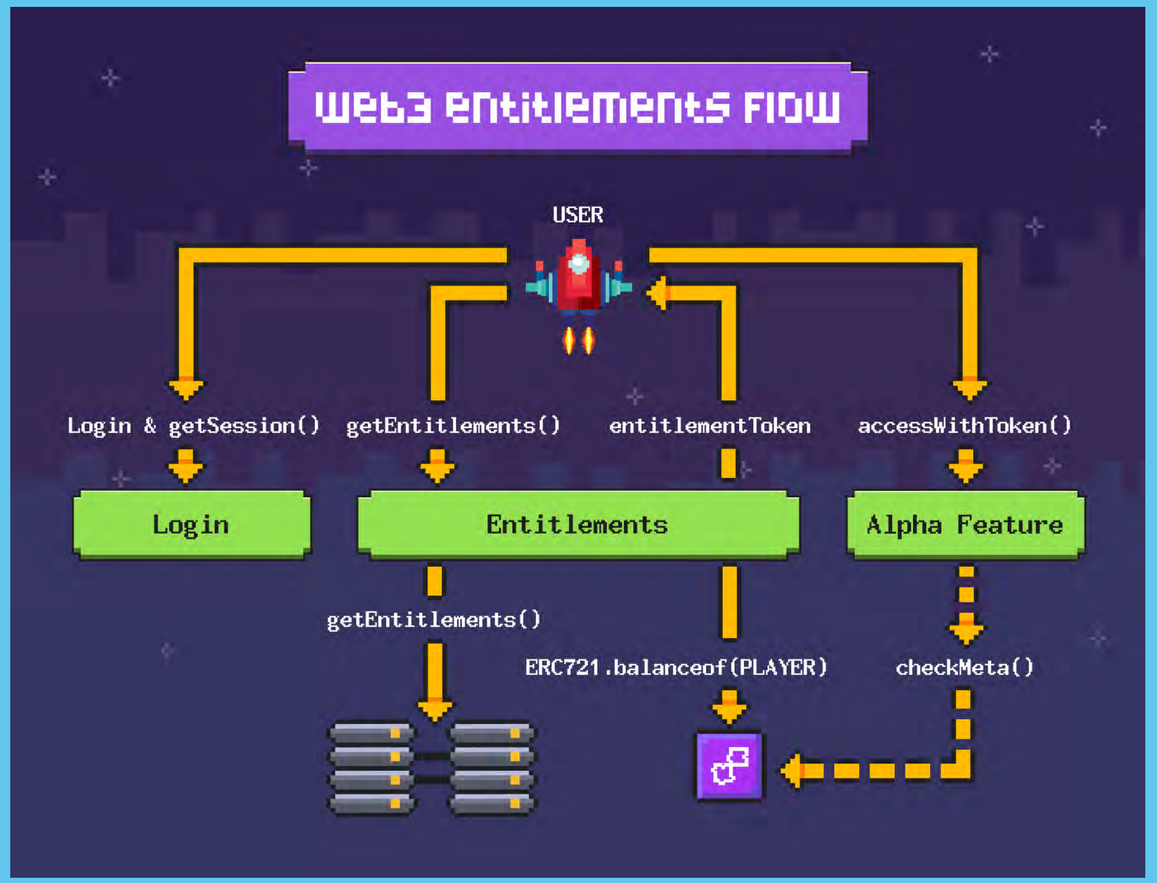
	Pros	Cons
Off-Chain	• Low latency retrieval of ownership and metadata (internal DB) • Can be iteratively created/updated/deleted as needed by the studio • Price can be entirely controlled by the game developers • Error proof / mutable	• Since it is managed within the game ecosystem, it requires additional work to move off-chain content to on-chain player-owned content. • Relies on trust with the game developer, since the content is technically “leased” to the player
On-Chain	• Can represent ownership on-chain. • Allows players to engage in trustless/ permissionless communities with the content outside of the game. • Player driven economic models • Liquidity • Some communities have strong feelings about this	• Pricing is subject to broader market trends (supply/demand). • Games need to implement additional logic to validate ownership throughout gameplay • Computational cost (and speed) of storing asset metadata on-chain and doing transactions is greater than doing so off-chain • Low error tolerance

All in all, the vision for a game–and what data the game asks of the user–will ultimately help you, as a developer, decide what inventory service you'd like to use.

Probing questions to ask yourself as you venture farther into Web3 functionality:

- What off-chain data does the game know the player has, and why is it ok for this to be off-chain?
- What on-chain data does the player bring to a game, and does it matter that this data is on-chain?

Experimentation will be key in figuring out the right balance to strike.



For items that are represented on-chain players may want to trade/sell on secondary marketplaces like OpenSea or Magic Eden. Non-custodial wallets allow players to do this freely, while custodial wallets give game developers the ability to limit what can and cannot be done with content.

It's important to note that enabling a player to trade away game content should not break the game. Users will appreciate this power! Digital ownership becomes *realer* when a game enables content to function as real-world objects. However, doing so does introduce technical overhead in building a cohesive player experience. You can read more about this in our marketplaces segment.

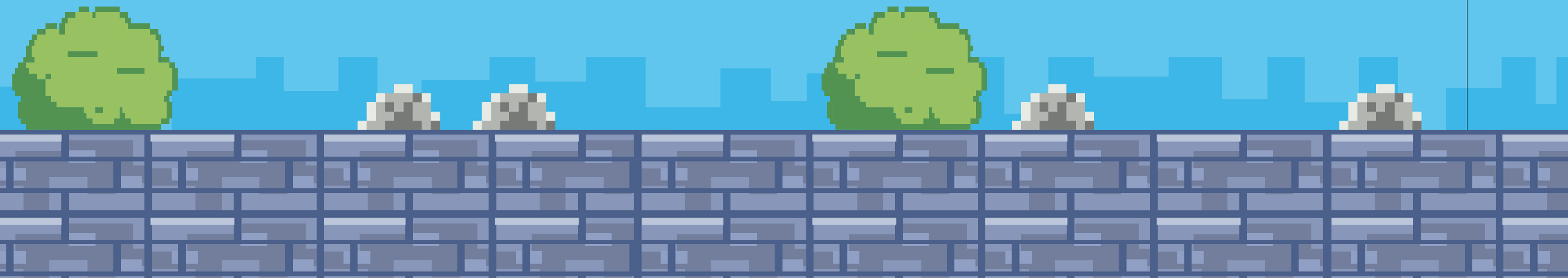
Smart Contract Implementations

Smart contracts, an execution of arbitrary code on-chain, can help you unlock Web3 functionality, and open new design space for your games. You can use smart contracts to both represent in-game items (e.g. equipable swords as ERC1155's), to fulfill trade requests (see marketplace section). However, executing arbitrary logic on-chain is more expensive and vulnerable than doing so off-chain.

The following are a non-exhaustive list of examples for smart contract implementations. Some minting providers may allow you to customize or bring your own smart contracts to the table; others will have their own templatized smart contracts for you to use.

Keep in mind that multiple use cases can have similar implementations, and that a single use case may have multiple different implementations!

- ERC20
  - Generally used to represent in-game currency (fungible tokens)
  - Specialized use cases for highly fungible materials (e.g. crafting materials)
- ERC721
  - Premium content, exclusive items (e.g.skins)
  - Can be soulbound, which means not transferable
- ERC1155
  - Multi-token standard that is referred to as a Semi-Fungible Token
  - Can implement both ERC20 and ERC721 functionality.



Content management

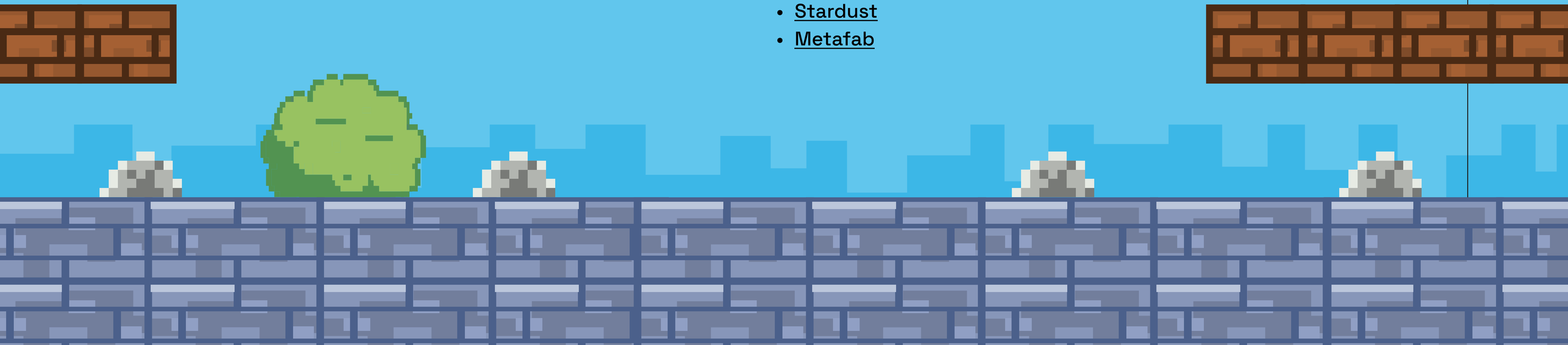
A key question that arises, from a design perspective, is how to provide on-chain representation of an in-game asset. Often, on-chain assets are constrained in the data they store; additional information, like image textures, sound effects, animations, 3D models, and so on, have to be mapped onto them (instead of those assets being represented on-chain).

You should pursue a content management system that can help not only with in-game representation, but also managing content in marketplaces. Throughout the life of a game, not all content is available for sale or obtainable at all times. Generally, this is configured through a content management system. This subsystem is responsible for managing collections for items that are available for purchase.

Web3 requires incremental content management system functionality. For example: minting requires art pipeline tools, plus any metadata updates that need to be in sync with an item catalog CMS, and so on.

But there are a number of tools that help you manage this complex in-game/on-chain interplay:

- Venly
- Nefta
- Stardust
- Metafab



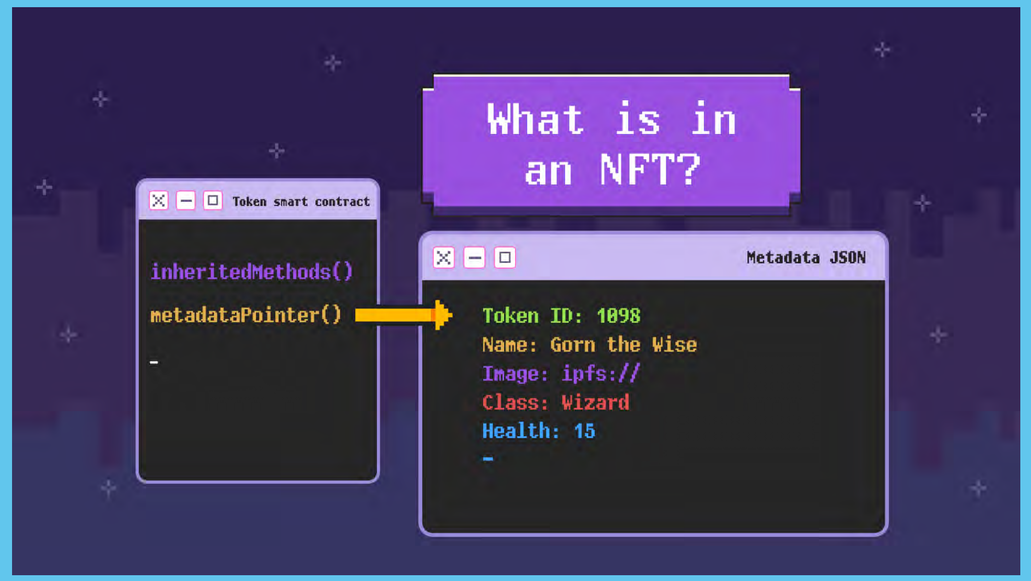
Terminology	Definition	Implication	Example
Transaction	The transfer of assets from a studio wallet to a player wallet, or from player-to-player.	The core of Web3 is peer-to-peer value transfers; such transactions can underpin Web3 gameplay, too	Buying a sword from a vendor or trading for it with another player
Minting	Process of creating an NFT	Multiple NFTs can be created through batch creation, or following ERC721a. This can also be “lazy” minted, where the item is converted to an NFT at a later time	Getting a sword as reward for defeating a boss
Burning	The process of destroying a token, on-chain	Can only be done by the custodian of the asset	Burning item components to craft on-chain sword
Dynamic	On-content whose metadata can be updated	Chain-state and on-chain assets reflect what happens in-game	Leveling up an on-chain sword

Metadata

Metadata is a common framework for associating game data with tokenized assets or on-chain objects.

For our purposes, we will define metadata as the game data relevant to a given token. Metadata can be item stats, proof of victory, guild membership, and so on. Think of metadata as the content behind the lock and key of NFTs. NFTs usually have an URI that points to a JSON metadata object (stored elsewhere), but there are many different ways to configure and store metadata.

The important thing is that you're storing metadata in a way that works for the performance and upgradability requirements of your game, but also community expectations.

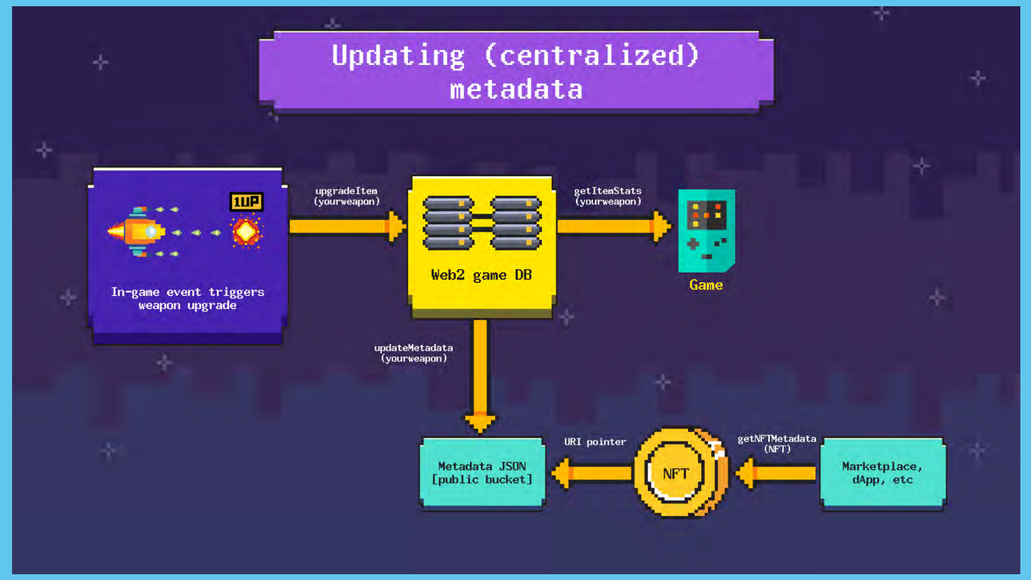


Here are some options for where to store metadata, from most to least "on-chain":

1. Fully on-chain
2. URI maps to immutable file storage, e.g. IPFS
3. URI maps to mutable file storage, e.g. AWS
4. Fully off-chain, e.g. unassociated with URI pointer

With Metadata architecture, developers face tradeoffs around decentralization, mutability, security, permissioning, and scalability. For example, you can make dynamically updating NFTs, but this may compromise either decentralization or security

Looking above, let's dive into option 3 – "URI maps to mutable file storage, e.g. AWS", in which NFT metadata is stored on a centralized, mutable database. Since the metadata is centralized and can only be updated through your services (or update events are easy to listen to), it is simpler to keep metadata in sync with game events. In addition, you can also have your game backend consume a Web2 db/cache of item metadata instead of having to query the NFT every single time, since you won't have unexpected changes in metadata. This is cheap and fast, since you are removing the latency and expense from always having to query the chain to get metadata. It is also secure, since you can roll back any unauthorized changes in metadata state. While this is very scalable, it is also centralized, and may not be suitable for games that heavily rely on on-chain UGC.



Contrast the schema above with a different hypothetical, in which a game ingests metadata from the chain directly, instead of using an intermediary database.

Such might be the case if you use proxy smart contracts or other on-chain frameworks for upgradability. These upgrades could be called by some 3rd party that isn't your game service, and could even be permissionless, which would be great for UGC.

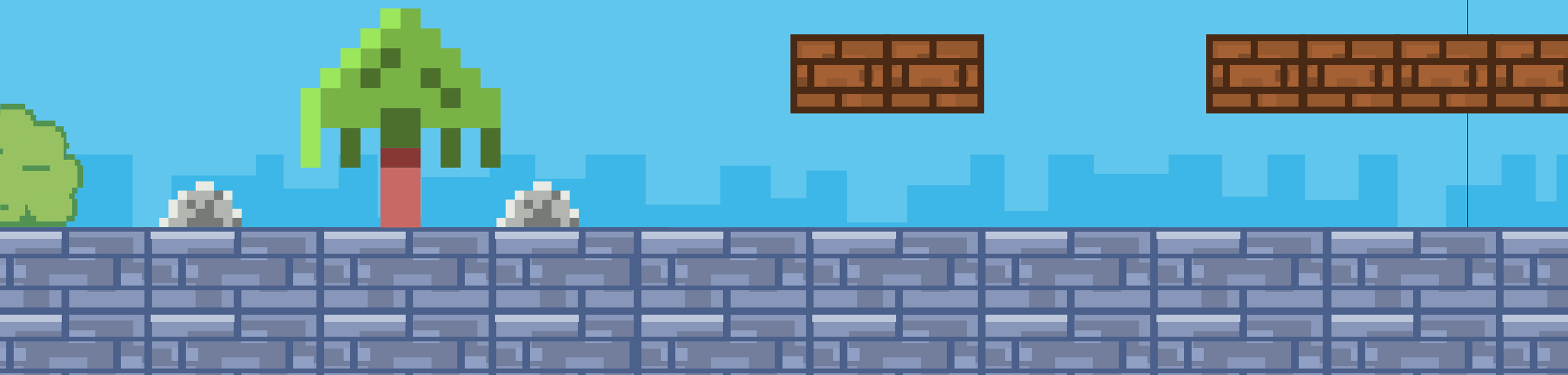
In this instance you'll have to listen and index much more thoroughly, in addition to hitting an IPFS. api much more frequently. Although there are services that combine these two calls, and you can optimize to only hit APIs when you know there is a change–in any case, you'll end up with higher latency and likely more cost.

BUT: you'll also have a more decentralized and permissionless content system. For some games and developers, that matters a lot.

Note: If you're using immutable file storage like Arweave/IPFS, but you want to have dynamic metadata, you'll need to use an upgradability framework. A number of methods allow you to switch the pointer of an NFT(s) to a different URI, or something to that effect.

These include:

- Proxy smart contracts
- Diamond standard / EIP 2535
- Token metadata represented by on-chain state.
  - This method requires implementing logic that updates on-chain state when the metadata should be updated.
  - Example from Alchemy using Polygon



Crafting

In addition to obtaining items throughout the game or marketplaces, some games may choose to have crafting systems where players can combine owned content to either create or upgrade content.

These systems can become complex with different recipes. Our friends at Pragma Platform have a great guide on how a system like this could be built: Creating a Crafting System Using Stores | Pragma Engine Documentation. On the Web3 side, Metafab has released guidance for crafting frameworks. AccelByte are also knowledgable on this.

Extending crafting functionality to on-chain items means existing content may have to be burned, and new content may need to be minted.

But it's also possible for crafting to be represented with on-chain metadata without requiring burning. Various upgradability frameworks, like the ones mentioned earlier, permit on-chain crafting. And if metadata lives off-chain, e.g. NFT URI points to AWS hosted JSON, you can implement crafting without having to do anything on-chain.
While there are complications in crafting when compared to Web2, blockchains are great at accepting permissionless contributions. Accepting UGC becomes easier, because you can parameterize what users can upload and craft, and then users can carry out these processes without requiring additional explicit authorization from the game.

There are plenty of sources that explicitly address on-chain crafting. Check out the following toolkits:

- Metafab
- Moonstream
- dNFTs

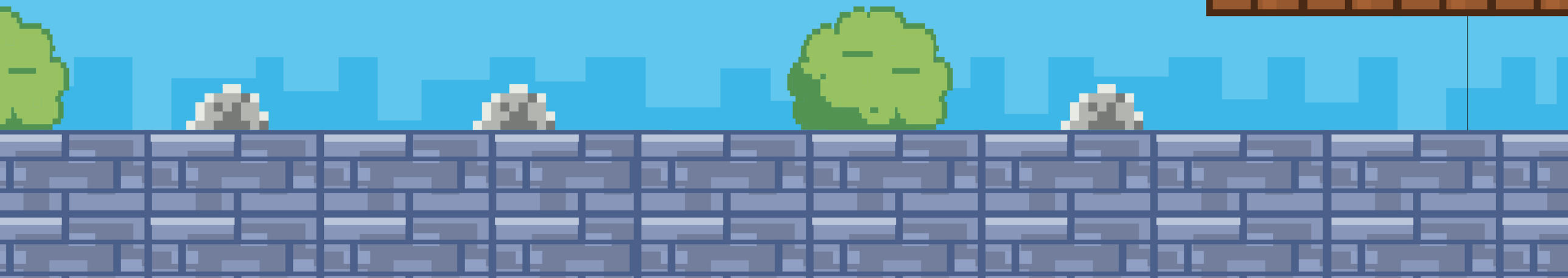
Store

Game studios distribute premium content through in-game stores. Important components of a store include:

- **Payment processors**, like Xsolla and Stripe, allow players to bring real world money into the game
- **Store catalogs** display what is available for purchase
- **Entitlements** tell game systems what a player is “entitled” to. This might mean access entitlements or one content entitlements
- **Store Processors** are the systems that communicate with your back end to grant entitlement or content once a payment has been successfully completed

On-chain assets complicate game design, and game developers face a number of new challenges, like finding a fiat on-ramp to help gamers buy digital objects even if they don’t have money on-chain; and NFT marketplaces, as a hub for gamers to swap tokenized money for tokenized game assets.

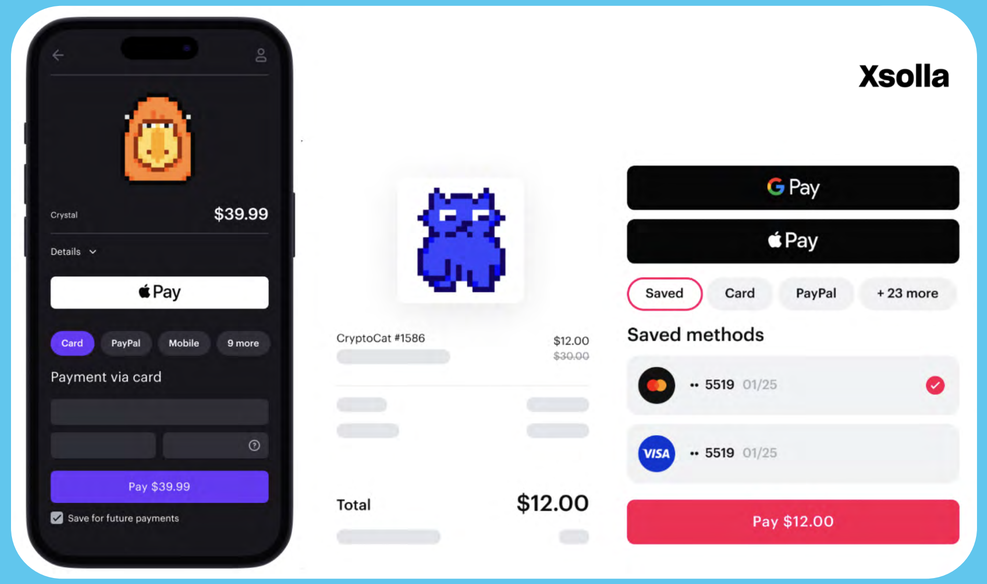
We’ll address both of these in the following sections.



Fiat On-ramps

Different player populations will have different payment preferences. Some players will already own tokens and will easily be able to transact fully on-chain. Most players will not have liquidity on-chain, or may only have liquidity on L1 (bridging should not be a native part of player journey).

Enter fiat on-ramps -- customizable widgets that developers can directly embed into their game, similar to non-crypto payment providers. Almost all will also handle KYC! Most, if not all, games should enable fiat to digital asset (NFT) purchases.



This is the most general onramp. Enables players to purchase a game's native on-chain token

Examples:

MoonPay

stripe

Abstracts away the interaction with cryptocurrencies and provides seamless UX to obtain an NFT with real world money

Examples:

paper

Xsolla

Marketplaces & Trading

Traditional stores facilitate one-way transactions from studios to players. But with on-chain assets and non-custodial wallets, developers can now create open marketplaces where players participate in buying and selling their own content, similar to the auction house in Diablo III (RIP).

This is a key value for Web3 games:

Player ownership levels up the network effects of games.

Integrating a marketplace into gameplay is an easy hack to enhance the overall user experience. In such scenarios, players never have to leave the game to participate in the game economy, even as their actions are recorded on the blockchain. Secondary marketplaces like Rarible and Opensea can be used to out-of-game trading, but such solutions come with additional overhead for non Web3 native players. Low priced tokenized assets (e.g. fungible craftable resources) should probably be bought in-game, while higher priced (and non fungible) items will benefit from the functionality of secondary marketplaces.

Developers should keep in mind malicious trading behaviors. Users may be able to sell items, for instance, while still in combat. Nevertheless, smart contracts are inherently robust at holding items in escrow, and developers can always confirm continuous ownership for a battle before service grants are awarded.

Here are some tools that can enable this experience, though they still require in-game UI work:

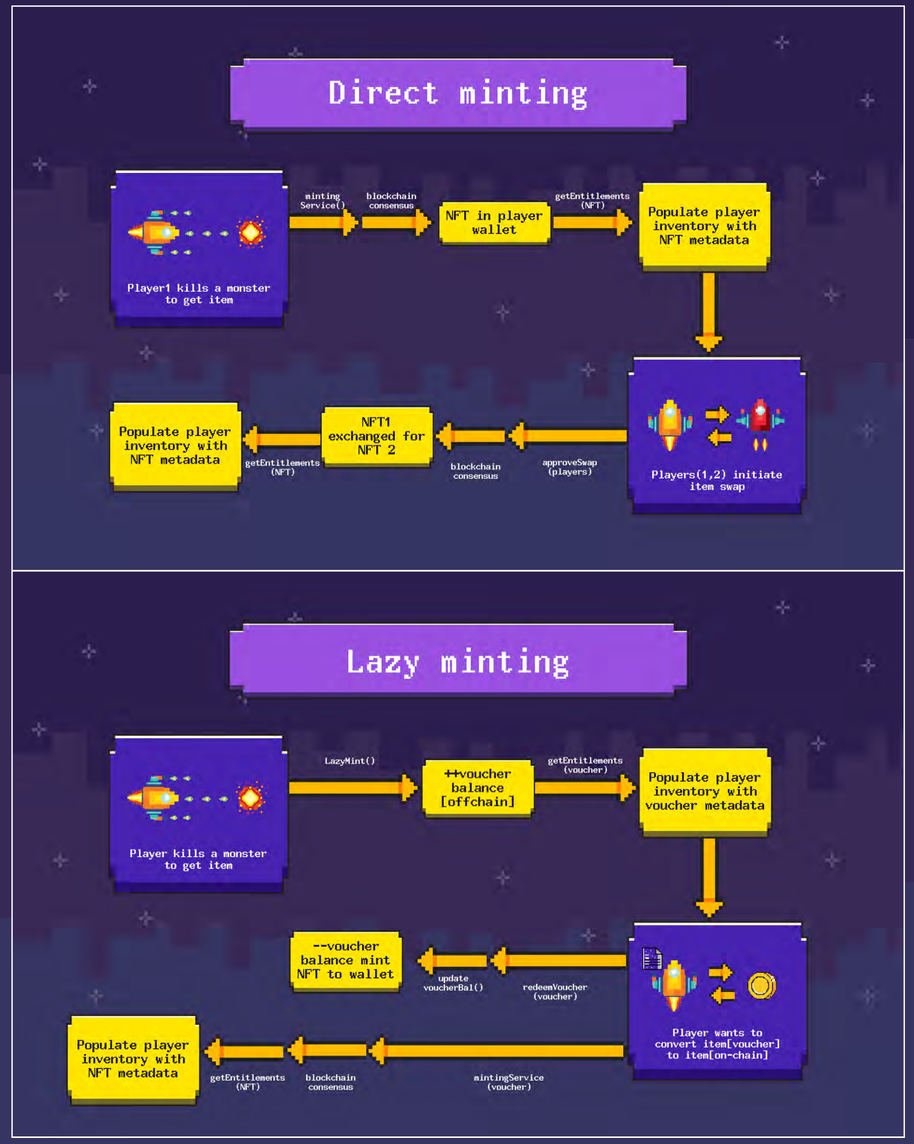
- NiftySwap (by Horizon)
- Opensea's Seaport
- Rarible Multichain Protocol

Blockchain primitives

Minting

As mentioned earlier, minting is the process of publishing off-chain information onto a blockchain at a unique reference point–in the case of games, a token. But there are nuances about when and how one decides to do this within the game loop.

Let's imagine the example of a sword NFT in an MMORPG. When the player buys the sword from a store, the item may be minted or airdropped at that precise moment. Alternatively, the content could be granted as a Web2 playable voucher, until the player opts to claim that NFT with their wallet. This is an important early decision to make: whether tokenized assets will always be granted on-chain by default to players.

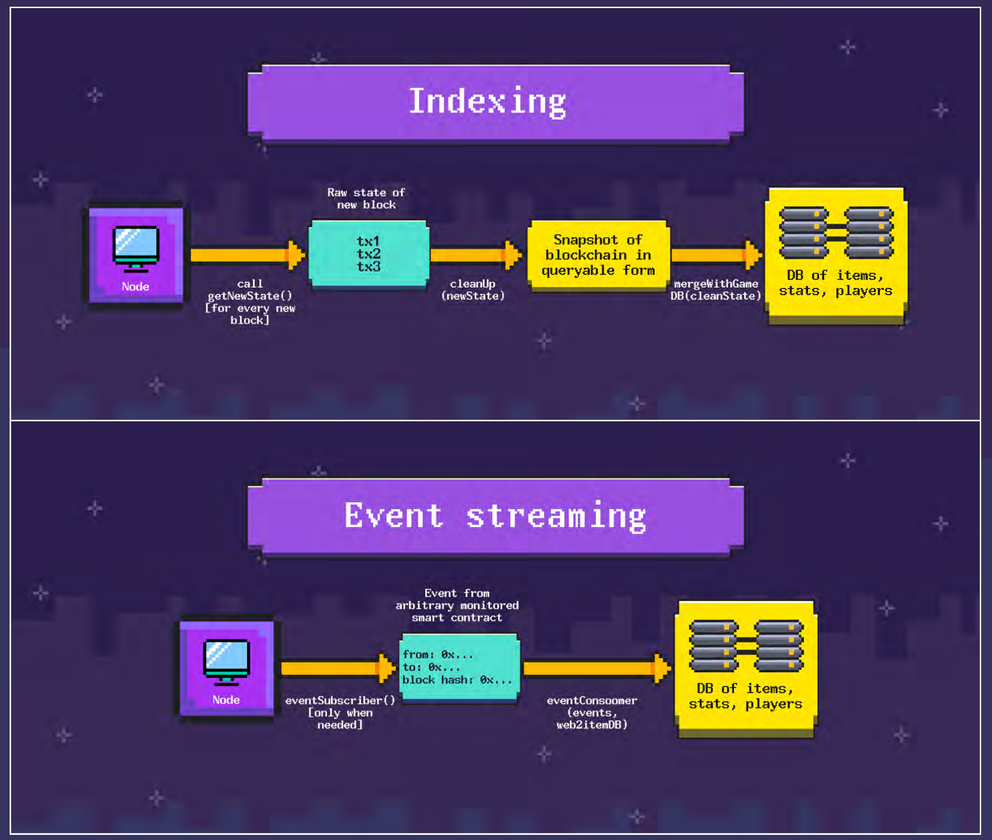


Let’s think about the pros and cons of each minting methodology

	Pros	Cons
Default Minting	• Simplicity of ownership reflected in just one (decentralized) place • Fewer vulnerabilities	• Requires players to have wallets (or the developer to spin up wallets in the background/pre-mint), • High cost/latency • Low engagement content will go on-chain • Less flexibility in player experience
Delayed or Lazy minting	• Cheaper due to less on-chain activity and indexing activity • Accommodates players that do not have wallets • High-value items will naturally go on-chain while lower engagement content will stay off-chain • More flexibility in player experience	• Developers will need to account for items that are not yet NFTs as part of the total economy, which will mean managing two different states for a given item. • Increased entropy / complexity

Reading the chain

Developers need to design a system for listening for on-chain events and reflecting those changes in the game state. Devs sometimes refer to this as “Indexing,” but chain reads may not actually implement the formal definition of a blockchain indexer.



Why would you want to listen to on-chain events?
The answer will depend on the vision and mechanics of the game, but there are many varied reasons. A few of them:

- All holders of a soulbound ranked season token are airdropped a wearable achievement cosmetic
- A player uses an out of game NFT marketplace to trade an item
- An item is burned in-game to upgrade another existing NFT

Unlike in Web2, listening to on-chain events in Web3 is more complex. Decentralization means blockchains introduce latency, as blocks take time to propagate. Along with an optimized minting framework, how you read events will be one of the most important architecture decisions to make.

Though listening to on-chain events is an open design space, there are at least two popular methods, each with various considerations. Let’s take a look at Indexing and Streaming.

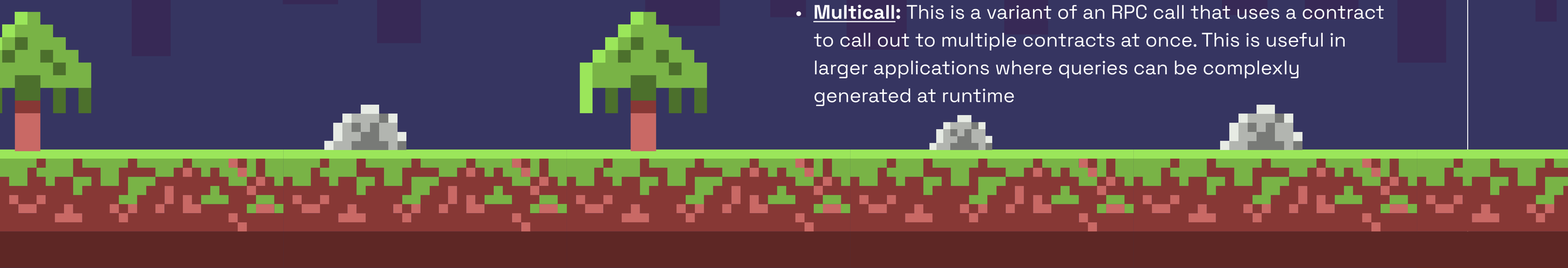
Indexing

Games that rely on frequent queries or aggregating big data turn to Indexers (e.g. The Graph, Sequence, Space and Time). Think of indexers as carbon copies on-chain state, an up to date (and historical) Web2 data warehouse for a blockchain. This is especially useful for app specific chains (or roll-ups) like Polygon Supernets where all the transactions are relevant to the game developer. An indexer can either be publicly maintained or function as a private data store.
Optimally formatting raw blockchain state for consumption can be coupled with other relevant off-chain data by leveraging an indexer.

Streaming

Streaming is more appropriate when a game or player must respond to specific events that take place in contracts on a backend service. Again, there are a number of methods to stream, and this remains an area of innovation.

- **DIY (e.g. Geth, Bor):** For self-run nodes. Can help meet latency or throughput requirements, and easier to self-guarantee
- **Node as a service (e.g. Moralis, Quicknode):** If using hosting services and event volume is reasonable
- **RPC Call:** For applications and web apps that need to fetch accurate, up-to-date information. RPC calls are usually easier than the construction state from a series of blocks. It is current-state-centric, and the slowest option
- **Multicall:** This is a variant of an RPC call that uses a contract to call out to multiple contracts at once. This is useful in larger applications where queries can be complexly generated at runtime



To summarize:

- Indexing is best for match history, item provenance, and so forth. Best for when it is okay to be a few blocks behind the current state, important to have snapshots of historical state at arbitrary block height. Indexing is also particularly useful for app specific chains where all activity is relevant.
  - Considerations: Indexers have worse SLAs around downtime and lag behind the canonical block. If an implementation demands the latest state, indexers may not be ideal.
- Streaming is best for in-game loop items on public blockchains. Low latency and high availability.
  - Considerations: When streaming events from a node, not all data may be included in payload, just the state delta expressed in that event. This may necessitate additional calls, adding up latency and cost.

Though covered earlier, it's worth remembering that metadata schema, or more broadly the decision of what item data goes on-chain/off-chain, will impact the ease and simplicity of indexing. Revisit the scenario where metadata is stored in an off-chain database. We would only need to read the chain for item ownership. But if the metadata is updatable by services outside the game, indexing becomes a lot more complicated. In this scenario, you would have to get item metadata from the URI on a frequent basis, increasing latency and cost. Managing synchronization between the item catalog and blockchain database would be more complicated.

In addition, how you consume blockchain reads is important. For high scores that are recorded on the chain, it may be ideal to directly consume on-chain information without caching. For inventory implementations, it may be better to consume/cache on-chain data in a Web2-style inventory service for better performance and failover handling. We think that caching and indexing have a good tradeoff of centralization for performance, but also that fully on-chain experiences open up new game primitives that should be explored.

Some games go a step further w/r/t on-chain data, using blockchain as the game server. In this scenario, a blockchain is leveraged as more than an item database, utilizing it instead as a server.

Further reading for fully on-chain gaming: MUD, Jump Crypto

User acquisition

Blockchains present new problems for UA, but also novel mechanics and solutions.

Typically, games will use game-focused tools like AppLovin to track attribution and marketing performance broadly. In Web2, the attribution of credits and rewards is granted to the “last-touch” platform. In Web2, tracking behavior from platfo rm to platform is really hard! Attribution infrastructure for ad agencies is based on trailing user lifetime value metrics, instead of actual forward looking monetization. All of which is to say: many games pay a lot at certain points in the value chain, and players don’t capture anything.

Blockchains unlock new network effects

Instead of paying advertisers to drive downloads, developers can directly incentivize users by passing along tokenized rewards for downloading or referring friends. Anyone with a following can be a platform to evangelize a game, even and especially players and their social networks.

This is how value capture increases not only for players, but game studios as well. Increasingly, this can be done in a trustless and programmatic way, with the possibility to scale a game to new heights.

Because wallet activity is public, developers can see what other games users play. Basically, this becomes an open-source, visible ledger of important data that showcases overlapping interests and competing worlds. Such data can be used to guide UA and product development. How so?

- Devs can see how much users spend on other games, what games lapsed users play, and a million other kinds of useful metrics that help guide design.
- Devs can also identify communities to target based on overlap and monetization behavior.

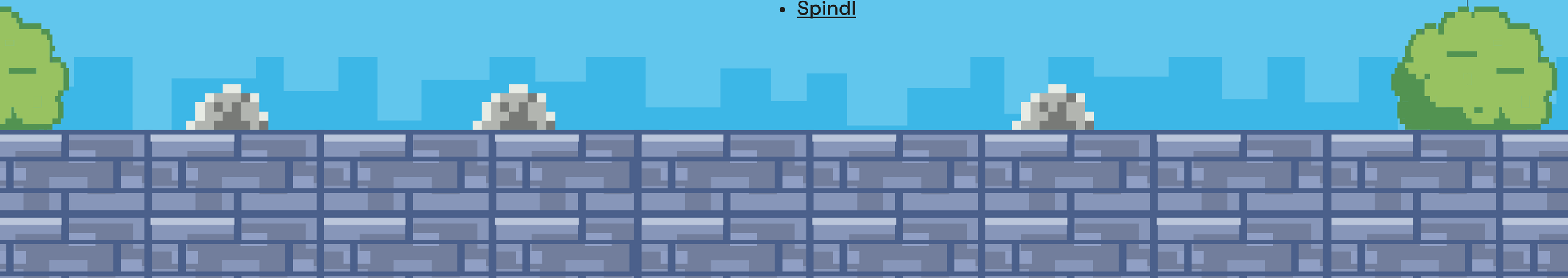
By using smart contracts, game designers can now reward marketers based on actual monetization, and so in a trustless manner, ensuring that the devs aren’t paying for more value than they receive. This is an improvement on a last touch model for all parties involved.

BUUUUUUT, there are some downsides.....

1. Many traditional ad platforms, like Twitch, Google, and Facebook, have mixed friendliness to blockchains. There are technological and cultural barriers for adoption by these legacy platforms
2. While more sophisticated attribution is possible, turnkey solutions are still nascent
3. Asset liquidity draws bots and non-players who farm assets and then churn. The lack of KYC/demographic credentials inherently contained in wallets makes detection difficult
4. Inorganic growth can crowd out a real, grassroots playerbase, especially with adverse selection of micropayments to users

So we can bucket features in UA, with the companies that are solving them:

- **Questing:** Creating activity based rewards to grant loots for in-game activity
  - Moonstream, Sesame Labs
- **Attribution:** Measuring campaign performance against long term player activity
  - Spindl




Tokenomics

Project Discovery
Product and participant discovery

What? The discovery process objective is to learn about the product, its participants, and the primary and secondary goals of the ecosystem.

Why? Token design always needs to focus on achieving the participants' and their ecosystem's primary and secondary goals.

How? Having discovery session(s) with the developers to understand the goals by asking appropriate questions.

Market Design
Framework of the ecosystemy

What? Market Design aims to design an ecosystem that will achieve mass adoption by increasing **size**, mitigating **congestion**, and increasing **safety** by implementing rules, incentives, and punishments that govern the ecosystem.

Why? Markets can be unstable, and participant preferences can change over time. Creating an ecosystem with good Market Design will encourage growth and adoption levels that will help increase stability and provide enough choice to participants as they evolve.

How?
- Markets can increase in **size** and reach adoption through network effect, brand awareness, and incentivization.
- Markets can mitigate **congestion** through governance, dynamic fees, reduced information asymmetry, and curation.
- Markets can increase **safety** through governance, credibility, transparency, platform ratings, and skin-in-the-game incentives.

Mechanism Design
Governance of ecosystem participants

What? Mechanism design governs participant interactions within the ecosystem to maintain **size** and **safety** by facilitating **engagement**, **reducing information asymmetry**, and **reducing bad actors**.

Why? Without additional governance on the participants themselves, markets can become at risk of losing adoption and reduced levels of safety caused by imbalances in market knowledge and a rise in bad actors.

How?
- Governance helps to keep participants safe during interactions and also incentivizes participants to vote, increasing **engagement** and **reducing information asymmetry**
- Non-Financial Incentives focus on participant reputations and the creation of fair ecosystems to **reduce bad actors** and **information asymmetry**
- Structure **reduces information asymmetry** by ensuring that participant interactions are accessible and equipped with enough information to make appropriate decisions.

Token Design
Token incentives for ecosystem participants

What? Token Design considers the role of the token within the ecosystem, specifically the incentives and governance that create **ecosystem participation** and **token demand**.

Why? Without proper token Design to facilitate **demand** and **participant engagement**, the ecosystem risks reduced size, reduced adoption, and exposes the token to volatility and price depreciation.

How?
- Monetary Policy controls token **demand** through token supply control, distribution, and leverage requirements.
- Choosing a bonding curve to facilitate an appropriate shape of price action based on ecosystem goals can help to control token **demand** and **engage participants**.
- Ecosystem incentives such as earnings and bonuses can also help to increase token **demand** and **participant engagement.**

Design Testing
Iterative design through modeling

What? A model simulates ecosystem participant behaviors (independently and interactively) based on incentives and governance to identify outliers and ensure a successful and sustainable token economy.

Why? Tokenomics is an iterative process because participants have unique behaviors and incentives that change over time. Once a Token Economy is live, new participant strategies and behaviors emerge. The goal is to identify these in advance through testing and iteration.

How? Agent Based Models using programming languages:
- R: Statnet, EpiModel, and RNetLogo
- Python: AgentPy, Mesa, and CadCad

Tokenonmic specific software:
- Machinations and Metanomic

Introducing an in-game token introduces additional considerations for designing a game economy. The principles of designing a token economy for a Web3 game draw from a combination of Web2 games, classical economics, and blockchain tokenomics. Gamedevs should rely on their previous experience, while at the same time without feeling constrained by it.

If/when designing a token, developers should understand these considerations:

- Purpose of the token
- Ease of use
- Deflationary/inflationary dynamics
- Governance
- Incentive mechanisms
- Token standards
- Capital raise

On top of designing for these factors, it's highly recommended that developers test their design through simulations. Some simulation partners who can help with this:

machinations · Nefta · NAMI · Economics Design

For more detailed information see our Tokenomics deep dive section

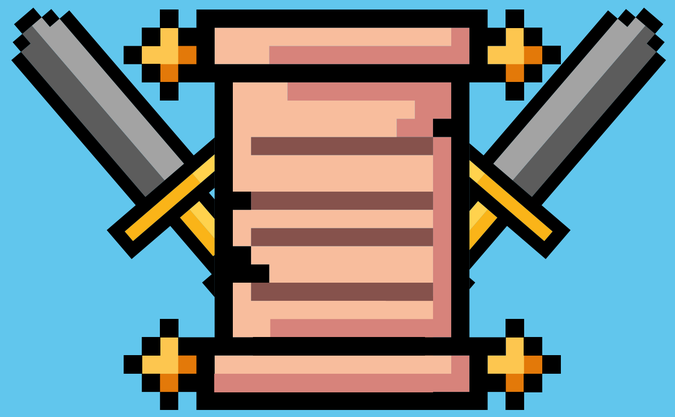
Tokenizing items in persistent games

The most important question for any developer thinking about tokenization is: How does tokenizing in-game items benefit the player?

In-game economies are complicated; putting them on-chain introduces entropy and constrains developer tweaking (as opposed to Web2). In addition to internal economy design, developers must anticipate the many complicated ways that out-of-game transactions, investments, and speculations will impact player experience.

For example, if non-players, what we might call “investors,” hoard items, this may indeed push the price of items up–but also decrease availability. A contagion of other effects may spread across on the metagame and to other items. Balancing changes and new content, crucial for game health, could have real world economic impact. Utility for players of tokenization needs to be clear to outweigh potential downsides.

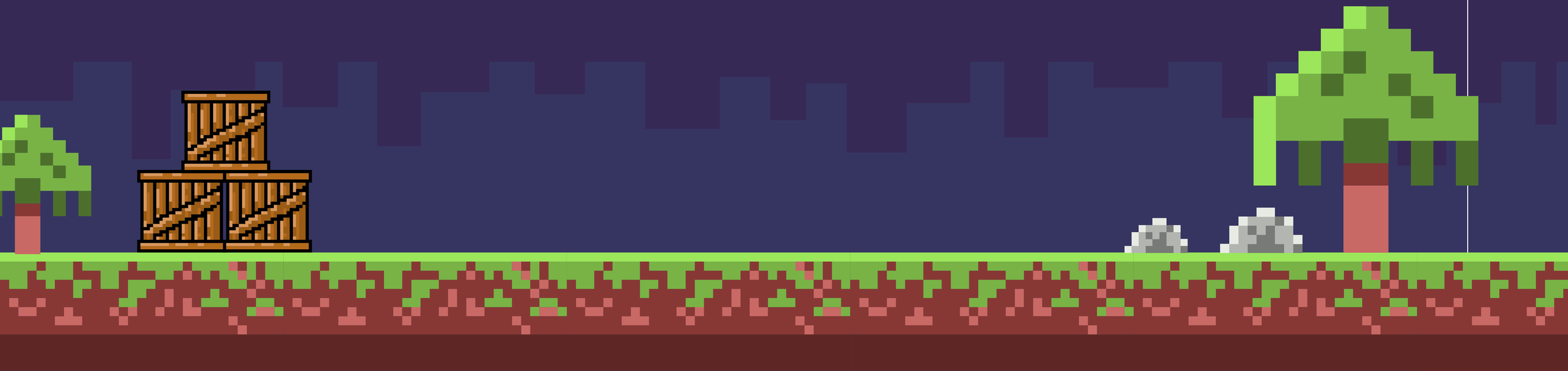
Publishing platforms



As always, it's important for developers to do their own research. You should check platform policies and ingest their ToS directly.

Platform	Publisher	Current outlook	Example of compliant flow	Revenue share model
PC	Steam	Somewhat prohibited / TBD	In client wallet integrations are discouraged	N/A
PC	Epic	Allowed	Can launch game on Epic Games Store, utilize Epic's payment rails to sell NFTs for fiat, or bring NFTs into the game that are purchased outside Epic	12%
Console	Xbox	Prohibited	N/A	N/A
Console	Playstation	Prohibited	N/A	N/A
Mobile	iOS	Allowed with restrictions	In order for an NFT to entitle content in-game, users must have the ability to buy that NFT within the app using Apple's payment rails. Refer to 3.1.3(b): Multiplatform services	<$1M sales: 15% >$1M sales: 30%
Mobile	Android	Allowed with restrictions	Similar guidelines as Apple, although games can choose to distribute through a third party app store outside Google Play. Refer to "Real-money gambling, games, and contests"	<$1M sales: 15% >$1M sales: 30%

Appendix



Wallets deep dive

Context:
The wallet type a game employs changes the overall player experience when interacting with the game.

Custodial wallets

Custodial wallets take custody over the private keys of users. That means that players are trusting their assets to a centralized third party. The game-studio takes on ownership of transferring/modifying the assets on-chain on behalf of an off-chain player request. A player has no on-chain authority over the items.

Pros:

- Low barrier to entry; easier wallet creation and onboarding
- Can reset password rather than store a private key; easier lost/deleted wallet management for the average user
- Game studios can have full control over assets and how it interacts with the game
- Players don't necessarily need to ever learn that the assets live on-chain since it's all abstracted away
- Can design and build your own implementation of this as it fits with your game.
- Better support offerings - assets can be reimbursed, removed, etc.
- Reduced friction in high-frequency transaction scenarios
- Can be deployed on behalf of the user

Cons:

- No flexibility for a user to transfer assets out of the wallet without the game studio's approval
- Only as secure as the centralized entity

Examples:

- Stardust Vault
- Forte
- Venly
- Nefta

Non-custodial wallets

Addresses where the user has full control over the assets and authorize each transaction. These may come in form of either Smart Contract Wallets (including smart contract wallets), Externally Owned Addresses (including MPC).

Upon creation, users get a mnemonic phrase used for recovery of the wallet in case a user needs to recover their wallet or social recovery.

Account Abstraction (note to reader, this is WIP)

Account abstraction levels up smart contract wallets, and we believe it will transform the way players interact with wallets, and thus with games. When a player interacts with anything that is contained in their wallet, traditionally they need to sign a transaction to confirm the action on the blockchain. With account abstraction, a player can delegate the signing to a trusted party (the game dev) throughout the duration of the game. It would be akin to session tokens while playing the game, but in this case the player authorizing the game within a specific set of actions.

Smart Contract Wallets

Pros:

- No explicit private key management
- Game studios can be granted permission to perform user approved actions with assets in the wallet
  - Can specifically enforce rules of specific transactions.
- User has full custody and control of on-chain assets
- Enables seamless UIs (no signing) for contract interactions
- Can be programmatically deployed on the user's behalf

Cons:

- Limited with interactions on a particular blockchain (e.g. a wallet on Polygon wouldn't be able to bridge to ETH without having a wallet also deployed there)
- May require users to share trust with other parties

Examples and further reading:

- Sequence Smart Contract Wallet
- Argent (Not gaming specific)
- OpenFort
- "The history and future of account abstraction" by Nethermind

Multi-Party Computation (MPC) Wallets

Pros:

- Private key is sharded amongst trusted individuals.
- Increased security as most, if not all, parties are required to perform actions with the wallet
- Similar pros to Smart contract wallets
- Can be used in conjunction with Smart Contract wallets to further secure the assets

Cons:

- Requires users to share trust with other parties

Examples:

- Fireblocks
- Coinbase Wallet
- Web3Auth

Externally Owned Addresses

Pros:

- Simpler backend development not worrying about player keys
- Same address can be used to access multiple EVM compatible chains (polygon, eth, optimism)
- Potential added security of having their private key on a physical hardware

Cons:

- Player needs to manage private keys and likely memorize a mnemonic seed phrase
- Players need to manually sign
  - You can get around this by requesting players to share private keys, however that is generally considered bad practice

Examples:

- Ledger
- Metamask
- Rainbow

Tokenomics deep dive

Process Overview
The principles of designing a token economy for a Web3 game have a lot of similarities to traditional game economy design. This portion of the document will outline the Token Economy design process, focusing on new considerations for Web3 development.

Discovery Phase
During the initial planning and discovery phase, it is important to consider some of the following questions. Does the game need a token? What purpose is the on-chain token solving (fundraising, liquidity for users, medium of exchange, etc.)? What are the shared goals and objectives for the tokens amongst users and developers (token stability, in-game transaction volume, etc.)? How many or little 'Web3' components should the user experience?

Market Design
Web3 Market Design focuses on increasing the size of the user base, all while creating a safe and easy environment to transact within. Here is a good twitter thread on NFT acquisition 101.

Size
Network effect is a successful strategy that has helped many Web3 projects. Project advocates will champion brands on Twitter, and many projects will give users access to the IP to make digital content and physical merchandise.
Airdrops and yields are two other effective strategies for fast and large user acquisition. Blur.io is an example of an NFT marketplace that has acquired a significant market share through a thoughtfully planned token airdrop. Here is a paper that looked at a series of token launches and concluded that yields were one of the primary user acquisition methods.

Safety and Ease of Use
Ensuring users have enough information, fast transactions, and purchasing options will create a friction-free environment. The governance of transactions (NFT sales, token swaps, etc.) can be hard-coded at the smart contract level. NFT ranking and analytics websites can help provide more information, and services such as premint can help with demand during NFT sales.

Governance
Users can actively participate in proposals and governance on important ecosystem concepts. This process helps to engage users and equip them with more knowledge. Users can propose and vote by holding a token or NFT and fitting any criteria the developer sees fit. Governance can be done directly or through a DAO and services such as snapshot make this easy. Certain concepts should stay out of the hands of the user, and governance can be hard-coded into smart contracts. Oracles can help pull in data from on or off-chain to satisfy conditions that need to be met.

Mechanism Design
Mechanism Design helps protect the users acquired and the market built by creating governance and incentives to promote participant safety, knowledge, and engagement.

Incentives
Incentives can promote safety by holding people accountable and reducing bad actors within the system. Some common methods are rating users, gating future token sales based on certain behaviors, staking for participation in decision-making, and even KYCing or linking to users actual identity.

Token Design
When designing a token for a Web3 game some important considerations are: what type of token standards will be used, where the tokens will be bought and sold, is the token a part of the capital raise strategy.

Token Standard
For tokens that will be used as a medium of exchange or governance, a fungible ERC-20 is standard. Games sometimes have multiple tokens in their ecosystem (hard currency and soft currency); in these cases, it is common to see one token on-chain and the other off-chain. When launching an ERC-20, at some point, users will want to be able to buy and sell the tokens. Token transactions on-chain occur on an Automated Market Maker (AMM) on either a Centralized Exchange (CEX) or a Decentralized Exchange(DEX). Keep in mind that going live on a Centralized Exchange can sometimes cost over six figures.

Capital Raise
When a token is part of the capital raise plan, it is important to consider what type of sale mechanism best aligns with the product. Token sales can be done in multiple parts and use different mechanics throughout the process. It is common practice for tokens sold during these initial phases to have vesting schedules (lock-up and release periods). Some common initial token sale mechanics for games are a private sale or a initial dex offering.

Design Testing
Gamers can be unpredictable, act irrationally, and change behaviors as they learn new game strategies. The former can impact token and NFT markets with the potential for cascading effects. For these reasons, it is important to test and iterate over the economic design through simulation. There are two major options available.

In-house Simulation
Developers can run a robust simulation in-house by developing an Agent-Based Model on Python (Mesa, CadCad, AgentPy) or R (Statnet, Epimodel, RNetLogo). The benefits are more flexibility, customization, and control over the simulation.

Third-Party Simulation
Third-party Tokenomic simulation services are available from vendors such as Machinations, Nefta, Nami, and Economic Design. These vendors range in services from full service to just analytics and simulation. We've heard quotes at around $50k. The benefits are a no-code user friendly process.


polygon labs