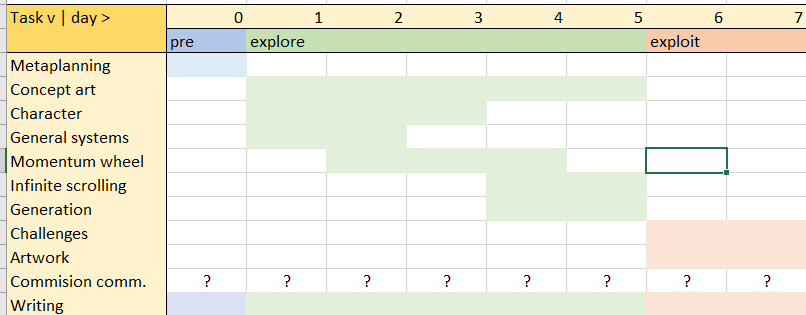
**Day 0 – pre production**

**Heartbeast’s Production Point**

To make things a little easier on myself I decided to base my structure and production plan on Heartbeast’s Explore & Exploit theory. In short, it tries to solve the problem of game development getting stuck in prototyping, production, or worse: ***both***.

Read his article [here](https://heartgamedev.substack.com/p/production-point) and keep a lookout for his book – [Production Point](https://heartgamedev.substack.com/p/im-writing-a-book) – whenever that hits the market.

As I already have a decent idea of what kind of gameplay I want to *try* making for the game, it wasn’t hard to split the tasks up into prototyping & production tasks. From there I made an estimate on how long it would take to make the individual components and put them in a basic gantt-like planner I made in Excel.



Planning the exploration phase goes against Heartbeast’s theory. Optimally, exploration should be as flexible as possible (scope by doing). Therefore, I have given it as much time as I think I could get away with. Some of the problems I could run into along the way are:

* Concepts I explore don’t work at all and need revising, resulting in adding or removing made systems
* Making the systems in the exploration phase takes too long delaying the exploitation phase
* Tasks I’ve put into the categories don’t fit and end up costing time because they should have been focused on earlier or later in development (possibly hampering other tasks along the way)

I’m going to be as flexible as I can with the time I’ve given myself. If things don’t go as planned in the exploration phase and I never make it to the exploitation phase, then so be it. Hopefully this will still stand as a worthwhile experiment that I’ll be able to bring into future endeavors.

**Time management**

You might have noticed that I simply put numbers on the days in the gantt planner instead of dates or weekdays. This is because I’m setting aside a total of 10 days for the completion of this project – 7 of which I’ll dedicate to the work. I’ll probably need a few days off due to unforeseen circumstances or simply because I want to relax. The blogposts will all come out on days that I’ve decided to work. Hopefully this all works out.

**My tools**

The following is a selection of tools I’ve chosen to use in the production of this project.

[Godot 3.5 (release candidate 8)](https://downloads.tuxfamily.org/godotengine/3.5/rc8/)

Godot 3.5 is very close to stable at this point and has some features I’d like to try out (such as the SceneTreeTween, and scene unique nodes). It is of course possible that there will be a few regressions even in this late release candidate but, I’m not too worried about it because of the short time schedule of this project, and I’m confident in my ability to find workarounds if necessary.

[Hack n’ plan](https://hacknplan.com/)

Hack n’ plan is my Trello-like sprint planner of choice. It has a bunch of features that are gonna aid in the time management and progress tracking of this project (task categories, time estimates, and priorities to name a few). I’ve used it before, and I will use it again.

[Github](https://github.com/) & [Github desktop](https://desktop.github.com/)

Classic solution for version control. In addition to also being a free backup it acts as an additional discipline check for me; I know I’m not focused if I’m committing multiple unrelated changes. I’ve tried git bash but didn’t find it friendly enough.

Other software:

* [Paint tool sai](https://www.systemax.jp/en/sai/) for making concept art and sketches
* [Aseprite](https://www.aseprite.org/) for making tilesets and other pixel art assets
* [Audacity](https://www.audacityteam.org/) for making simple sound effects

**Conclusion:**

With that I feel sorta prepared to start working on the player, systems and maybe some concept art tomorrow. There is more that I could’ve covered here like:

* Godot plugins
* Godot project management
* Plans for music and sound
* Plans for animations and character pixel art

I’ll leave those for future posts if I feel like writing about them.

Cya on day 1.

<https://mega.nz/folder/9qY2GKBL#BvoQMohs3OS_5IH0IjhMRg>

**Day 1 – Game Plan**

**What am I even making?**

Last time I talked mainly about my approach to the process. But what is it that I’m even working towards currently? I have an idea for the world that this game is going to take place in but I’ll keep to the gameplay mechanics for now.

The main idea for the gameplay is a mishmash of different gimmicks that have stuck with me over the years, including but not limited to:

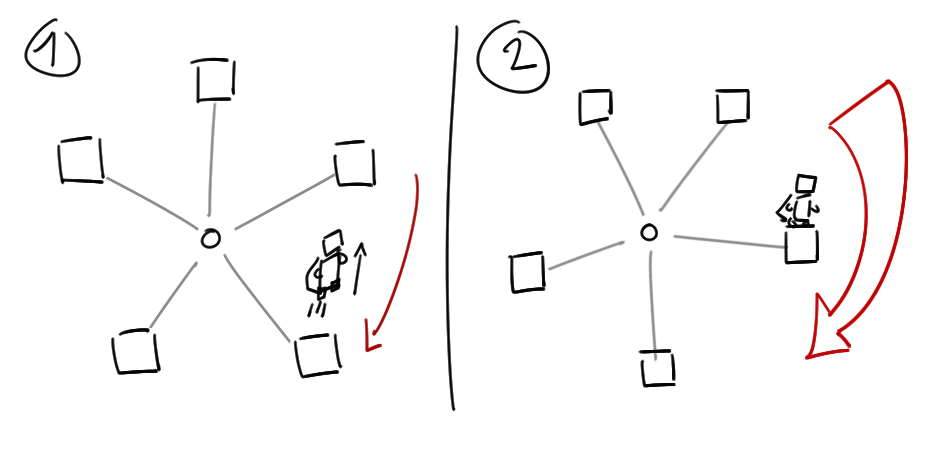
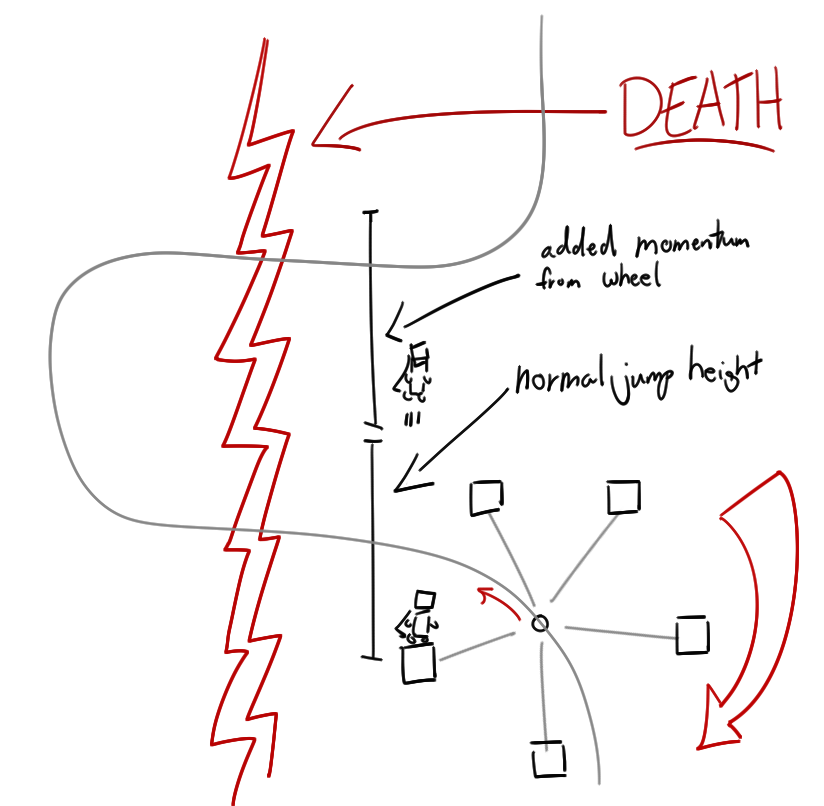
* Platform wheel from Yoshi’s Island that moves along a path using velocity that the player applies to it + fangame style platforms
* Celeste dashing (but only downwards)
* Infinite climber type generation (but with an altitude goal)

Diagram

Description automatically generated



Why did these gimmicks combine in my head? I don’t really know, but they did, and now I’m excited about trying it out. I’ve got a few theories for how this could work out.

1. The player can jump through and snap to the wheel’s platforms. This can be used to generate speed fast by standing on the edges of the wheel and jumping up through the platforms.  
   
2. The player can jump off a platform and use its’ momentum to stay in the air for longer. This can be used when the platform enters a treacherous section but has enough speed to make it through on its own.  
   
3. The wheel gimmick combines well with other gimmicks such as:
   1. Gravity flipping (all four directions)
   2. Vi’s [butterfly gimmick](https://delicious-fruit.com/ratings/game_details.php?id=22538)
   3. Lava moving from side to side engulfing the wheel in short intervals
   4. Etc. etc. etc.

With all these ideas on top of the fairly complicated mountain-scaling-device that is the wheel itself it’s beginning to sound like a hard-to-learn AND hard-to-master type of deal. If that turns out to be the case then it’ll still hold up to fangame difficulty curve standards (maybe lol). But if I can make this feel really good and introduce some of these ideas in a smooth way, then maybe it can stand on its own to some degree. The near future will tell.

**How about today’s progress?**

I started today by setting up the hack n’ plan’s programming tasks since those will be the ones I’m going to tackle first. I set up which tasks would have high priority, estimated how long each would take, and started programming.

Graphical user interface

Description automatically generated with medium confidence

And today I managed to hook off 4 things on the list, which was… underwhelming.

The problem was the platform snapping behavior.

I wanted the player to snap only when the player’s feet where under the platform in the previous frame, and over it in the current. Sounds simple enough, especially since there’s not too many different combinations of nodes you can do to solve this problem.

* KinematicBody2D (RigidBody2D and StaticBody2D seemed unfit in this case)
* Area2D
* RayCast2D

At first I thought the platforms had to be of type Area2D because KinematicBody2D nodes seemed not wanna overlap with each other, ever. But there is no signal in KinematicBody2D that lets you know when it is overlapping an Area2D, and connecting to all platforms’ body overlap signal on ready seemed counterintuitive.

For a short period of time thereafter I wondered if I should make the player use an Area2D instead. That would mean that I would have to handle all collisions manually, but at least I’d have my precious overlap signals right in my player class! It sounded possible but it would take more time than I was willing to sacrifice then. Besides, there should be a better solution to this, right?

I got an idea for a hybrid solution of sorts. The player’s main body would be a KinematicBody2D, and it would have a Area2D or RayCast2D to detect platforms with. When I detect platforms using my new platform detector I can start using the KinematicBody2D’s move functions in a platform-logic-y way.

That’s when I got stuck for real. I couldn’t find a way to do this in a way that satisfied me. No matter how I thought about it, the thought of there being a better way was still irking me. Then I remembered how the [Godot 2D Platformer](https://godotengine.org/asset-library/asset/120) example project had moving platforms that the player could jump through. How the hell did they do it?

Turns out there was a crucial detail I had overlooked. CollisionShape2D (yeah, not KinematicBody2D) has a one-way-collision boolean. Finally, the puzzle pieces all stuck together: When on top of the platform the player is automatically moved along with the platform (because it is a moving KinematicBody2D as well). When moving through the platform, some distance checks done by two RayCast2D’s at the feet of the player will trigger a snap. ON GOD. And that would’ve been the end of the story if I hadn’t decided to also implement a state machine for the player to clean up all this code.

See you tomorrow.