StarkBots ✨

by Grillo

Introduction

DarkForest and **0xMonaco** have led the way and shown that blockchain programming games can be fun, exciting and produce amazing outcomes when thousands of players interact with them. But making this type of games **mainstream** might prove to be challenging.

StarkBots (name not final) is my attempt of describing a competitive programming game that can run on a public blockchain and be played with economic value at stake at different skill levels. **The key idea behind the project is to create a visual programming interface which would allow players to build their strategies as a set of interconnected, composable NFTs**. My view is that this innovation will solve or help mitigate most problems that a competitive programming game might face when running on a public blockchain, enabling **true mainstream adoption**.

All thoughts in this document are a work in progress.

Goals

A. Make it a live, social experience

StarkBots should have a GUI where matches are played in real time. This will enable social interactions such as streaming, commenting, rooting and even betting on live matches. The main technical implication of this goal is that matches should, to some extent, be non-deterministic.

B. Keep it within clear game boundaries

While CTF games usually celebrate hackers exploiting the game logic itself to gain advantage, **StarkBots should aim to limit competition to the strategies built within the intended design**. This is meant for the game to target casual and enthusiast developers as well as elite hackers.

C. Integrate sustainable economic incentives

Reputation and fun are good incentives, but they are both fully achievable on a Web2 infrastructure. **StarkBots should implement a full Web3 architecture, with matches having some economic value at stake.** Finding the right economic design is crucial to incentivize players while avoiding ponzinomics.

D. Make it playable at various skill levels

At the very same moment an economic competition exists, elite coders will tend to monopolize the games, leaving most players behind. **StarkBots should aim to split players by their skill level.** This is especially challenging when the game is kept **pseudoanonymous** and **permissionless**, but it can be attempted on an economic and game theoretic level.

E. Diminish the no-privacy issue

While stored on a public blockchain, a successful game strategy might be copied and redeployed by anyone. **StarkBots cannot provide full privacy for player's strategies, but it should make them hard to copy and even harder to take advantage of.**

F. Fully serverless architecture

StarkBots should use the blockchain as its only backend. The game should never depend on a server being online for any of its functions. Live player interactions should remain fully P2P.

The Game

StarkBots is a competitive multiplayer game where players build robots, code their strategy, and make them battle against each other in real time deathmatches.

The game is mostly inspired by a classic Java programming game called **Robocode**, in which players coded tanks and made them battle in deathmatches, and the late 90's TV Show **Robot Wars**, where physical robots where actually controlled by RC, but the customization of weapons and fighting styles played a central role.

StarkBots will use NFTs to represent different robot components (body, wheels, radar, weapons, etc.) which players will combine to ensemble their own battle bots. These bots will be controlled by a program built with a **visual programming language**, also based around NFTs (see next sections).

Robocode (https://robocode.sourceforge.io/)

Robot Wars (https://en.wikipedia.org/wiki/Robot_Wars_(TV_series))

Once a player has finished assembling a StarkBot and coding its logic, he'll be able to join deathmatches, in which he will risk damaging his bot, while standing to win loot from other participating players. **Battles will take place over the browser, with players interacting through a WebRTC P2P connection. The blockchain will serve as a backup execution environment and a security guarantee in case of player misbehavior.**

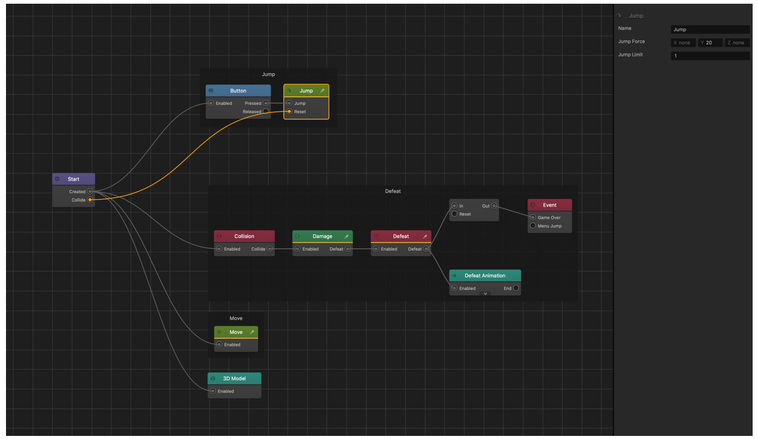
The objective of the game is clear: build the best StarkBot, code the smartest strategy and win by being the last bot standing on the battlefield!

Visual Programming

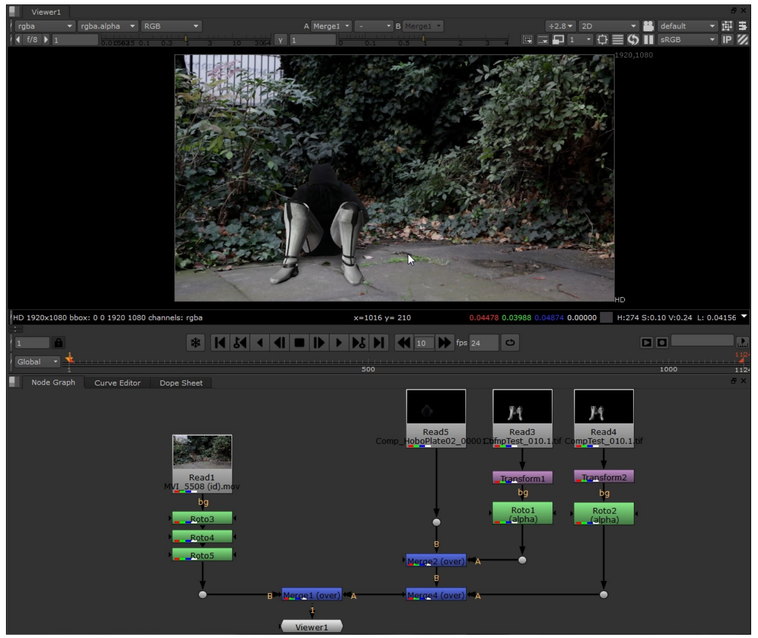
Implementing a Visual Programming Language is most likely the key innovation of StarkBots. While not evident at first, using visual programming instead of deployed contract code will allow us to broadly widen the target audience of our game, set apart players by their skill levels, have fast paced off-chain battles and even implement an original and sustainable business model.

Visual Programming Languages have existed for many years and are widely used in 3D modeling software like **Houdini** and **Blender**, post-production software like **Nuke** and kid-friendly game engines like **BuildBox** or **Scratch**.

The main feature of a Visual Programming is that no code needs to be written. **StarkBots will use a subset of Visual Programming**

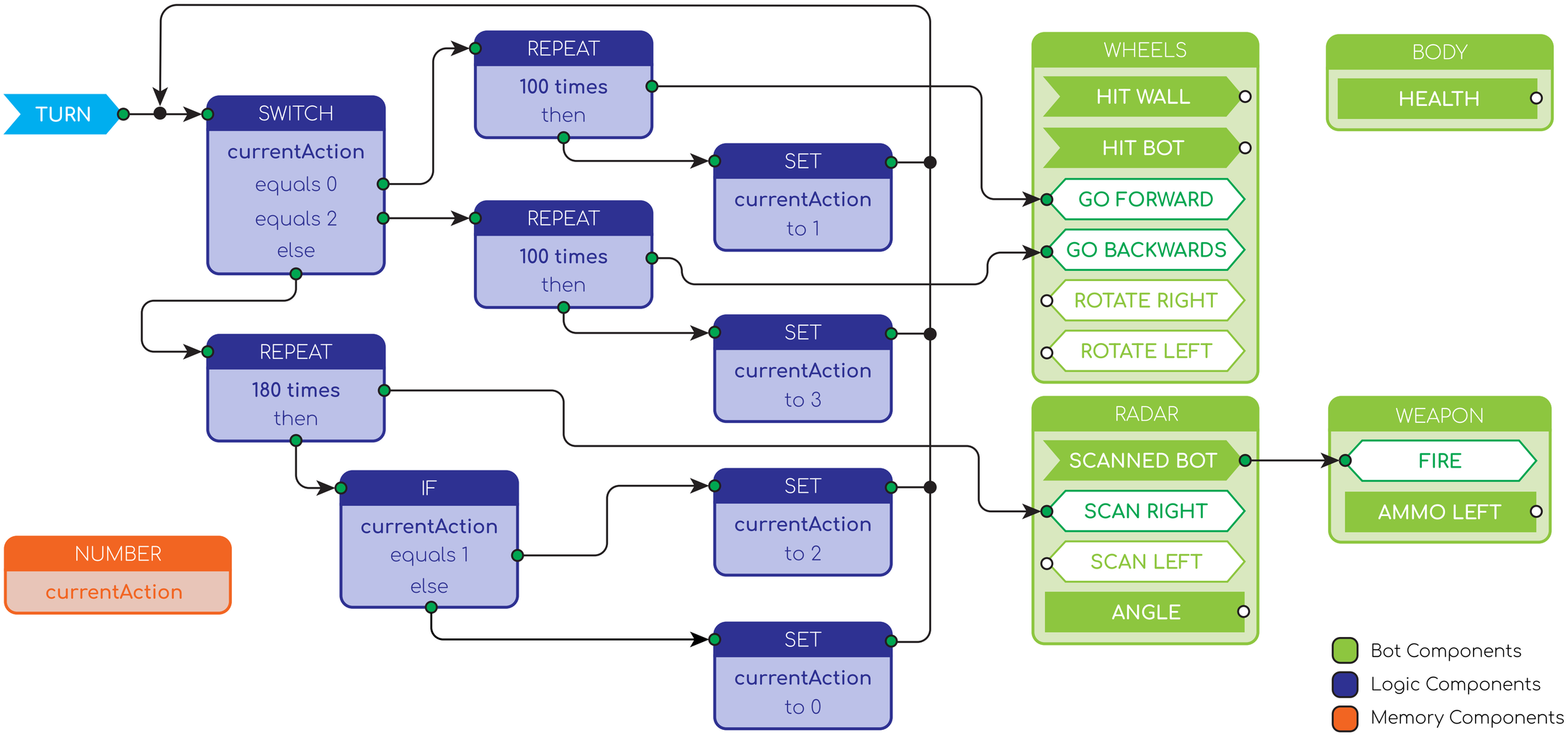


BuildBox (https://www.buildbox.com)



NukeX (https://www.foundry.com/)

Languages known as Dataflow Programming, where a process is represented as a set of interconnected, configurable nodes. Here's an example of what a StarkBot program might look like:



The example shows the program of a bot that would move forward for 100 turns, then rotate the radar 360° degrees (at 2° per turn), then move backwards for 100 turns, rotate the radar again, and repeat. If the radar were to ever scan an enemy, the weapon would fire.

Bot Components (in green) represent the "physical" parts of the bot and would have a graphic representation on the game as it's being played. Combining these components should allow for widely different bots to be designed. Each Bot Component has different events, actions, and variables that the program can interact with. **Logic components** (in purple) make up the actual programming logic of the bot, connecting events to actions. In this bot, most logic is executed on **the TURN event** (light blue), which is fired once every turn. Finally, there is a single **Memory Component** (in orange) which creates a numeric variable, globally accessible by all Logic Components.

The system should be simple enough for non-devs to attempt to learn it and composable enough for advanced players to build complex logic on top of it, using thousands of different components. But the true advantages of using a visual programming language really begin to show when we link these components to ERC1155 NFTs and create an economic model on top of them. We'll explore this in the next section.

It should be noted, at last, that designing this visual programming language is probably the biggest technical challenge that StarkBots will face. **Picking the right components to start, creating a roadmap of future components and involving the community in the development of missing components will be key to the game success.**

Economic Model

A. Fun, Work and the Play-to-Earn model

Before starting to describe **StarkBots'** economic model, it's important to understand why an economic model is actually required for our game. In discussing play-to-earn game mechanics, famous Crypto Twitter OG @cobie once made a famous comparison:

When taking into account that most DeFi "staking" pools have nothing really at stake, this comparison actually means that, **in a game theoretic analysis, play-to-earn games should be considered ponzi games**, where early participants get the most benefit, initial profits come from new participants joining in, and the only true winners are the ones who exit before the collapse.

On the game developer side, this implies that no productive enterprise needs to be attempted, as profits will come from having supplied the team with a big share of token emissions. **Their business is about creating a good enough narrative for people to hop into the ponzi game, not (necessarily) about producing a long term viable product.**

Now, there's a second @cobie analogy which I find to be somewhat less accurate:

Play-to-earn games could be comparable to simple jobs, but:

1. Walmart is a profitable business that pays their employees with their profits, not with a self-made token that their employees are simply farming to try to sell to a greater fool.
2. The comparison seems to imply that if a game is not played for fun, then it's not a game at all, which totally misses the many similarities between games and work. In fact, **from item grinding to literal work simulation games, the line dividing games and work is not as clear cut as it seems**.

So, the question we should ask ourselves is: What is the actual game being played? If we follow the yield farming analogy, we could say that most play-to-earn games are ponzi games. And what is the problem with ponzi games? To me, the most important answer is quite simple:

Most people playing the game don't know what game they are playing.

If all players knew the game rules and competed over skill-based techniques to find early opportunities and exit timings, the game would be defensible. But the truth is that only a few people know and play by these rules, while most players get in with false expectations and are actually playing a chance game. This is why ponzi games are not actually fun for most players: if you are not improving your skills as you play, you are actually playing at random, like in a Casino slot machine. A player might get addicted to these type of games, but is he having fun?

My view is that the work comparison has nothing to do with the lack of fun. In fact I think the opposite is quite possible nearer to the truth: a game needs to feel like a good career job in order to be fun. Players must feel that the work and the time they spend on the game makes them better, which gives them a better chance of winning. And above all, players need to know which game they are playing and the game dynamic cannot rely on luck or chance alone.

Having said all this, **my aim when designing the economic model for StarkBots was to allow the players who actually "work" by learning to play and getting better, to be compensated with actual profits and not mere token emissions, thus avoiding the usual ponzi game**. The first step to achieve this was to find some value model for the NFT Components described in the Visual Programming section.

B. Creating value

If we want to avoid simply selling hype and speculation, we must start by asking ourselves how to create value in crypto. The three models that have inspired our own model are:

1. **Limited Supply Assets with Proof-of-Work Mining**
2. **"Infinite" Supply Assets with Proof-of-Stake slashing and EIP-1559 like burning mechanics**
3. **"Backed" Assets with Protocol Controlled Value (PCV)**

Starting with Proof-of-Work Mining, we have taken the concept of **mining difficulty** and decided to use a similar benchmark. In our case, the benchmark used will be the **daily number of games played**. Initially, a target will be set up manually. **The NFT Components used to build and play the game will become harder to get as the number of games played rises above the target, and easier as the number falls below said benchmark.** In time, this target could be algorithmically controlled, following a mechanism like RAI, trying to stabilize the number of games played over time.

In contrast to limited supply proof-of-work models, we don't want to limit the supply of NFT Components, as we can't know beforehand how many components players will use in average to play the games. That is why **StarkBots won't technically limit the supply of Components, but it will implement a game of chance (like mining) to mint them**:

1. NFT Components are minted randomly according to their given rarity.
2. When a user tries to mint NFT Componenents, he doesn't know how many or which Components he will get. Everything depends on the rarity, the current difficulty and a VRF function call.
3. Let's say the average price of a Component is $1. If the daily number of played games is below target, a player might spend $10 USDC and receive 12 or 15 Components. If the number is above target, he might only get 5 or 6.

This mechanism is meant to boost activity when not many games are being played and to promote people actually playing the game instead of holding them for speculation: if players only held the tokens, the number of games would decrease, to which supply growth would follow.

Now, if NFT Component supply is technically "infinite", it becomes greatly important to include a slashing and burning mechanism to limit supply. In fact, that's what playing the game actually does. **When a player builds a bot and plays against other players, he is actually staking his components. If he wins, he will keep all of his Components and will get some Components from his opponents' bots. If he loses or**

if he is found to have cheated, he will lose some or all of his own Components. Random functions will be used to decide which Components of a loser bot get burnt, which get transfered to the winner (as loot) and which he gets to keep. If the profit of winning a game comes from another player losing his stake, the game technically becomes a zero-sum-game, but in **StarkBots** every player involved knows which game they are playing. **A "friendly battle" or "practice" mode could also be created (without risk), but these games shouldn't be counted towards the daily games played benchmark.**

There's a final problem to handle: if the market prize of Components was to fall so much that players become unmotivated to play, the random minting of Components linked to number of games played might create a death spiral situation. In order to avoid this, our plan is to implement the "backed" asset model: **A large portion (90%+) of the value used to mint Components will be considered Protocol Controlled Value**. This means that if the price of Components was to fall too much, buybacks below backing prize might be implemented to push the prize back up. In this way, if StarkBot becomes even mildly successful, the PCV would quickly grow to be much larger than the market prize of all available Components. **The remaining portion of the minting prize would be considered protocol revenue.**

C. Secondary benefits of the model

The described economic model is meant to give NFT Components some actual value, which also allows the game to reward winning players. **But the model also creates an important side benefit: if players are actually staking their components when playing the game, a player using thousands of components will have more value at stake than a player using only hundreds of components. If we consider that the number of components can roughly be considered equivalent to the bot complexity, we have a tool in our hands to separate players by their skill level**: players will logically want to play against players with a somewhat similar value at stake.

Taking this idea further, we decided to add a power metric to all Components. By doing this, we can estimate the "objective" power of a bot, limiting battles to different power ranges. In the next section, we'll explain how this limit will be implemented.

D. Other sources of revenue

The main source of revenue for StarkBots will come from the fees generated when selling NFT Components, but other sources of profit or revenue might also be considered. Most notably, we must mention:

1. Vanity Components

A set of components without any gaming benefit might be sold directly to interested players. These components would be sold purely for revenue, and would allow players to change their bot colors, or add sound or visual effects of their bots as they are playing the games.

2. Fuel

I have considered creating a token to work as burnable Fuel for the bots, making longer games more expensive than shorter games and creating a variable deadline for battles (when no player has won the game and every player has spent all their provided fuel). Though an interesting possibility, I have momentarily decided not to implement it, so as not to increase complexity that much.

3. Governance Token?

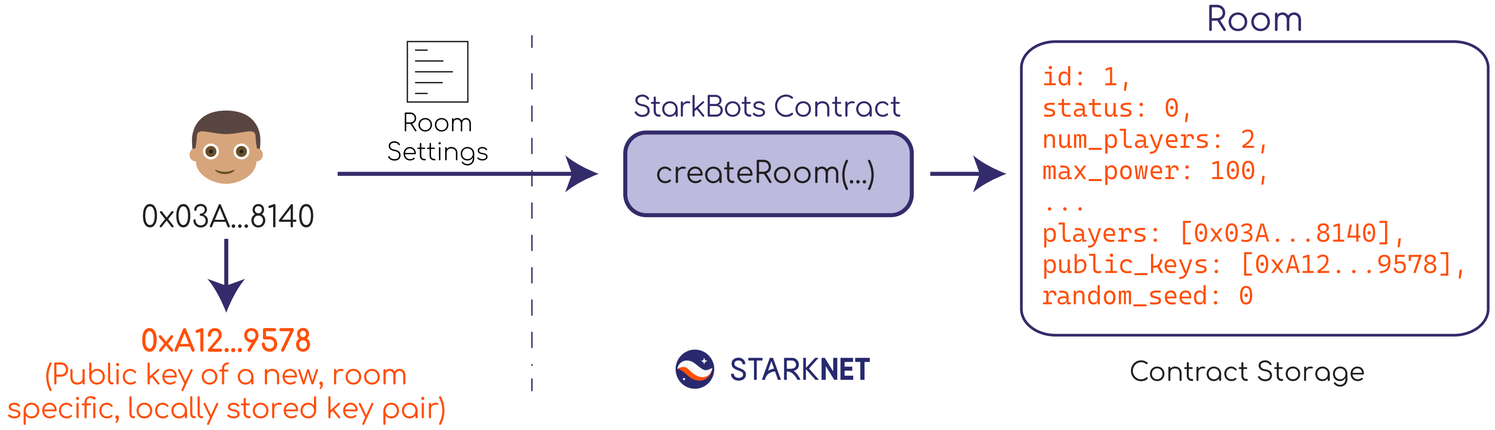
We have previously considered how token speculation lead to ponzi games, which we wanted to avoid, but given how our protocol might build a sizable treasury, a governance token might be considered in the long run. If this was to be seriously considered, the #1 priority should be to separate the game economy from the token speculation.

Player Interaction

StarkBots aims to be playable over the browser through WebRTC P2P messages with **StarkNet** providing security guarantees and backup execution. **Player interaction was designed to allow fast, live, and trustless interactions, while also reducing privacy concerns and remaining fully serverless, so that games can continue to be played for as long as a frontend is available and StarkNet is online.**

A. Main Player Interaction System

1. Considering a player already owns enough Bot Component NFTs to build a robot and the Logic and Memory Components needed for his desired program, he can create a game room. Game rooms are created on-chain by providing some room settings:

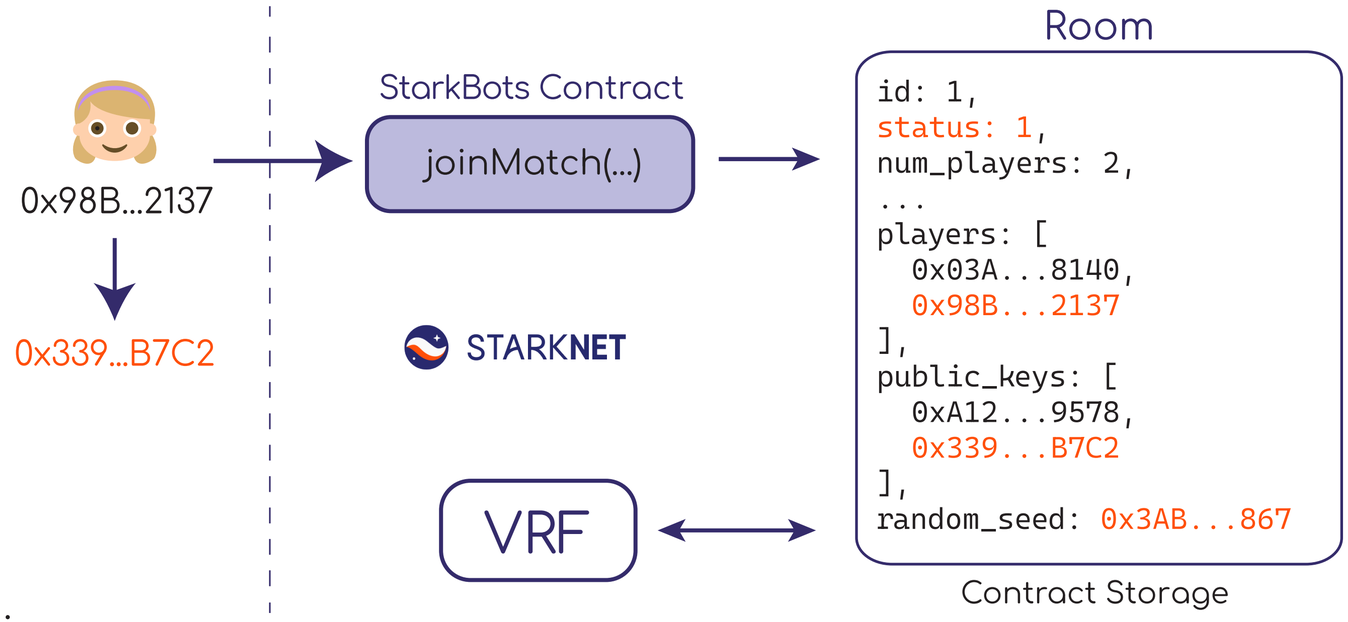


Especially important within the provided room settings, are two values:

- **max_power:** As described in our economic model, Component NFTs won't only have some market value, but also a preset power value. This variable filters the allowed bots by their total power (the sum of the power of all of their components). If max_power is set to 100, no bot with over 100 power points will be able to join the game. Some players might choose to use more powerful Bot Components, while others might chose to use more Logic components to create a smarter logic. **This is how Skill Level segmentation is achieved.**
- **public_key:** when creating a room, the system also creates a new public key pair, stored locally on the player's cache, and meant to be used only during this match. The public key provided will be used to verify off-chain player actions without requiring players to constantly interact with their wallets.

2. Other players can now join the room until the number of players reaches the **num_players** setting. The game status then changes to 1 and a VRF source is used to fetch a random seed for the game.

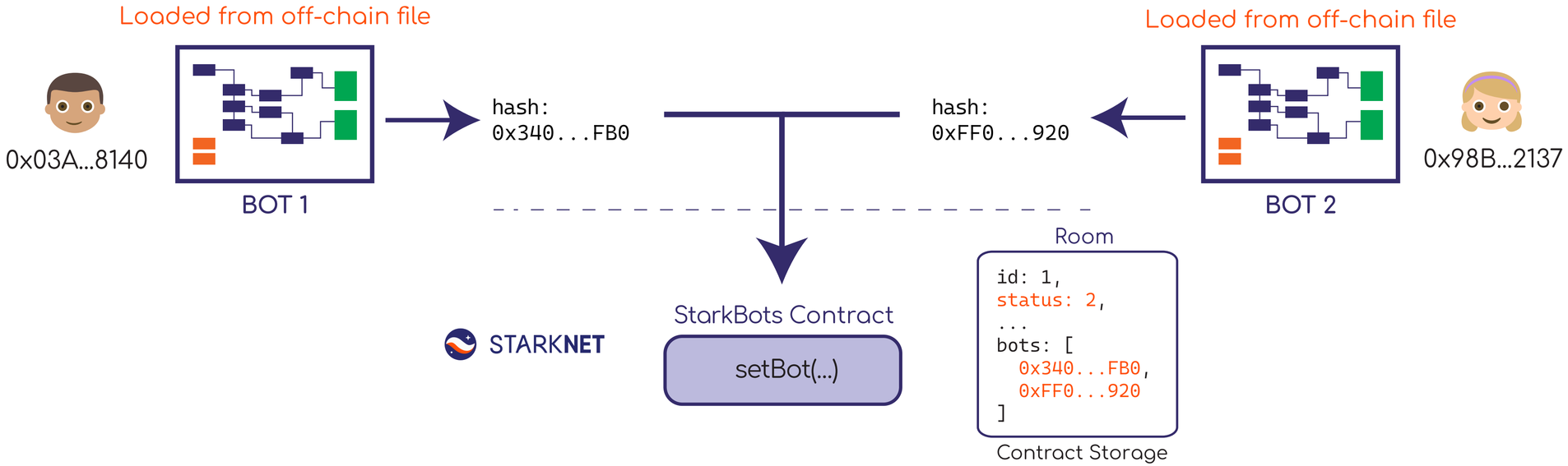
Players immediately begin to connect with each other through WebRTC, verifying their identities by signed messages. In our tests, we've used the Trystero Serverless WebRTC matchmaking Library, which uses BitTorrent or IPFS to connect players **without relying on a central server**.



3. Once the game room is full, the random seed is used to create an algorithmic map. All players calculate this map off-chain from the shared seed and sign a hash of the map. The game cannot continue until all players have every other player's signature attesting that the hash is correct.

4. At this point, players have still not chosen which bot they will use or which logic they will implement. **This is meant to mitigate the privacy issue of advanced players deploying specific bots in response to the ones chosen by the other players.**

After all players have signed the map hash, they can begin customizing their bots. Players are not meant to code their bot logic on the spot, but **configuration files should be stored off-chain, so that no advantage can be taken from knowing the possibilities beforehand**. For the game to continue, all players must choose their bot and publish a hash of it on-chain before reaching a deadline.

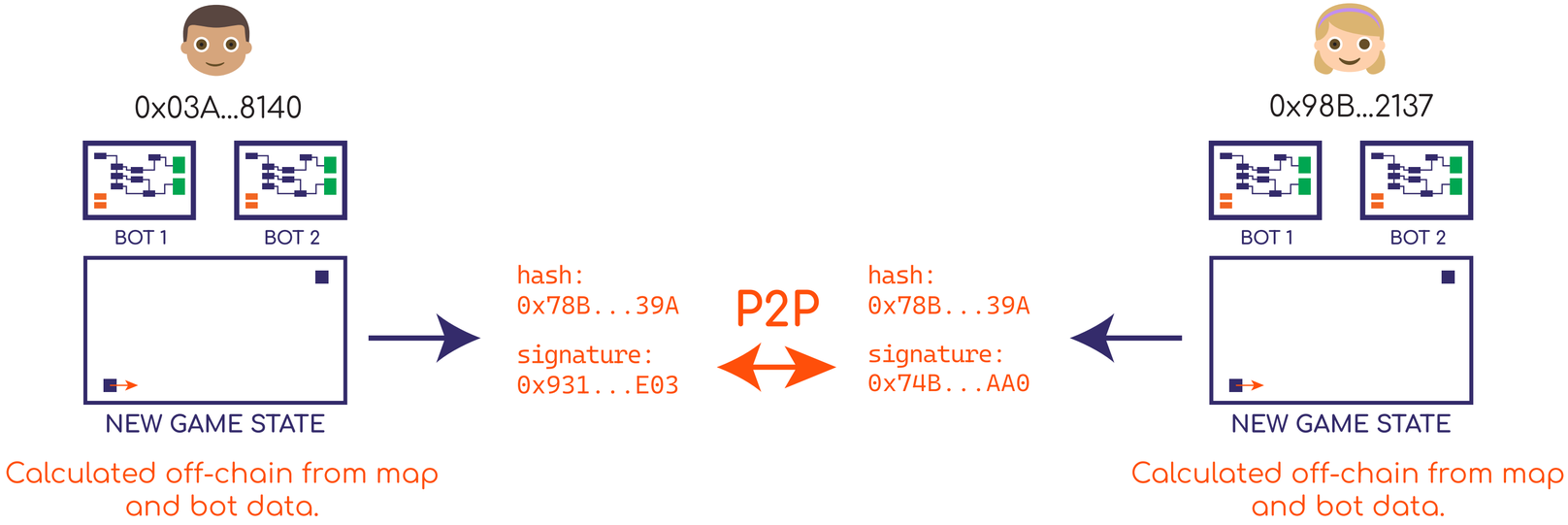


Once all players have provided their bot hash, the room status changes to 2 and is ready to start.

5. **From this moment onwards, all Component NFTs from participating players get locked.** They will only become unlocked once the game ends.

Players start exchanging their full bot information. Every player verifies the hash, the max_power limit and the ownership of all used NFT components. Any inconsistency on this step can be verified on-chain. A cheating player would get punished. By the end of this process, every player will know the complete configuration of every other player's bot.

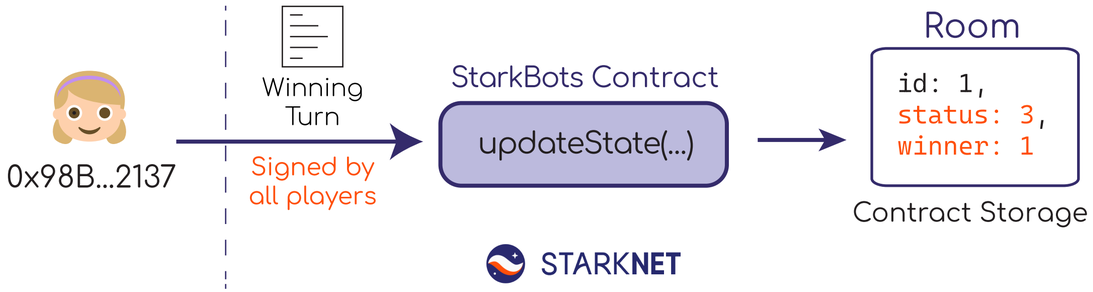
The game can now be played fully off-chain, with each player calculating the next turn on their own and sending the signed, resulting game state to the other players. **Players only continue to the next turn if they have signatures from all players of the previous turn state.**



With the whole game being simply calculated off-chain (without even querying the blockchain), **the gameplay will feel smooth, almost real-time**. It must be noted that:

- **Players keep a record of all signed turns.** At any moment they want, they can publish a signed game state on-chain to update the on-chain game status.
- **Randomness will be required to keep the game non-deterministic:** Calculating the hit power of a given shot, for example, should depend on a common random seed. Changing this seed throughout the game will then prevent the game from being fully deterministic from the very beginning. To achieve this, an online VRF source could be agreed upon or players could commit to a set of random numbers before starting the game. (See the On-Chain Backup Player Interaction to see how this is handled on-chain)

6. Once a winner is decided, the winning player can simply publish the last, fully signed turn on-chain to finish the game. **Awards and loots are distributed according to the game rules.**

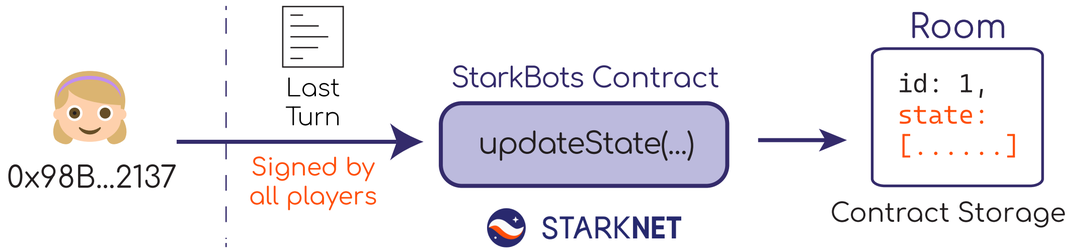


B. On-Chain Backup Player Interaction System

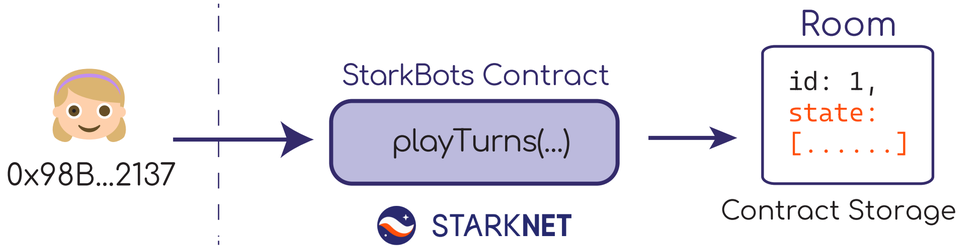
StarkBots can be played fully off-chain, but only with some cooperation from all players involved. **In case any player misbehaves or is simply offline and cannot sign messages, any player can decide to finish the game on-chain.**

1. Playing on-chain requires at least that players have at least agreed upon a map and a set of bots. If that's not the case, the game room will simply time out.

2. A player can begin playing on-chain from turn 0, or he can load the last state available with a complete set of valid signatures:



3. Then, the game can be played on-chain by simply calling a function. Only a set number of turns is allowed to be played per game, per block to keep the game non-deterministic.



4. Randomness in on-chain games cannot be equal to off-chain randomness. Either a VRF source is used for every new block, or the block hash and the original seed are combined to create new seeds every X number of turns. **This means that on-chain games might have different results than off-chain games, but they should be equally fair.**

Putting it all together

Now that all general **StarkBots** game concepts have been described, let's see how our design decisions help us achieve the goals we set up to achieve:

A. Make it a live, social experience

Our proposed game interaction model would allow players to trustlessly play between each other through the browser while remaining non-deterministic. This means that **games would take actual time to finish, and winners would remain undecided until the very end**. This would even apply when played on-chain.

B. Keep it within clear game boundaries

Having all programming logic encoded as a set of interconnected NFTs over a visual programming interface will allow StarkBots to remove or greatly constrain unintended game logic attacks, like hidden memory accesses or action replaying. Regardless of that, games will surely become more sophisticated and complex over time, as Logic Components multiply, and new strategies begin to be tested.

C. Integrate sustainable economic incentives

Our economic model is simply meant to incentivize players to play the game and avoid useless speculation. Our thesis is that this model will be crucial for mainstream adoption, motivating players to get involved and learn how to play rather than betting on fast profits. The model would also allow **StarkBots** to produce revenue, while growing a treasury and helping separate players by their skill level. Various other sources of revenue have been considered.

D. Make it playable at various skill levels

By having a measure of the total power of a bot (including not only the strength of its body and weapons, but also the complexity of its programming logic), **StarkBots can roughly estimate the skill level of a player**. Advanced players might create widely complex logic with thousands of components and lots of value at stake, while novice players can still play and learn on low-power rooms. If the game is successful, this design should lead to players specializing on different power ranges (in the same way as weight is used in boxing).

E. Diminish the no-privacy issue

Our player interaction system makes sure that players don't have enough information about other players' bots before committing to their own selection. This design choice **eliminates the main problem of running the game on a public blockchain: advanced players choosing their bots in response to their rivals**. With bot configurations stored off-chain and components having limited supply and some economic value, **the possibility of copying a successful game logic is reduced**, but not completely removed.

F. Fully serverless architecture

All described player interactions can be accomplished without a backend server. StarkBots will be playable for as long as frontends are available, StarkNet is online and other players are availble and willing to battle.