Disclaimer

The project is for the sale of ART COLLECTIBLES in the form of NFTs (non-fungible tokens) only.

This document is intended for INFORMATION PURPOSES ONLY, and may not be incorporated into any contract. Nothing herein constitutes legal, financial or investment advice. It is not a promise, binding agreement, or legal obligation to deliver any material, code, or functionality, and should not be relied upon in making purchase decisions. The development, release, and timing of any products, features or functionality is solely at our discretion, and subject to change.

Your NFTs should not be used for investment or speculative purposes.
# Table of Contents

Part 1: Background 1

Part 2: Technical Decisions 3
  Platform 3
  Game Criteria 3
  Game Type 3
  WebRTC 4
  Blockchain 4
  Solana 5

Part 3: The Game 7
  Gameplay 8

Part 4: The NFTs 9

Part 5: Architecture 10
  Overview 10
  The PDA Accounts 10
    Game State 10
    Round State 12
  Frontend 13

Part 6: Building atop AvsP 15
  Solana Programs 15
  Web Applications 16

Part 7: Legal Considerations 17
  Strategy 17
  Rewards 17
  Timeline 18

Part 8: Changes 19
  Technical Limitations 19
  Feedback 19
  Improvements 19

Part 9: Conclusion 20
Part 1: Background

Aliens VS People ("AvsP") spawned from a simple idea: NFTs shouldn't just be functionless JPEGs. They should provide some utility. Some enjoyment. Something that keeps NFT holders coming back for more. Something that not only excites holders, but also motivates the entire blockchain community to build projects that offer more than just JPEGs.

Hence the decision to make a game. Preferably, an entirely on-chain game (which we managed to build!). A game that will be as timeless as Monopoly or Battleship. These games will be enjoyed by people hundreds of years from now, just as much as they were enjoyed by people almost a century ago. These classics have been played for generations for nothing more than the enjoyment they bring. Think of your favorite games. How many of them give you something of value beyond the game itself? We want to build a game that is equally timeless. Not something you play just to earn rewards, but something that is so irresistibly fun that the game itself is the motivation to keep coming back.

Equally, the game should be enjoyable by everyone. From kids to their grandparents. By people from all walks of life. By people in every inch of the world. Blockchain is for all and inclusive; it knows no gender, race, geographic boundaries or age. If our game is to truly embrace the principles of blockchain, it too must be for all.

With an entirely on-chain game, the possibilities are endless. We can build an entire ecosystem, supported by third-party developers who build new versions and features atop ours. Not only would it benefit our NFT holders, but also, potentially provide a source of income for developers.

Why allow third-party developers?

The best products aren't closed. They're built atop an open ecosystem. Think of Microsoft, Apple, or even Ethereum and Solana. They all have ecosystems that encourage participation from the community and/or third-party developers, for the benefit of all stakeholders. The combined creativity and skills of thousands of participants will always surpass that of a small bootstrapped team.

Just the same as the most successful products and projects today, we want to open our game for third-party developers to build atop. Ultimately, this benefits everyone involved. It will make the project a lot larger than what we could possibly have achieved on our own.
But in short, our criteria was simple:

1. The game should bring enjoyment in and of itself
2. The game should preferably be on-chain (but definitely decentralized, transparent and immutable)
3. The game should be playable by anyone and everyone

The answer?

Aliens and people are embroiled in an intergalactic war. In a Battleship-style game, the two sides must destroy each other’s fleet. Winner doesn’t go extinct, d’oh!

But before we get into specifics of the gameplay, let’s first discuss the technical and legal limitations that defined our decisions.
Part 2: Technical Decisions

**Platform**
Let’s start with why Solana.

If our goal is to make an on-chain game, we need fast transaction times, low gas fees and capacity for a large number of transactions per second. Solana hits the mark.

There are, of course, several other major benefits – but we won’t bore you with the geeky details that get us excited. Suffice is to say that we were impressed by Solana, and wanted to be an early part of its ecosystem.

**Game Criteria**
There are several technical criteria that our game would need to satisfy:
1. **Secret State:** Users must be able to submit their position (e.x. 2x5 on a grid) without it being readable by everyone else
2. **Multiple Players:** The game should be playable by multiple NFT holders simultaneously
3. **Transparent:** Anyone should be able to view the raw gameplay and build their own frontend around it
4. **Decentralized:** No central authority should be able to control any element of the game
5. **Immutable:** Developers and players cannot manipulate or falsify the gameplay

**Game Type**
If we don’t use state channels, an MMORPG game would be a dreadful experience.

Imagine every action - every move, every kick, every punch - having to be pushed to the chain before it’s reflected in the gameplay. There couldn’t be real-time interaction between users.

If we want to be entirely on-chain, we would need a strategy-based game where waiting is the expectation and doesn’t detract from its enjoyment. Not something that has a kick or punch that takes minutes, or even seconds, to reflect in the game.

So, can a game that uses (off-chain) state channels still satisfy our initial criteria?

Let’s discuss.
**WebRTC**

One possibility in building a peer-to-peer game is skipping the blockchain altogether. Instead, using WebRTC, which permits direct communication between players. This is the approach many purported “blockchain” games have used.

Typically, a game server (assumedly ours) would be needed for signaling (to connect all parties), and possibly serve as the single source of truth. Without it, each player would need to be the authority for their own moves, thereafter broadcasting it to everyone else and hoping they update the game states (even if they’re going to lose).

With the help of a hash (“Merkle”) tree, we could store the game state on the blockchain and prevent cheating. Using state channels would reduce the number of on-chain transactions, but there still remains a multitude of issues that make WebRTC an unideal solution for our needs.

Specifically:
1. The need for a **central** signaling server to connect the players
2. Peer connection limits. For example, some browsers have soft limits for the number of simultaneous connections. Even then, the bandwidth requirements for a session with a lot of players would make the game unplayable for those with lower bandwidth
3. Privacy concerns, such as WebRTC leaking real IP addresses – which could then be linked to the players’ wallet addresses holding the NFTs
4. Without an entirely on-chain game, third-party developers can’t build features atop ours to grow the ecosystem

There are certainly benefits to using WebRTC and state channels. However, noting the issues above, it just wouldn’t work for us.

**Blockchain**

Before we can begin our on-chain game, we need to consider several caveats.

First, given the transparent nature of blockchain, players’ positions would be immediately readable by everyone. Woops!

What if users cryptographically hash their location with SHA256? Problem is, with a limited number of possible locations and SHA256’s deterministic output, one could easily determine the hashes of all possible points. So, a user’s position could be determined from simply looking at the corresponding hash. Ouch.
What if we add a “salt” to the hash? Basically a secret user-generated key, which is coupled with their position to generate the hash. But, this secret key would still need to be sent to the smart contract to decrypt the position. When sent on the blockchain, this secret key would then be known to all participants. You can see the issue.

What if we first let users hash their position with their own secret key, submit the hashed position to the smart contract, then their guesses about others’ positions, and finally the original secret key to decrypt their position? The smart contract can then determine the users’ original positions, while, already having the guesses so that they can’t be changed. This is called a “commit/reveal” scheme. Now we’re on to something.

We of course need to add other game mechanics to prevent cheaters. This includes setting time limits to disqualify users who don’t share their secret keys (because they know someone got their position), or those who try to submit their strikes after the secret keys have already been revealed.

There are some better methodologies, such as zero-knowledge proofs (“ZKPs”), as well as its adaptations in the form of zk-SNARKs and zk-STARKs. Here, a user would be able to prove their position, without actually revealing it. However, no implementations of ZKPs exist on the Solana blockchain yet.

Finally, we could use a data oracle in the smart contract, but this would require a trusted third-party and alleviate from our goal of being a strictly on-chain game. Plus, trusted solutions like Chainlink only offer price feeds for the Solana network at this time.

In conclusion, we decided on a strategy-based game that operates on the Solana blockchain, using a commit/reveal scheme.

**Solana**

There are several caveats to building an on-chain game on Solana.

Unlike Ethereum, you cannot store the state or any data in the contract. You have to make a separate “account” for each state, and you must know the size of the account when creating it. You cannot change the size of the account after creating it.

The account is charged a rent exemption fee to keep data available, or it is removed forever by Solana. The rent exemption fee is calculated based on the
aforementioned account size.

You get a maximum of 32 kilobytes of heap for the contract execution. This is basically the amount of computation power the contract has.

There is also no read functionality for the contract. You must parse the raw data.

It is easy to see why no one has built an on-chain video game on Solana yet.
Part 3: The Game

It all starts with a fascinating storyline:

The year is 2100.

Mars is colonized. Solana is the defacto currency. People have found neighboring aliens. And, we all surprisingly get along.

Until one day, a mysterious disease wipes fertility in all the universes. Early research shows that a distant planet may hold the ingredients to restart it. But, there's only enough juice there for one civilization. The other, will go extinct.

It will take scientists on both sides time to build robots capable of getting to the planet fast enough. But, neither side wants to wait. They cannot afford to wait. Their survival depends on it. As much as they like each other, the war must begin now. They must destroy each other for their own survival.

Hence begins the ultimate battle of the cosmos. The best warriors on each side have already volunteered. They must be found and destroyed at all costs.

So warriors, we ask you:

ARE YOU READY?
**Gameplay**
1. Players are either an alien or people depending on their NFT
2. Player initiating the game defines how long each round should last ("time limit")
3. Player is then presented with a public key to the game, which can be shared with others to join
4. Players select where they want to place themselves on their own planet in a 10x10 grid, i.e. [8, 4]
5. Players select a secret key ("blinding factor") that is only known to them. A unique blinding factor should be used by each player every time they pick a position for their avatar
6. Players submit where they want to be in the grid by sending a transaction to the game’s smart contract
7. Players then have to submit the blinding factor, along with where they want to “strike” the opposing team on the grid
8. Players whose positions were hit are removed from the game. Those who did not submit the blinding factor or submitted invalid ones are eliminated
9. Survivors repeat steps 4-8, except until everyone on the opposing team is eliminated
Part 4: The NFTs

An important aspect of our game is great art. The NFTs are a set of 8,888 randomly generated characters. 4,444 of which are aliens and 4,444 are people. To ensure a fair mint, Aliens VS People utilizes Metaplex Candy Machine for the mint.

The NFT images themselves are stored on the blockchain with Arweave. This ensures that the NFT minted by players are timeless. Imagine, in the year 2,200 inheriting a super rare Alien VS People NFT minted by your great grandparent or relative and using it to play on a virtual reality fork of the game.

Each NFT contains metadata on the type of NFT minted including the team, the attributes etc. This metadata is used within the game to validate the users and their teams. The NFTs contain unique attributes and each attribute has its own rarity weight attached to it.

While using the Metaplex Candy Machine for the mint, the mint order is stored on the Solana Blockchain. As the blockchain is public, botters can see which NFT would be minted next and calculate rarity etc. Their bots can snipe a rare NFT and mint it straight from the contract within milliseconds. This is an officially known exploit on Metaplex and botters have been using this on various mints, no wonder it seems unproportionally difficult to mint rare NFTs.

To ensure our rare NFTs are going in the hands of our community and not bots, we are introducing a reveal scheme during the mint. When the NFT is initially minted, there will be a placeholder gif. The NFT’s metadata and image will be updated to reflect the actual NFT minted later.

In addition to playing with their NFTs, users can view their NFTs within the dashboard along with the NFT’s features. Additionally, users can join the Aliens VS People global chat with the NFT they minted.
Part 5: Architecture

Overview
The game logic is handled by a collection of 2 primary zero-copy program derived accounts (PDA): the game state and the round state. The game state contains vital information about the game such as the current round, players, and if the game is finished along with the game winner. The game PDA references the top level data of the game. The round state PDAs contain more specific data about the current round.

Handling any variable data within Solana becomes challenging as we must know the exact size of our account when we initiate it. These two PDAs are independent of each other and only reference each other’s public key to derive the entire game state. This allows for us to be able to make a variable amount of rounds for each game, as the game may extend indefinitely if there is no winner.

Independently, these accounts would serve little utility however, using these two PDAs together we can derive powerful remote procedure calls (RPCs) & program instructions to handle the gameplay logic in a decentralized fashion.

Example of instructions we can call by combining these accounts include:

1. Current eliminations
2. Timeouts
3. Game winner or draw

The PDA Accounts

Game State
The game state PDA is a top level account that handles the higher level logic of the game. Including adding/eliminating players, creating rounds and managing game winners.

When a user creates a game, the frontend passes the round duration in seconds. This allows users to tailor the game speed to their liking.
The game always initializes with support for up to 10 aliens and 10 people NFTs. When joining the game, the createPlayer instruction must include the Metaplex Metadata account, game address and the authority. When a player makes a request to join the game, the game smart contract (Solana program) verifies that the user is joining with a valid Alien VS People NFT before adding the user to the game. The program is able to derive the team of the NFT, i.e. if a user joins with an alien NFT the program will add the user to the alien team. If a user joins with an invalid NFT the program will throw an error to prevent unauthorized users from joining.

The game state includes a round_head field which is incremented every time a new round is created. If the round_head > 0, the program will throw an error for new users joining the game as the game has already begun.

When a new round is created the program updates the current_round_ends_at field in the game state with the block unix timestamp + the round duration specified by the host while creating the game. If a user tries to create a new round while the current time is before current_round_ends_at it will throw an error for “round time remaining”. However, to handle situations where all players have already submitted their positions in the round there is a can_create_new_round field in the game state that takes precedence above the round time. The can_create_new_round field is updated when the check_game_state instruction is called.

The check_game_state instruction call takes the current round address and the game address. The call loops through the submitted positions in the round state and eliminates players along with updating the game state with the eliminations and if there are winners.

If a player has been eliminated in a previous round and attempts to make a move in a future round the program will throw an “unauthorized move” error.

From the frontend, the check_game_state transaction is chained with other instructions. This is to make the gameplay more seamless instead of having players submit another transaction to update the game state. Regardless of the user who calls the check_game_state from the frontend, the game state updates for all the users with the latest state.
The above architecture makes the frontend more seamless. From the frontend, to query if a user has made it to the end of the game and has won, we simply need to query whether the player.isEliminated is false within the game state. We can combine various data from the program to derive various game states directly from the blockchain.

**Round State**

The round state contains the players’ positions and attack positions along with some additional metadata.

Users are only able to submit their attack positions if `round_state.attacks_start_at_unix > current block unix time stamp`. The `attacks_start_at_unix` timestamp is a future timestamp derived by adding half the time of the round duration to the time the round was created. As half the time is designated to submitting positions and the other is for attacks. If a user tries to submit their position when attacks have already started, the program will throw an error for the user and consequently timeout the user. This also prevents users from updating their positions during attacks or waiting for the attack phase to set their positions.

Similar to the game state’s `can_create_new_round` boolean, if all users have submitted their positions, the round state includes a `can_attack` field that is set to true. This speeds up the rounds for the players instead of having users wait for the timeout, if everyone has already submitted their positions, users can continue to the attack phase.

When a user submits their initial position, the program instruction is called with an u8 32 byte array hash. The position is hashed using a blinding factor. The blinding factor is stored in the local storage of the user’s browser. The blinding factor is automatically generated/refreshed in the background every round to prevent any malicious user from reverse engineering another player’s positions. The hashed position is stored for the player in the round state under the `position_hash` variable for the player.

For example if a user submits their latitude for their position as 5 and longitude 1, the position [5, 1] is hashed with the blinding factor. The hashed position is sent with the submit position program instruction. This encrypts the user’s position, preventing users from reverse engineering competitor moves.
When a user submits their attack position, the `submit_attack` accepts the unencrypted attack position. As the round is divided into two independent attack and set position phases, the attack position does not need to be hashed since other players are unable to mutate their existing positions during this phase.

Additionally, when the blinding factor is sent, the program will attempt to dehash the `player.position_hash` with the blinding factor provided. If it is able to decrypt the hash into a valid position, it will accept the attack and update the dehashed position. Otherwise, it will throw an error for invalid blinding factor.

After the attacks and blinding factor are submitted, the program is able to derive the eliminations and update the game state with the eliminations. The program also updates the eliminations with `eliminated_by` with a reference to Metaplex Metadata account for the successful attacker. The frontend is then able to show who eliminated you and who you eliminated from the blockchain.

**Frontend**

Although the game logic is very technical and powerful, the gameplay itself should be seamless for all types of players. As evidenced by the architecture of the smart contract, the contract is designed to support reactive frontends to provide a seamless experience for users.

The Aliens VS People official frontend subscribes to an event emitter for both the round state and game state. This allows the user interface to update in real time based on the game’s state. The user interface shows if you are eliminated, who joined the game, if you can attack, all in real time.

All the technical calls are handled behind the hood.

The program instructions are also designed to be modular and chain transactions, instead of having users submit multiple transactions.
For example to create a new game and add the host, you can chain these transactions together as followed:

```javascript
const txn = new anchor.web3.Transaction();
const [gameAddress, gameNonce] =
  await anchor.web3.PublicKey.findProgramAddress(
    [provider.wallet.publicKey.toBuffer(), gameCrc32Hash],
    program.programId
  );

const createGame = program.instruction.createGame({
  gameNonce,
  gameCrc32Hash,
  new anchor.BN(roundTimeSeconds),
  accounts: {
    host: provider.wallet.publicKey,
    game: gameAddress,
  },
});

const createPlayer = program.instruction.createPlayer({
  accounts: {
    authority: provider.wallet.publicKey,
    metaplexMetadataAccount: metaplexMetaAddress,
    game: gameAddress,
  },
});
txn.add(createGame);
txn.add(createPlayer);
await provider.send(txn);
```

The above transaction creates the game and adds the host to the game in one transaction. To the end-user, it’s just one click of a button.
The goal of AvsP is to provide a decentralized game mechanism that makes it simple for developers to build on top. Developers can leverage our existing technology to build their own apps to supplement gameplay without having to worry about having a trusted source for the game state. Additionally, the RPC calls for AvsP make developing on it very simple.

Developers can easily tap into AvsP user base and additionally charge for using their apps, providing them a stream of revenue to develop on AvsP.

In the following examples we will demonstrate how powerful AvsP can be as a basis for your application.

**Solana Programs**

If you are developing a Solana program on top of AvsP, you can query the game state account and round accounts directly from your Solana program.

For example, consider an escrow-styled program on top of the game for the benefit of the winner. This program would be fairly easy to develop on top of the AvsP program.

We can query the game state and round state directly from our Solana program by calling `to_account_info().data` on the respective account within the escrow Solana program.

From the game state, we can derive if the user joining the escrow is actually within the game. Furthermore, the Alien VS People Solana game state program account also derives additional player metadata.

Consequently, from one simple call, we already have a verifiable source of data to derive the team of the user, the player’s Metaplex Metadata account, whether the player is eliminated, who the player is eliminated by and more. This makes developing highly sophisticated programs simple.

If we want to check who should win the prize from the escrow, we simply need to check if the game has finished from the game state and check the users who have not been eliminated in the game state.
Web Applications
The speed and transaction costs of Solana is unparalleled. Consequently, Solana-based smart contracts make a great building block for web applications.

In the following illustration, we will discuss the foundation to build a web app to notify followers of a player on social media about a game victory.

To start, we can subscribe to updates on the game state by creating an event emitter. We can listen to the “change” event on the event emitter to get updates in real time when the game state changes. From the game state change, we can check if the game is finished and not a draw, depending on whether the player is in the winning team and not eliminated. If these conditions are met, we can go ahead and call our API to tweet about our amazing victory.

Already, from a single event emitter listening to the state changes in the AvsP program, we are able to deduce in real time from the blockchain if the player has won and consequently do some further action based on the result.

```javascript
const updateSocialMedia = async (program, gamePubKey, userPubKey) => {
  const emitter = await program.account.gameState.subscribe(
    gamePubKey,
    "processed"
  );
  emitter.on("change", (state) => {
    const { aliens, people, aliensWin, isDraw, finished } = state;
    const allPlayers = [...aliens, ...people];
    if (finished && isDraw) return;
    const player = allPlayers.find((player) => player.authority === userPubKey);
    if (player.isEliminated) return;
    if ((player.isAlien && aliensWin) || (!player.isAlien && !aliensWin)) {
      // call API to post user win on twitter
    }
  });
};
```

In the above coded out example, we can see how we can make a highly sophisticated app reading from the AvsP Solana Program with a couple lines of code.
Part 7: Legal Considerations

We want our game to be accessible by everyone.

To ensure this, there are several considerations that went into the gameplay.

**Strategy**
Our game should be based on strategy and not just pure luck. Plus, what fun is pure luck anyway?

To incorporate strategy, there would also be a chat room, accessible to only those on the same side. This way, teams can collaborate on where to strategically position their avatars. Imagine all team members colluding to place their avatars on the exact same position on the grid, so the opposing team has a tough time finding it. Or, a naughty team member secretly placing their own avatar elsewhere, so that when the entire team gets struck – they are the only ones who survive. The possibilities are endless.

**Rewards**
Initially, we contemplated modifying NFTs to decorate them with their wins. But, this isn’t ideal, since first and foremost – our NFTs are art collectibles. Some people simply want their NFTs as such without any intention of participating in the game. They would justifiably be upset if others’ NFTs are decorated with medals, while theirs are not, simply due to their disinterest in participating in the game.

This also alleviates another concern, which is, that the reward should not be something of value. If we gave away something of value, validators may avoid publishing blocks that don’t serve their interests. It may also make our project into a security, since the won NFTs could be treated as dividends of an initial investment (being the purchase of NFTs). Regardless, if users were only playing our game for rewards, we didn’t do something right. Since the idea of our game was always to be a timeless classic that users play for the game’s enjoyment in and of itself.
Timeline
No empty future promises.

Our game would be ready and playable as soon as you have your NFTs to participate. This allows you to attribute a value based on our delivery at the time of mint, and, holds us accountable to deliver everything before you spend a penny.
Part 8: Changes

Several changes were made from our previous conception of Aliens VS People. This was due to newfound technical limitations, feedback during beta testing, as well as general improvements.

Technical Limitations
With a computation limit of only 32 kilobytes on Solana, and no state storage, the entirety of the prior game play data needs to be parsed. As such, the computation power simply doesn’t exist on Solana to support thousands of simultaneous players.

As well, the host of the game would have to pay outrageous fees to support a large number of simultaneous players. In addition to the data associated with the game and positions, 32 bytes for the pub key * the number of players would need to be stored somewhere.

Simply put, it would be cost prohibitive even if Solana had greater computation limits.

Feedback
Based on feedback received in our internal testing, players didn’t want prolonged gameplays that required waiting on a large number of users. Instead, they wanted shorter games that they could host at whim to challenge their friends or family.

A game that would take days to conclude, simply allowing for thousands of people to submit their positions, didn’t seem appealing to our beta testers.

Improvements
Given the technical considerations, as well as the feedback received from beta testers, we decided to implement several new things since our last whitepaper:

1. Players would now be able to initiate rounds themselves
2. Players would be able to share a link to invite others to participate in the game
3. Players would be able to define how long each round lasts, so that they can challenge their family/friends to a “quick” game
4. Developers would be provided documentation on how to build for the AvSP ecosystem

Part 9: Conclusion

At the end, we have an entirely on-chain game that meets all of our requirements. Anyone can interact with the gameplay directly from the blockchain. They can even build their own user interfaces ("UIs") and features. The game is entirely on-chain, decentralized, immutable and can be played by all NFT holders.