Technical Design Document

### Lead Statement

I want to be a tech lead that listens, takes the other programmers’ perspectives and opinions into account, and a tech lead that doesn’t assume that he necessarily knows best in every situation.

I want the programmers to have the freedom and trust to implement tasks and solve problems the way they best see fit, so that they can work as efficiently as possible, instead of bugging them down with too much overhead.

I want to be a tech lead that focuses on functionality first, and form second, though I do also care about the quality and cleanliness of the code, so I want to find the balance where the overhead to making it cleaner doesnøt become too overwhelming.

### Technology Pillars

* Use enums instead of strings and booleans (negotiable) (Be careful of using enums too much, currently we transformed an enum to a class with an array instead, so we could contain functionally related to the enum in the same class.)
* No public field variables (non-negotiable)
* Low coupling/high cohesion (negotiable)
* Make code as reusable as possible (negotiable)
* Don’t reinvent the wheel, if someone has done it before, reuse what they have made. Use Assets and Packages if they offer the needed functionality (non-negotiable)

Why no public field variables? What benefit does this rule bring?

Why always enums over booleans? Are there not cases that can be better expressed through a boolean?

In our project as a start we keep most variables public, so designers easily themselves can go in and modify them to test out new values. Later on when balance is “done” they can be made private.

To Be fair he did mention below that they are using serializable variables so they can tweak them without having them public.

### Key Technical Innovation

There are not really any new technical innovations. From a technical side, the game is focused more on robustly integrating audio, visuals, and input into an enjoyable experience, and using some powerful 3rd party subsystems to do so (URP, FMOD, Unity).

### The Basics

* Target Platform: Console, though the builds will be made for PC, but the game is designed to be played with gamepads for modern consoles.
* Engine: Unity
* Programming Language: C#
* 2D sidescroller
* Shape: Metroidvania, though not as large as a typical metroidvania, the progression through the gameworld is in a metroidvania fashion
* Genre: Puzzle Platformer with a focus on music
* Duration: 10-20 minutes, though we have not completely settled on the length, it is generally agreed upon that it should be possible to
* Controller and player count: Xbox + PlayStation controllers. 1 player.

### Inspirations

* Finite State Machines, as a way to manage the control-flow of gameobjects
* KISS (Keep It Simple Stupid), don’t overcomplicate or overengineer a solution.

### Tech & Tools

* Game engine: Unity
* Addons, Libraries, Assets: URP, Input System, 2D Tilemap Extras, FMOD Integration, (Easy Save 2, maybe)
* Design tools: Unity’s builtin TileMap Editor
* FMOD (audio), Photoshop (sprites)
* Whichever IDE the programmers prefer
* Project management tools: Discord, google drive
* Communication tools: Discord
* Version control tools: Github
* Bug tracking and reporting: trello

### Code Quality

We don’t focus too much on this, as some of the programmers are against heavy practices for this project, as they see it as a large overhead, and would rather just get things working, instead of making sure that it is maintainable, as they said “No one else is gonna be looking at this”.   
When that is said we use the debugger and playtest a lot during development to make sure that our code acts as we want it to. Furthermore the QA-lead has also been doing some testing on certain features to find further bugs.

I have tried to uphold some coding standards (like, use Enums instead of strings, Enums for managing gameobject states, and no public fields, but use Serialized fields for the inspector and properties for interfacing with other classes) but there seems to be a bit of a Laissez Faire approach.

(For this short project, this might not become a problem but if you are going to work in bigger projects, try to use something simple like SOLID, otherwise you are going to have a codebase, where everything breaks when you make changes)

### Tools for Content Production

Custom inspectors, TileMap editor, and decoupled interaction patterns (interaction between player and objects) and object logic, to make the designers able to build more complex puzzles with these building blocks. The FMOD project (audio stuff) is also in our google drive, such that it is decoupled from github, so the sound designer can work on it, and the rest of the team doesn’t need to worry too much about it.

The interaction patterns and object logic is not fully done, as we have not had a clear list of what puzzles the designers want to have implemented, though that should happen during this week.

The designers have not yet used the TileMap editor to create any levels, but we programmers have used it to create some prototype levels.

### Production Methods

We use a Kanban board to keep track of tasks, and distribute them at weekly/biweekly meetings where we talk about how we are following the production plan, and what the next steps need to be. Usually the tech tasks are distributed based on which person has more experience with that particular subsystem of the codebase, or who would be more interested in working on that task, if people have no preferences I just assign them one.

### Team Communication

We primarily communicate through discord, and hold weekly meetings, if someone needs to collaborate on stuff throughout the week they usually arrange that themselves. Sometimes tech or design have their own meetings to discuss stuff, but most often it is the whole team.

The communication in that way happens rather organically, and programmers and designers primarily talk with each other instead of all of it going through the tech lead.

I have also made a couple of different documents outlining some of my thoughts regarding some of the different major decisions (like which engine to use, how to structure git, and why we switched to URP).

### Challenges

The biggest challenge is to ensure the quality of the code, both in regards to when it is written and when it gets checked afterwards, and managing the workflows associated with this quality assurance.

Further is to clearly communicate with the design team, and making sure the tech team has the specifications for how stuff in the game should work, so that they can implement the right thing.  
It is also a bit hard to get a feeling for how well the designers can use the version control system to make changes themselves, it feels a bit like they are a bit scared of github, even if we use github desktop and we have had a session where we ran them through how to use it.

### Subsystems

