Gaming: How Solana is Changing the Playfield

What's this Article About?

Not your keys, not your crypto

. This adage, central to the ethos of crypto, is equally relevant to video games. Integrating blockchains into the gaming landscape is more than just an innovation. It is a paradigm shift enabling decentralized asset ownership and autonomy. At the forefront of this shift is Solana. It offers the fastest smart contract platform to harness the benefits of Web3. By leveraging Solana's high speeds, low and predictable fees, and low latency capabilities, developers can create unique gaming experiences that aren't possible anywhere else.

This article explores the potential of blockchain in gaming — examining the arguments for Web3 gaming and why game developers should consider building on Solana. The article also discusses installing, configuring, and using popular frameworks and platforms to create Solana-integrated games.

Why Web3 Gaming?

For Gamers

No other video game's collectibles come close toCounter-Strike. Depending on the rarity and float, some of the rarest Counter Strike 2 skins (i.e., in-game weapon cosmetics) can be worth thousands of dollars. Neon, a well-known trader in the Counter-Strike community, had an inventory of skins worth over $100k. On October 13th, 2023, he was falsely bannedby Valve's Anti-Cheat (VAC) System after enabling AMD's Anti-lag+ feature. A VAC ban renders a player's inventory un-tradable, meaning Neon's $100k inventory was permanently bound to his account. Valve has formally addressed the issue, and luckily, Neon was able to have his ban removed. Counter-Strike 2 is still having problems with false game bans.

In a traditional gaming platform likeSteam, there's always an underlying risk of losing access to purchased games or collectibles. There is nothing stopping Valve from taking away your assets. This seems unlikely due to the severe reputational damage and destruction of entire player bases. However, they could increase the drop rates of rare items, thereby decreasing their worth. This is precisely what happened with Team Fortress 2. Regardless of its worth, every game you buy and every in-game item you collect is managed by a centralized entity. Gamers are at the mercy of gaming corporations. Even Ethereum was created because of World of Warcraft's 3.10 patch. Vitalik Buterin has spoken at length about the "horrors of centralization" as Blizzard nerfed his character's siphon life spell, ruining his years of playtime. Gamers are tired of being at the mercy of centralized entities. Web3 gaming provides a way out.

Web3 gaming is a paradigm shift. It leverages blockchain technology to decentralize asset ownership and decision-making processes. It ensures transparency and security, where players genuinely own their in-game assets across platforms. This results in virtual economies, like Counter-Strike's skin market, with the benefits of transparency, security, and composability. These economies facilitate new user-centric gaming models, such as Play-to-earn (P2E), where players are rewarded with real-world value from their gameplay.

This is far from a niche concept. The potential of Web3 gaming to revolutionize the industry is evident with several successful and influential games. For example, Decentraland offers a virtual world where players can buy, sell, and develop real estate as NFTs. Tokens.com paid over $2 million for real estate in Decentraland's downtown core. This showcases the possibilities of virtual economies, digital experiences, and personal space within the Web3 ecosystem. Similarly, Play-to-earn games such as Axie Infinity and lifestyle apps such as STEPN (a move-to-earn platform) have garnered international attention by demonstrating how players can earn an income via gameplay. Games such as Star Atlas and Aurory have had similar success on Solana. These are not isolated examples. They are part of a broader trend where gaming transcends entertainment, merging with real-world economic and social systems.

For Developers

The advent of Web3 gaming opens up new opportunities for game developers, extending far beyond the confines of the current gaming paradigm:

- Deeper Markets and Trust Guarantees

: Web3 gaming creates deeper, more robust markets. Game developers can tap into a global audience where participants engage with a higher degree of trust. All in-game ownership and transactions are verifiable and immutable. This is thanks to blockchains' inherent transparency and security.

- Leveraging Existing On-chain Programs:

Developers can easily integrate existing programs and their associated assets into their game. This reduces time-to-market while enabling game developers to focus on creativity and innovation via ecosystem collaboration

- Economic and Gameplay Innovations

: Game developers can design intricate game and economic loops that leverage other on-chain applications or experiences. This interconnectedness allows for the creation of rich, dynamic game worlds where actions in one game can have ripple effects in another

- Encouraged Modding:

The openness of the blockchain gives game developers the chance to build games that are made in a way that promotes modding. This openness can lead to community-driven innovation, where players contribute to the game's evolution. This is a shift from a closed-source environment to a collaborative, community-centric development model

Blockchains offer more than just a platform to build games. They are conduits for pioneering new forms of interactive digital experiences. Blockchains pave the way for a more dynamic, participatory gaming ecosystem.

Why Build on Solana?

Low and predictable transaction fees are crucial for Web3 gaming. Gamers do not want to pay exorbitant fees to play a game. Solana has low transaction fees, often falling below $0.001 USD, with the average non-voting fee floating around the 0.000005 to 0.00007 SOL range. These fees are also predictable due to localized fee markets. They isolate congestion, ensuring that only transactions trying to access high-demand areas of state see increased fees. This differs from gas-based networks such as Ethereum, where transaction fees are unpredictable and can cost upwards of a hundred dollars for a simple transaction during heavy congestion.

Solana's architecture offers unparalleled composability. For players, this means a streamlined user experience with a single wallet. Players do not need to worry about the complexities of bridging to different platforms. Everything is accessible and interconnected within the Solana ecosystem. For game developers, composability presents a wealth of opportunities. Developers can effortlessly integrate and leverage existing on-chain programs (e.g., The Remnants and Magic Eden partnership). This ensures that liquidity for game assets is not fragmented across other L2s or shards. Solana fosters a composable ecosystem, facilitating the seamless collaboration of games, communities, and other applications on the network.

High speeds and low latency are essential for gaming. Slow transactions can hinder updates to game state, cause unbearable lag, and cause long wait times. This would provide a worse user experience than traditional gaming. Solana is designed for high transaction throughput. It averages 400 user-generated transactions per second (TPS), reaching over 2000 TPS during high demand. With innovations such as Firedancer on the way, Solana has the potential to scale to support any traditional consumer-grade application. Solana also boasts a 400 milliseconds target slot time (i.e., the time it takes for each ledger to ingest transactions and produce a block). It can vary between 500-600 milliseconds based on network demand, making it the fastest available blockchain. This makes it a perfect real-time database for genres such as strategy games, turn-based games, city builders, and idle games.

Forever Games are fully on-chain games. This is the most extreme case of blockchain integrations, where a game's state and logic run entirely on the blockchain. This results in a persistent game connected to a rich system of protocols, dApps, and programs with a low barrier to entry. Examples of this on Solana include:

- SolanaCiv: a 4x strategy game inspired by Sid Meier's Civilization
- Geoverse: a top-down resource management RPG
- The Heist: a high-stakes, risk-based idle game that is token-gated to its NFT holders
- Sol-Chess: a chess engine written in Anchor

As a gamer, a 400-millisecond response time sounds dreadful for real-time racing or shooting games. However, not everything needs to be put directly on-chain. Smaller integrations using NFTs to represent in-game items, using tokens for in-app purchases, and using a player's wallet for authentication, for example, can be done quickly. There are a plethora of tools available for you to start integrating with Solana, today.

Tools for Building Solana Games

This article examines several popular frameworks and platforms for building Solana-integrated games. For the sake of brevity, there are a few tools that we did not cover but have linked down below in the section for additional resources. These include tools such as the Godot Solana SDK or the Phaser 3 TypeScript Starter Template. For the purposes of this article, we examine how to install, configure, and use the following tools:

- Anchor
- Turbo
- Magicblock
- Solana Unity SDK

- SOAR
- Solana Unreal SDK
- Honeycomb
- GameShift

Anchor

Developing a game on-chain is made significantly easier with frameworks such as Anchor. Anchor is a robust framework used to build secure programs quickly. It simplifies the development process by reducing boilerplate, handling various security checks, automating client library generation, and offering a comprehensive testing setup. The easiest way to develop a fully on-chain game would be to store the game's logic and state in an Anchor program. Anchor helps by abstracting the complexities of Solana's lower-level programming model. Now, developers can focus more on game development than fighting with the SVM.

Installation

Rust is required to use Anchor. It can be installed here or with the following command:

Anchor also recommends those new to Rust read the first nine chapters of the Rust book. This covers the basics, as most of the time, advanced Rust knowledge isn't required to write Anchor programs.

Anchor also requires the Solana Tool Suite:

You can confirm your desired version of solana

is installed by running the command solana --version

.

Yarn is also required, which can be installed using the npm package manager: npm install --global yarn

.

Installing Anchor using the Anchor version manager (avm) is recommended. It can be installed using cargo: cargo install --git https://github.com/coral-xyz/anchor avm --locked --force

. Then, we can install and use the latest version:

The Anchor documentation also provides an in-depth installation guide.

Usage and Examples

SolanaCiv is developed fully on-chain using Anchor. Managing resources, recruiting units, researching new technologies, fighting barbarians, and trading happens all on-chain. It's also open-source! This means it can be used as a reference for breaking down game logic and state to see how everything is stored on-chain.

Clockwork is an open-source automation engine for Solana. This is particularly useful for game development as it facilitates transaction scheduling and building automated, event-driven programs. It can be installed with the following command: cargo install -f --locked clockwork-cli

and integrates nicely with Anchor. A list of examples of how to use Clockwork can be found here.

Solana Playground is a browser-based code development environment for testing and deploying programs. Its integrated development environment (IDE) makes it an ideal choice for developers new to Anchor who want to learn Solana game development. It has several Anchor tutorials relevant to on-chain gaming. For example, Tiny Adventure is a beginner-friendly on-chain game tutorial. The tutorial teaches you how to use Anchor to move a game character left and right and how to use a Program Derived Address (PDA) to save game data. It also shows how to set up a front end in Next.js using the Solana wallet adapter. The finished code can be found here.

To quickly create an Anchor program with a TypeScript client, use the npx create-solana-game

command followed by the name of your game. This command streamlines the Anchor and TypeScript development process, structuring a brand-new project. The tool was developed by Jonas Hahn and Beeman, and can be found in the solana-developers GitHub repository.

Turbo

Turbo is a next-gen 2D game engine and marketplace designed to empower indie developers. It includes several innovative features:

- Games hot-reload whenever code, shaders, or sprites are modified
- Games run headlessly so you can verify game state on remote servers
- The engine runs on WebAssembly and WebGPU so that it can run on any platform
- The engine's built-in APIs are beginner-friendly

Turbo is being built by Jozanza in Rust. Since it's being built in Rust, developers can easily reuse Anchor structs in their programs without needing any SDKs or plugins. The project is still currently in early development. However, its website contains several examples that can be executed in the browser, showcasing Turbo's power and intuitive developer experience. Also, a "Hello, world!" walk-through can be found here.

Installation

To begin using Turbo, we must first install all relevant Rust dependencies:

From here, we want to download and run the installer for the Turbo command line interface (CLI):

Once the installation is complete, we can use the turbo-cli -h

command to ensure the Turbo CLI is properly installed. With this command, you should see Turbo's usage information:

Development Process

Turbo has several macros to simplify the development process. As an aside, Rust macros are powerful meta-programming tools for code generation (i.e., code that writes other code). They are beneficial for reducing boilerplate and enabling more expressive code patterns.

The println

macro is a custom implementation of the standard println

macro. It formats a string and then passes it to the game engine's logging system:

The cfg

macro is used for configuration purposes. It exposes the config

function, which returns a 64-bit integer combining a pointer to a configuration string and its length. This is useful for passing configuration data from Rust to another environment, such as WebAssembly:

The init

macro is useful for game state management. It defines a struct and provides methods for loading and saving its state. It implements the load

and save

methods for the struct, using Borsh for state persistence:

The go

macro is used to define a game's main logic. It wraps a code block in a function that can be called from an external environment, such as WebAssembly. It also defines an entry point for the game logic and a run_snapshot

function for running the game with a snapshot of its state:

Turbo also uses the strike

macro from the struckstruck crate to enhance Rust's struct definition capabilities. Typically, nested structs must be defined separately. The strike

macro allows for defining a struct with its nested structs inline. This feature simplifies the code, making complex data structures more readable and concise. It also automatically applies Borsch serialization to the entire structure. And it propagates the top-level struct's derive macro to all nested structs.

Thus, the entire developer workflow for a Turbo project is as follows:

Solana Support

Turbo differs from other Rust-based game development engines as it will have full Solana support

. For example, it can pull account data from an RPC and decode it in only two lines of code It also automatically subscribes to account updates via WebSockets, so your data is always up-to-date:

Magicblock

Magicblock is a versatile open-source framework for building on-chain games. It enables anyone to make and play games on Solana or the SVM. It boasts a rich toolset that allows developers to develop fully on-chain games or integrate Solana into existing games. It includes the following features:

- Solana Unity SDK
- IDL → C# generator
- Native WebSockets
- SOAR (Solana On-chain Achievements and Rankings)
- Solana Mobile Stack integration

With all these features, Magicblock aims to abstract away complexity so developers can focus on creating the best user experience possible. The MIT-licensed framework is maintained by Magicblock Labs (i.e., Andrea and Peak).

Solana Unity SDK

The Solana Unity SDK is an open-source package with tooling to enable game developers to integrate Solana-based NFTs, tokens, and on-chain programs into their Unity games. The SDK leverages the Solana Unity Core, a .NET standard 2.0-compatible framework that integrates Solana with the .NET ecosystem. It contains many features, such as full JSON RPC API coverage, instruction decompilation, Solana Mobile Stack support, and native DEX operations.

A list of example games built in Unity with the Solana Unity SDK can be foundhere.

Installation, Configuration, and Usage

To create with Unity, download the Unity Hubhere. The Unity Hub manages multiple installations of the Unity Editor, allowing you to work on and create new projects across different versions. Before proceeding, ensure your system meets the following requirements:

- OS

: Windows 7 SP1+, 8, 10, 64-bit versions only; Mac OS X 10.13+; Ubuntu 16.04, 18.04, and CentOS 7

- GPU

: A graphics card with DX10 (shader model 4.0) capabilities

With Unity installed, we can now create and open a new project.

With your new project open, open thePackage Manager window. It is accessible in Unity's top menu, using Window > Package Manager

.

With the window open, click the add (+) button in the status bar. The options for adding the packages will appear. Then select "Add package from git URL" from the menu.

Enter the https://github.com/magicblock-labs/Solana.Unity-SDK.git

Git URL in the text box and click "Add." You can install a specific package version by entering https://github.com/magicblock-labs/Solana.Unity-SDK.git#X.Y.Z

and replacing X.Y.Z

with the desired version. All available releases can be foundhere.

With the Solana Unity SDK installed, import Samples

using the Package Manager inspector. This includes a sample scene you can reference or modify to integrate into your

Unity project. This scene is located in the Scenes

folder and consists of an example of wallet authentication and swapping tokens viaOrca. The Scripts

folder contains different script examples, including the script for Orca swaps. TheWeb3.cs script can be attached to any object in your project. This script is the main entry point for the Solana Unity SDK. It is a singleton that manages the connection to Solana, facilitating wallet logins, signing transactions, and more. The Prefabs

folder contains templates that store Object configurations. They can be used to create instances of the same asset that can be reused throughout your project.

The Magicblock YouTube channel contains tutorials on connecting Solana wallets to Unity, fetching a wallet's public key and balance, conducting tests with in-game wallets, and building a chess engine on-chain.

The Solana Unity SDK GitHub repository also contains a several example scripts, ranging from instruction decoding to building transactions.

Example Jupiter Implementation

In addition to Orca, Jupiter is natively supported in the Solana Unity SDK. The SDK offers utility methods to facilitate swap quotes and build and send swap transactions.

To perform a swap, we must first create an instance of IDexAggregator

, providing a default account. This instance will be used to interact with Jupiter:

Then, we define the tokens we want to swap. In this example, we'll exchange SOL

for BONK

:

Next, we obtain a quote for the swap. This is what the quote would look like for swapping 1 SOL

:

We can also display the route path of the swap for greater transparency:

Now we can generate the swap transaction, sign it with our wallet, and send it off:

Porting a Program to Unity

Developers can use the Solana Unity Anchor package to integrate Anchor programs into Unity. This package contains an IDL to C# Converter. The converter streamlines converting Anchor IDLs (i.e., JSON representations of Solana programs) into C# code for Unity projects. The integration process follows two primary steps:

- Installation

: the command dotnet tool install -g Solana.Unity.Anchor.Tool

is used to install the Solana Unity Anchor Tool. The -g

flag ensures the package is installed globally, eliminating the need to install this tool with every new project

- Code Generation:

the command dotnet anchorgen -i idl/file.json -o src/ProgramCode.cs

is used to generate the C# representation of the program. This command converts the specified IDL into a C# file, ready to use in Unity.

Developers can efficiently port Solana programs into their Unity environment following these steps. A step-by-step video walk-through of this process made by Jonas Hahn can be found here.

SOAR

Solana On-Chain Achievement and Ranking (SOAR) is a program designed to provide a seamless solution for managing leaderboards, achievements, players' profiles, and automatic rewards distribution on Solana. It supports several features, including:

- Leaderboard creation

- Achievement management
- TypeScript client support
- Interacting with the SOAR program through Cross-Program Invocation (CPI) calls

Support for the Solana Unity SDK is currently underway, and is expected to be coming soon.

SOAR can easily be integrated into a project using a few lines of code. For example:

The above code is directly from the Magicblocks documentation on SOAR. The snippet begins by importing the necessary modules from @magicblock-labs/soar-sdk

, including SoarProgram

, GameType

, and Genre

.

The SoarProgram

class gives the client access to every instruction in the on-chain SOAR program. It also includes functionality for deriving PDAs and fetching multiple accounts.

Then, it initializes an Anchor provider (anchor.AnchorProvider.env()

) and sets it as the default provider for subsequent operations (anchor.setProvider(provider)

). Next, a SoarProgram

client is created with the previously set provider.

The snippet begins initializing a new game. A new game keypair is generated, and essential game details such as the game's title, description, genre (e.g., Genre.Action

), and game type (e.g., GameType.Web

) are defined. An NFT metadata keypair is also generated for the game. Then, a list of authorizations (_auths

) is prepared to define access control within the game.

The initializeNewGame

method is called on the SOAR client, passing in the game's public key, title, description, genre, game type, NFT metadata, and authorizations. This method returns a newGame

object and prepares the new game transaction to be sent. The transaction is sent and confirmed using the sendAndConfirmTransaction

method available on the SOAR client. The game keypair is used as a signer for the transaction, ensuring that the transaction is authorized by the game's creator.

Initializing a client and setting up a new on-chain game with SOAR is easy. Now, we can use the GameClient

to register our game, create a leaderboard, and access our game's state. The possibilities are endless.

SOAR's program address is [SoarNNzwQHMwcfdkdLc6kvbkoMSxcHy89gTHrjhJYkk

](https://solscan.io/account/SoarNNzwQHMwcfdkdLc6kvbkoMSxcHy89gTHrjhJYkk#anchorProgramIDL) and its TypeScript documentation can be found here.

Solana Unreal SDK

The Solana Unreal SDK is a plugin and toolkit that allows developers to build blockchain games on Solana withUnreal Engine 5. The toolkit contains starting templates for Massive Multiplayer Online (MMO) Solana-integrated games, including a template for the game server. It also includes features such as:

- Full Solnet capabilities in C++ and Blueprint Visual Scripting
- An in-game wallet system with data protection and encryption

- Minting Metaplex NFTs
- Payment processing

The plugin leverages Solnet and the power of the .NET runtime so developers can communicate directly with Solana without creating an entire C++ library from scratch. LinkStream can send transaction messages from games to wallets, such as Tranquility, via TCP servers and client protocols. The Star Atlas team initially built the SDK, and Bifrost has forked and updated the code recently. For our purposes, we will be referencing the Star Atlas team's build for the next section on installation. The process is the same for Bifrost.

Installation

We must first install the Epic Games Launcher to use the Unreal Engine. Next, we need to create an Epic Games account and log in. Once logged in, navigate to the Unreal Engine

tab and click the Install

button to download the most recent version. A detailed walk-through of this process is available here. Note the system requirements recommended for Unreal Engine 5:

- Windows 10 64-bit, quad-core Intel or AMD, 2.5 GHz or faster, 8 GB RAM
- macOS Big Sur, quad-core Intel, 2.5 GHz or faster, 8 GB RAM
- Linux Ubuntu 18.04, quad-core Intel or AMD, 2.5 GHz or faster, 32 GB RAM

Open the Epic Games Launcher and launch your installed version of Unreal:

Create a new project and select C++ as the project template:

Now close the editor and open the Project Location

folder set when creating the project. Then, create a new Plugins

folder in the root directory:

Copy the plugin in the Plugins

directory. Right-click the .uproject

file in the root directory and select Generate Visual Studio project files

. Double-click the .sln

file and open the project with your C++ IDE. Then we want to open the file under Source//build.cs

. Add the plugin's name to PublicDependencyModuleNames

. In our case, we want to enter Foundation

:

Add the plugin's name to our .uproject

file:

Close your IDE and re-generate the project files. After open your project solution (i.e., the .sln

file), and the installation process is complete.

Usage

All plugin-related content can be found under Plugins > Solana SDK

in the content browser. A LinkStream blueprint can be found in Plugins > Solana SDK Content > Blueprints

. This blueprint enables developers to communicate with Solnet. The sandbox map found in the Maps

folder can be used as a reference for how to set up the Solana SDK properly. A breakdown of general usage and how to customize your LinkStream wrapper or how to interface with the REST API for server authentication is available on the Solana Unreal SDK GitHub. A more detailed breakdown of how to interact with Solana using the Unreal SDK can be found here.

Unreal's focus on optimizations and post-processing effects in 3D allows for creating photorealistic games compared to other editors such as Unity. A popular example of this hyper-realism is Star Atlas. Star Atlas is an immersive strategy game that takes you through the depths of space. It's filled with opportunities, territorial conquest, and political intrigue. Other games, such as Yaku Corp, leverage the photorealism of the Unreal Engine to create immersive MMORPGs.

Honeycomb

Honeycomb is a protocol developed by Tumi Labs aimed at streamlining the Web3 game development process. Honeycomb is a suite of on-chain programs and state-compression tools that handle crucial game lifecycle functions. All programs are designed to be interoperable and modular. These programs can also be customized to meet a developer's individual needs. Their infrastructure development toolkit is focused on the following core functions:

- Idle player rewards (staking)
- Quests (missions)
- Loot Boxes or RNG Prizes (raffles)
- Guilds / Party
- Game Tokens (custodial and non-custodial)
- Mutable Digital Assets
- Creating and Managing Digital Assets
- Player Profiles
- Fee Structures

Their tools can be broken into the following categories:

- [Hive Control

](https://docs.honeycombprotocol.com/concepts/hive-control): the master program that manages all the permissions and delegations of program authorities throughout Honeycomb

- [Asset Hub

](https://docs.honeycombprotocol.com/concepts/asset-hub): a suite of tools to create and manage in-game assets

- [Nectar Utility

](https://docs.honeycombprotocol.com/concepts/nectar-utilities): a suite of utility tools designed to allow NFTs to provide unique game features, such as idle token rewards (staking), quests, and loot boxes

- [Buzz Kit

](https://docs.honeycombprotocol.com/concepts/buzz-kit): a suite of tools to facilitate in-game guilds. Tumi Labs also plans to develop match-making and game state management programs under this toolset

Installation

Developers can interact with Honeycomb using itsJavaScript SDK and @solana/web3.js:

Example Usage

Once installed, developers can initialize a Honeycomb object with a Solana Connection object:

From here, the options are endless. For example, developers can interact with theIdentityClient to manage user identities and profiles. The guestIdentity

is the default identity used as a placeholder until a keypair or wallet is registered with the Honeycomb instance. A keypair can be linked as an identity with the following command: honeycomb.use(keypairIdentity();

. Similarly, a wallet object (Solana Wallet Adapter) can be registered as an identity using the walletIdentity

function available on the Honeycomb instance. Identities can be used for authentication purposes, such as signing messages or transactions.

Another interesting example is the HttpClient. This module facilitates HTTP requests usingAxios or Fetch, providing HTTP-

related functionalities. It allows for the customization of request headers and provides methods to make GET, POST, PUT, and DELETE requests:

This is particularly interesting as developers can access third-party APIs, such as theDigital Asset Standard (DAS) API to query other on-chain digital assets outside of the game.

These code snippets are adapted from the Honeycomb Protocol documentation. The SDK's documentation can be found here.

GameShift

GameShift is a platform launched by Solana Labs for game developers to integrate Web3 elements easily through a single API. It aims to facilitate the creation of user-friendly Web2 experiences while leveraging the benefits of gaming on Solana. GameShift provides a clean user experience within the Web3 ecosystem by handling all gas fees, blockchain interactions, and wallets. The GameShift API has four verticals:

- Asset Creation

: minting game assets with structured metadata

- Asset Trading

: buy and sell assets in USD in branded marketplaces, which are expected to launch soon

- Payments

: credit card payments and payment-processing integrations supported in 170+ countries, with a 100% chargeback protection

- Wallet Asset Self-Custody

: allows players to hold and trade their assets in non-custodial wallets (i.e., no passwords or seed phrases!)

Developers can get started with GameShift by creating an account on the GameShift console, setting up an organization, adding developers to their organization, adding a game, and managing their developer API key. The whole process is outlined here.

A merchant account is required for fiat transactions. An account can be made in two steps:

- Know Your Business (KYB)

: requires developers to provide information about their business to maintain compliance with money transmission laws. Currently, GameShift only supports fiat settlement to US-domiciled bank accounts

- Setting Up a Fiat Settlement Account

: developers must contact GameShift for a link to their merchant onboarding form once they are ready to enable fiat payments in production. The KYB information is set with this form. Validation can take a few minutes, but some cases may require manual review

All GameShift activity is organized within organizations:

Onboarding Users

Once everything is set up, developers can interact with the REST API. Developers can register users to grant them access to marketplace features, chain assets, and payment flows. This is done by making a POST HTTP request to https://api.gameshift.dev/users

using their API key, the user's email address, and a referenceId

(a unique value, such as a UUID) in the request body:

A user's information and assets can be fetched at any time once they're registered. To fetch a user's information and assets, a GET HTTP request can be made to https://api.gameshift.dev/users/{the-user-id-you-used-to-register}/assets

:

A paginated list of the user's assets will be returned, if successful:

All code snippets for the onboarding process can be found here.

Creating an Asset

As an example, developers can programmatically create on-chain game assets by making a POST HTTP request to https://api.gameshift.dev/assets

:

Assets require the following details:

- description
- a 64-character string providing information about the asset
- name
- a 32-character string to identify the asset
- imageUrl
- a direct link to the image that represents the asset
- displayType
- an optional formatted title
- traitType
- an optional name for the trait
- value
- an optional field that represents the attribute's value

The POST HTTP request also requires a user's referenceId

provided by the destinationUserReferenceId

field. A more detailed breakdown of working with game assets can be foundhere.

Conclusion

Integrating blockchain technology in gaming, particularly through Solana, represents a significant leap forward in how we perceive and interact with digital worlds. Tooling such as Anchor, Turbo, Magicblock, and the Solana Unity and Unreal SDKs enable developers to create new ways to game. It is easier now than ever for gamers to develop virtual economies and interact in unprecedented social matrices, in a completely transparent and accessible way. Platforms such as Honeycomb and GameShift further exemplify the potential of Web3 gaming by streamlining the development process and enhancing player engagement.

Solana is at the forefront of this gaming revolution. Solana's gaming community is growing steadily, with dedicated hackathons and new games appearing. Solana Labs and Solana Foundation lead initiatives, creating tutorials, tooling, and platforms to help grow this budding community. The possibilities for what gaming can become are endless when building on top of the fastest blockchain available.

If you've read this far anon, thank you! Be sure to enter your email address below so you'll never miss an update about what's new on Solana. Ready to dive deeper? Join our Discord to start building the future of gaming on the most performant blockchain, today.

Additional Resources / Further Reading

- Coin Flip Game with VRF Randomness
- Dart and Flutter Applications / Libraries for Solana
- Godot SDK
- On-chain Game Development Workshop by Superteam Germany
- Phaser 3 TypeScript Starter Template
- Solana Game Examples GitHub Repository

- Solana Playground
- Verified Unity Asset Store Listing for the Solana Unity SDK