Datorien Anderson

I would recommend looking up or asking about any unknown terminology or concepts that need further explanation for this to be an effective post-mortem report. To begin, POV: Pixelshift / Data Division / etc., was meant to be an information-heisting game where the player can cooperatively use their AI and immerse themselves in this fictional world. In hindsight, a quick elevator pitch has always been Beyond-Two Souls meets Lethal Company and Deus Ex. Why use these games as a descriptor for the elevator pitch?

In Beyond: Two Souls, a playstation exclusive game, the player can switch between their main avatar and a supernatural entity called 'Aiden'. In Lethal Company, every '#' of days the player must collect '#' profit for the company and in Deus Ex: there are varying ways of completing these and some level of reactivity.

There was no estimated budget to begin nor a clear project end date. The product's purpose was to create a game using the POV world's setting about AI, Drones and what the scare-factor and potential harm that current and emerging technologies possesses and poses to our current society. The intended customers were at first students due to initially having AI / ML modules – but shifted completely to games, and then near the end drone-enthusiasts by project owners' decisions.

The initial scheduling of this project was:

- Game Design Document
- Proof of Concept & Pitch
- Prototype... (Iterative)

As the project was being developed the heart of the game came out to be an experience where the player learns about how their data is taken, what can do with it and the possible future(s) that can arrive. An immersive learning experience about AI/ML and Data Science but it's not overtly beating the player over the head about it.

True outcome is that the game is stuck in prototyping and there was an overall concept being forced throughout the weeks despite it not working – the source code base is pretty large and the POV project as a whole due to it being relevant gives it a relatively large media size. But the compiled build of the project is only (see last seeing it) several hundred megabytes. There was a series of assets as well as refactoring of my own systems to develop POV: Pixelshift.

When it came to designing this–my thought processes and what it should be has stayed the same. I was consulted to make a game adaption of a short film but also for my knowledge regarding AI / ML.

With that in mind, the first thing I did after watching the short film a few times was brainstorm what can be gamified based on what is seen. Though a lot of other parts regarding POV were thrown my way during the process; there were research papers read about ludic objects. Which is playing as a primary part of learning in simplistic terms. Ludic pedagogy is the reason why gamification is a huge thing now–like with Duolingo and what GRX Immersive Labs has done with the Arts, Beats and Tech App.

What went right?

- I was able to successfully make the initial make of:
    - Finalized GDD to use to create proof of concept.
    - Creating the proof of concept – Collecting data as a core mechanic.
    - Refactorization of the Independent Systems:
        - Time Trial System.
        - Initial shooting / targeting practice system
        - Elimination Mode
    - AI / ML Rough Draft Learning Modules
    - The initial drone-human POV switching.
    - Making the initial utility tools and data items.
    - Creating the pixel shifted hidden object.
    - Downloading the data, holding data on a flashdrive and collecting said item to be sold or donated.
    - Level Management System.
    - Base Integration of the Calendar System
    - NeoFPs → Cowins FPS Engine 3Cs Overhaul

The codebase had been maintained well and the independent systems were relatively easy to ad hoc and then refactor into the project. There was an experimental idea with the data collection that was done and tested initially which was a success (albeit truly expanding it wasn't a given goal).

I had been able to at first get the project into a unified whole that made sense for the company it'd be developed under (GRX Immersive Labs) as well as a concise adaption by having Pixelshift be a prominent group in the game albeit – much of the actively story stuff would have been told later.

The prototype was going to be a part of the game that lasts 'three' months. Which was to give enough time to see what could make it more fun – the game doesn't actually last three months but in game time. That could easily be like 3 - 4 hours of playtime.

And what went wrong?

A decent amount but those are things that I did scope against far early on; I forewarned that a dedicated engineer was absolutely needed from prior experience both personally via hobbyist projects and professionally.

- At the very least, for the levels, I did also share ("The Level Design Book," Gitbook)to help show what ALL creating a level entails.
- Constantly changing 'deadlines' which weren't deadlines by wanting a holistically complete item to use for an event to show.
- In the case of the project, I noted that more meetings resulted in more confusion and changes on what was to be done.
- There was seemingly no tolerance for risk – nor anything to base the project scope on.
- In the end, I didn't have an effective way to communicate ideas (such as storyboarding) flowcharts only make so much sense – albeit, my drawing tablet stopped working.

The initial biggest issue is a risk factor that I identified back in January, being that we needed a more senior gameplay engineer and also a level designer. I already suggested someone I worked with and had participated in game jams with but they were glossed over despite being on three indie projects from start to finish with experience working with publishers and working on milestones-- as well as being able to take on multiple roles. That had been someone I trusted to get the job done and done well who has indie game industry connections that could help smooth a lot along.

The second biggest issue that occupied time when building this prototype was that I also wasn't given a clear timeline to scope the project against which is also a part of the budget.

I would have *severely* downsized the initial parts because I assumed I was just building an adaption to POV and was being consulted for that, and not doing too much development work unless needed. However, that wasn't the case.

This had led to me creating three major revisions to the initial design (before it was switched over to Unity) and researching topics and technology that ultimately didn't matter. Such as *cryptocurrency* -- to integrate that technology would have required consulting a blockchain engineer and we would have already needed a foundation to start generating content to test. As well as knowing how much money we would have wanted to back us up when it comes to minting cryptocurrency as well as accounting for gas fees.

The fastest summation in regards to this is that cryptocurrencies have differing ways of 'proof' which goes into how it is valued. There is proof of stake and proof of work. One of these are more GPU-intensive than the other and is valued more such as the first version of ethereum as well as bitcoin. It's not energy sustaining but that's while it's priced up. These cryptocurrencies have the same issues with GenAI in that its very GPU-intensive–powerful, graphic cards are used for calculations and more which is why Nvidia, AMD and Intel are highly valued.

Play to earn is more than possible without cryptocurrency but would have required starting from scratch again because that needs to be involved with the game's architecture and such from the beginning.

It's a big concept and in most games such as Call Of Duty, 'COD Points', PUBG's 'G-Coin' and more recently Helldiver's 2s' 'Super Credits' are all game resources that have a real-world trade value– buying 800 Super Credits for '$10'. In Helldiver's 2, this can be found in-game in amounts of 50 to 100 (since I last played) and, also, warbonds while it can't directly be purchased with money, they can be found in-game to buy items that the player uses super credits to use for the pass. So, this isn't something that can be added in for the sake of it unless it's just supposed to act as a 'time-saver' for in-game or in-app purchases (such as in Assassin's Creed games.

So, I was able to significantly mitigate the two biggest issues by pushing for the prototype to be developed in Unity. Without that it would still have been purely an abstract idea and no testing against the concept.

The third issue is going more along with the second being that there wasn't a clear scope. I wasn't able to identify a fun factor for the game because the timeline of what the game needed to be and when it needed to be available kept pushing with nearly each industry event as well as with each month, or biweekly meeting. This led me to needing to triage what works and what doesn't work and shift priorities with what I should clearly be working on. Because of this I failed at identifying what can be fun about this, I wasn't able to experiment with what can possibly make it fun because at the end of the day with game development. Gameplay is king. If the gameplay is fun and solid, graphics are not going to matter.

Note that: in game design and development, a good amount of things are not user facing unless the player is actively interacting with it in some way.

A Smorgasbord of Mid-Progress Changes

I did initially try to mitigate feature creep–which I explained quite early on and its dangers–as well as reduce that but whenever I did mitigate it – it was requested yet again. Inferring that it was a needed component to the project. But I also completely disregarded wanted additions later such as cryptocurrency which wasn't thought of again from the initial documents until GDC. If cryptocurrency was to be planned for this–it would have been in the finalized GDC before I started working on the proof of concept and then the prototype.

Involving cryptocurrency would have increased the overall risk level and required capital to start. I will highly advise against that, despite it being funded in GDC and talked about, many–many gamers are adverse to cryptocurrency and the only ones adopting it are people who are okay with it. For example, your average gamer doesn't know about, care about nor were they affected by *Sam Bankman-Fried*–all they care about is how fun your game is and if it appeals to them.

If the game was purely action–even drone racing wouldn't be the way to go as racing and obstacles is ultimately a racing genre of game–it would be something like **battle bots** or another type of **action beat-em-up**. Pivoting to a drone racing would NOT have been easier at all – there would have been a bunch of AI parts that would have needed to be created such as a Race Director. Racing game is not a simple pivot. There would have been a need to have a Race Director; each AI would need its own profiles. The framework for this would still need to be made as well as the game's race track.

Ultimately, the increased requests for action and 'loops'–added to the overall complexity of the project making it harder overall to manage and build even for a prototype. These changes ultimately also changed prediction estimates which are also based on a sole developer. (Note: Aside from Stardew Valley, another really huge game that was released by a sole person, Manor Lords, took the guy 7 years to create even with contracting people.)

A gameplay *loop* comes in micros and macros and are inherently more of design and abstraction. (which I also learned that loops are frequently talked about on an investor and executive level); it also seems to be more of a free to play, KPI or player retention thing too. The Helldivers 2 primer of a loop I made works better than one for example, Stardew Valley, because Helldivers 2 is a live service game.

Meaning that it lives and dies by content and engagement. For example, they had a grand idea of releasing 1 warbond (like a battlepass) every month. As a result, the cosmetics and weapons are honestly just reskins and aren't special, not to mention the game balancing is more towards a PvP (player vs player) rather than a co-op PvE. (co-operative player vs. environment). And some players actively wanted a slow down and more utility out of the monthly passes because they've stopped feeling worth it. (Fun fact: Helmets having more effects on gameplay were cut to release the game. Source)

A big case study can be made with Helldiver's 2 for a long in learning, such as while they are partnered with Sony for publishing that actively hurt their player count and reputation due to requiring making a Sony account to play (due to game server issues, they stopped requiring that and then in May said it was going to be needed) but the issue is–Playstation Accounts are only available in several regions per the terms of service, which meant the game was made purchasable and playable to people outside of these regions and they'd have to break the law to play or get the game removed and refunded from their account.

What got cut as well as risk mitigation.

- Using a tablet to navigate menus normally as well as with the Terminal.

What Went Wrong: There were a lot of things dependent on this being done, and due to a time factor I wasn't really able to test functionality. What went wrong with making this - as I've made a similar system before that worked and was functional - is more due to time pressure. I had already identified what it needed to be functional but time sensitivity

made implementing this unnecessary even if it would be extremely valuable to the identity of the game. I'd spent too much time building this and that's time I didn't really have to measure against anything
.
A few systems were dependent on this to work.

- Switching between different bot types & drone customization.

I added the code in as a placeholder, so, it exists as there was going to be a battlebot styled thing where the player is fighting another bot. For obvious reasons, this was quickly cut and the ends clipped so, I spent much less time working on this (as well as the drone customization). Thus I was able to save time elsewhere in the project.

- 3Cs (Camera, Character, Controller)

What I didn't account for as a risk point is how complex switching between the drone and the player would actually be. The logic is relatively simple: if I am controlling the drone, I should be able to view the drone's camera and not move the player's avatar and vice versa. Where this went wrong is evaluating what controller to use - as well as if I should make a controller. The risk mitigation for this would have been for someone else to make these systems–which I did by purchasing assets–but they only helped to a point and I still had to integrate these with the project.

I realized I had to also make a Camera State Management system, keep track of gameObjects (actors) in a serialized way (otherwise they lose their reference) the project kinda went haywire with that and I had to ad hoc multiple solutions and brainstorm others. Nearing the finalized prototype (our last one) - I still have issues with the camera management because... quite frankly there are a lot of cameras in the scene.

If I remove the drone - it's a vital part of what is thought to be it– but to me the breakaway from POV was oppression, a surveillance state and information being at risk. But, a drone racer and such -- it's kinda boring unless someone is purely into that. I would have devised the game way differently if I were to pivot to a drone only experience; such as using raycasts and having the player 'scan' appropriate things.

The insistence of having a drone playable and which means it will also be constantly preset -- also negated the overall interactivity that can have been done, implemented and tested. I can see now why most games that have drones it's either a 'companion' that

follows using traditional AI behavior trees, a time-limited thing to use OR the game has a specific execution.

A hobbyist or drone enthusiast, will play something like these:

Uncrashed: FPV Drone
https://store.steampowered.com/app/1682970/Uncrashed__FPV_Drone_Simulator/

Liftoff: FPV Drone Racing
https://store.steampowered.com/app/410340/Liftoff_FPV_Drone_Racing/

Or, one of the most interesting -- well, executed games about drones is Duskers. In which, we play a drone operator on a spaceship and it's a horror game and we control the drone via text commands.

And ultimately, this stuff is what I meant by what makes POV special? Ultimately, a learning company is creating this game which is what I wanted to lean heavily into. Each sprint the game's identity that was being told to me changed. Which near the end, pivoting more towards something that explicitly negates any other identity that was planned. This adds to an earlier point that any thing I had planned and was actively working on had to be scuttled due to the project owner's changing visions of what the game is marketed and envisioned to be.

The insistence on drones hurt the project as well as pivoting away from that despite my suggestions. I did note a risk due to this that could be mitigated with someone who has knowledge on how to program 'vehicles' in gameplay – I also vetted people on LinkedIn based on their domain knowledge with what we needed. Also regarding the project pivot away from drones – Discord was ultimately meant to be a tablet-based multiplayer game. It did not start as a chat app at all – that was a game feature. Now, it's a chat and messaging platform that hosts game activities and is quite underutilized.

Now, the main task of this feature is that you can PLAY it but at the same time it's a Buddy AI–throughout development this seemed like a requirement despite. A Buddy AI is simply

an NPC AI that behaves differently than all other NPCs. This single decision is something that has a propensity to have a great scope for complexity. In the proof of concept, I simply added: a follow AI component, AI component Nav Mesh Agent and another piece so that it could track and toggle OFF all those functions when the player wants to switch to controlling the drone. The good thing is that viewing the drone did considerably less logic.

I keep using the word scope against; in game design, for this feature let's consider what this (with sources, Dale Green):

- How much room for error the AI has (if the AI can accomplish everything without the player it's 'too smart'; if it can't do anything without the player, it's 'too dumb'.
- How often are they 'in-focus' (meaning they will be with the player throughout the journey) so they are in-focus much more frequently. As such there is an increased magnitude for simple, repetitive and/or flawed behaviors to be noticed by the player and as such it will feel less polished and the AI will be spotted.
- The focus also accounts for differences in perspective behaviors any AI that shoots needs to handle reloading and what happens when a magazine (or clip) is empty, with a buddy AI – often they are cheating with unlimited ammo but the logic for it still needs to be scripted as if it's not as well as account for should the buddy reload OR should the buddy just have a cooldown until they can shoot again.
-
    - If the buddy agent has unlimited ammo and they do a good deal of damage, this could be seen as an exploit.
    - But also the focus needs to account for decision-making logic in the behavior tree while shooting and whatever else is 'queued' in the logic.
- Positioning! Non-buddy agents spawn for a given task (to be traded with, quest given, killed and more). Both non-buddy and buddy AI agents have a risk of increased decision complexity involving positioning and this can become taxing on the CPU if not architecture correctly causes performance issues.
- Determining what needs a 'buddy agent' has to be brought to life and then, choosing a brain. This is the level of autonomy that the AI agent has. Again, there is a CPU especially, performance budget and this can greatly add to complexity.
- Positioning is also very important in relation to what and where the drone would be during any action sequences. If the drone is in action sequences, they should also have health and correlated logic regarding that.
- The 'best' approach to high autonomy decision making would be GOAP (goal-oriented action planning / not that, this is a technology before ChatGPT /

> GenAI's multi modals) the behaviors are procedural instead of a Finite State Machine or Behavior Trees. However it is very much processor intensive (and likely the reason Rockstar uses C# based on job-postings I've seen throughout the years) but the game's design needs to permit that approach – POV does not.

In this project, what the project owner views for POV would be high in CPU performance and would need to handle a 'pausing' in the desired autonomy when the player takes control of this; meaning that I also need to handle taking control – which fun fact, most drones behave as a 'helicopter' so to say but to have this feel interesting and worthwhile, I had to ensure it behaving mostly as a spaceship in space a la Elite Dangerous.

So, while the feature of controlling the drone, moving the drone and then just switching back to the player actually works, it's impossible for me to healthily focus on content and also level creation as well as also debug, tweak, test, and make the flight controls feel nicer. While there are 'three' of us–again–hours are critical and unless there is an effective team, there is a limit to what can be healthily done in a two week sprint much less a one week sprint. Much of what works internally in game development is also not user-facing until the later stages of how a system works. User-facing as in viewable or interactable.

Not enough time, spread too thin!

This was the BIGGEST issue by far. I had to pivot around multiple things and I wish I could continuously test and iterate over what I have working as I was doing initially but the significant numerous time pressures throughout made it to where testing was not a choice. To be able to test I need to have a build with things to test. The GDC & SXSW, timelines and around that time had caused the biggest shifts especially from me doing iterative development to attempting to make one big project to use.

I'd spent so much time laying down the foundations and such, that I didn't have enough time to build content. I've spent well over 300 hours working on this project simply because if I didn't put in way more hours than what is planned and tracked - I never would have gotten this far. Racing other drones wasn't added or thought of to put into the prototype because there would have been a need to make the AI for that, which meant making the behavior logic trees for the drones racing which isn't simple to implement. It's why I mentioned drone time trial obstacles instead of drone racing.

Project Management wise - we can work in sprints - but ultimately game development is a marathon where sprints help meet goals and should be very malleable. Which was achieved in the last two.

It's also why time estimation suffered. The option with the most relative ease to shift to and not in the actual final two weeks would have been to switch back to the core mechanic with at least four weeks to recenter and continuing focusing on the switch–even if the project owner was more enthusiastic with something purely with t

Concerns with advice given:

My given and informed advice often, was somewhat taken into account, or never quite mattered despite the fact that I was contracted on as a consultant which would have been fine if I weren't the sole person developing. And while a bullet-point, it was something I noticed many times throughout this project. It will be important to remember for any and all possible feature game design & development projects that Game Design is ultimately software development that's interactive and (often) entertainment and that interactive software is high-risk. It's prone to be bug ridden by nature.

Game features can't just be taken on or pivoted easily to another, especially if there is only one person actively developing the project. Do not look to large A - AAAA studios for game design decisions, gameplay and ways to do things because they usually have over 300 people working on them usually and rather than many hats, the contributors are often specializing in one explicit thing. When there is prototyping to pitch a game in these larger studios, it's done internally and still with a small team.

Majority of game made and developed are indies where people (going in for the first time) will usually have a couple of years worth of saving up as a solo-dev, they do rev-sharing with people they've done game-jams with or in terms of funding, they are a specialist who've lift a AAA studio and were funded based on that alone. And if not any of those, they are already an established game developer who has a fanbase and are already working for an indie publisher.

- Such as David Szymanski whose publisher is New Blood Interactive.
- As well as Xalavier Nelson Jr. who's the lead game designer and dev for, Strange Scaffold, the biggest release last year was El Paso, Elsewhere.

What to do next?

Microsoft's Post Mortem Research paper, check out the results.
If the project is being pursued further:

I do not suggest using GenAI to create usable art to show investors / publishers for funding.

I would highly suggest getting someone who does Game Concept Art and have them create mockups of the builds as well as what each faction should feel like. For example, does the 'independent' faction give feelings of 'self-expression' / 'hacker'? Does Pixelshift give off a demure of rebellion yet corporate? Does the government feel oppressive?

If you still added the tablet in– would the player unlock 'stickers' to put on that? Unlock a different color cable for interfacing? These are all things that can be shown with a game concept artist.

You don't need to have someone for the entire duration of a project–you can, for example, hire a level designer to build the blockout for the three company levels and won't need their services unless a huge addition is needed.

Hire a 3D modeler to clean 3D models for game development and make a tablet game prop.

Hire an engineer to make the drone and human switching systems and give documentation on how it works.

(Note! If you decide to switch back to Unreal later – there is a HUGE difference between blueprints vs. C++, especially on a thinking level. Blueprints can also end up being hard to read.)