Poking Holes + Iterating

Coming up with *ideas* is fun, but the real work of design is finding ways to make those features harmonize with the technology. On blockchain this is especially hard because the technology is very constraining, and that makes it even more important to make *self correcting systems* that don't require frequent maintenance or tuning changes. When refining ideas it can be helpful to *"poke holes"* in the design so we can refine or reject weak ideas as early as possible. Below are some thoughts and guidelines for blockchain design:

- **The Game only updates when a player sends a transaction**
  - There is no way to "update the game in the background." We cheat this constraint by using predictable formulas, where all variables are known ahead of time except for the duration.
    - ✅ *"A Shadowcorn earns 100 UNIM every hour while staked."* This is possible to calculate with one variable (`duration`).
    - 🚫 *"A Shadowcorn earns 10 UNIM every hour for every minion staked in the Dark Forest."* This is impossible to calculate because the number of minions staked can change over time.
- **Randomness is slow and expensive**
  - Since the blockchain is fundamentally predictable, we have to rely on a third party to inject randomness for us (Chainlink VRF). This process takes two transactions, which we refer to as the "commit/reveal" pattern. The player commits to the action before they know the result. Then Chainlink VRF injects randomness and the result of the action can be revealed.
- **Keep data small**
  - Transactions have a very limited processing budget, and part of that includes reading or transferring data. It is very important to keep features compact.
    - ✅ *"What is the name of Shadowcorn #2700?"* This is a small data lookup.
    - 🚫 *"What are the names of every Shadowcorn that Staked during the past week?"* This requires us to filter a lot of Shadowcorns, and then it requires a (potentially) large number of names to be returned.
  - We work around this limit by pre-calculating as much as possible. For example, if we wanted to know the average `arcana` stat for all Shadowcorns, it would require us to check every Shadowcorn, sum their stats, then divide to get the average. This touches 3000 pieces of data in one operation.

  If we planned ahead we could capture that data in a rolling average as the Shadowcorns hatched, spreading the work out so every player computed 1/3000th of the work. We would do that by saving `rolling_average` and `shadowcorns_counted` values. When a new Shadowcorn gets minted, we add `arcana / shadowcorns_counted` to the `rolling_average` and increment the counter for the next person.
  - Whenever possible, we want to put the "processing" burden on the player, rather than on our code. Chess is a great example because the rules are very compact; there is no hidden information, and nothing happens outside of a player's actions. Yet the game is still deeply strategic. All of the processing in Chess happens in the players' heads.
- **Players WILL try to exploit us**
  - Anywhere that "value" changes is a possible point of attack. Either the Design or the Tech (or both) must protect against players who would extract or destabilize value from our economy. Some questions that every design should be able to answer:
    - If the player can "get something for free," could they repeat that behavior by jumping to a new wallet over and over?
    - Can multiple players work together to create an imbalance in the economy? Could one player do this themselves with multiple wallets?
    - Does the feature work if a player has 1000 Unicorns in one wallet?
    - Does the feature work if a player has one Unicorn in 1000 wallets?
    - With enough time, money, and perfect understanding of the system - how would someone exploit this feature?
- **Mind the Golden Path**
  - We start by imagining new features in the best case scenario. Players are having fun, there is an exciting back-and-forth happening, and everyone is excited to see what happens next. Features are fun when they work, but we have to account for the unhappy path as well.

    - Is the feature still fun if one side always wins?
    - Would I use this feature if my actions didn't affect the outcome?
    - Am I being punished for the actions of others? Can I succeed by playing smart, even if my team is dumb?
    - Could the first player into the feature scoop up all the rewards before anyone else?
    - Could a whale end the game early if they un/staked all at once?
- Try to design Self-Correcting Systems:
    - Tug-of-war is a bad game because the stronger team will always win; there is no incentive for a new player to join the weaker team. We could make Tug-of-war a better game by adding a new rule: *The winners must give one player to the losers after each match*. This would create a trend where the two sides would eventually even out (self correcting).
    - Rock-Paper-Scissors is a better game because each team has balanced (but asymmetrical) strengths and weaknesses. If everyone is throwing Rock, we can expect clever players to switch to Paper. After a while, Paper becomes the new meta, and the clever players will jump to Scissors. When one faction gets out of balance, that faction becomes weaker and the others have an advantage until they regain equilibrium.
- Remember that it can be fun to break the game *a little bit*. This is the concept behind our Rituals in the Minion Hatchery: a player could get a recipe that results in generating more output than input - free money! - but this is limited by the `charges` system. A player can only break the economy a few times before their rituals break down.