Technical Design Document

(soft) and (hard), are used to indicate how negotiable something is, (soft) is a bit more Laissez Faire and something the tech lead is not super set upon, and (hard) is pretty non-negotiable (though if you ask nicely, it can still be up for discussion).

### 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

* Tech Style Guide
  + Use enums instead of strings (soft)
  + No public fields, use *[SerializeField] private* for inspector stuff, *private* for internal logic stuff, and *Properties* for interfacing with other scripts (hard)
  + Use enums instead of booleans to track the overall state of gameobjects (hard) (booleans can still be used for checking binary stuff, like whether or not the character is grounded)
* Git version control
  + main-branch is untouched, it is used as a backup in case we need to reset something else, and for the final version of the project (hard)
  + Development branch for merging features, content, bug-fixes, and adjustments. Versions are named: D#2 Name (hard)
  + Release branch for pushing development-builds when reaching milestones. Release versions are named: R#2,D#2 Name (hard)
  + Separate branches for features, content, bugs, and adjustments. These are named with one of the following prefixes: f\_, c\_, b\_, a\_ (hard)
  + Usually the tech lead handles merging into development and release (soft)
* Flexible code
  + Low coupling/high cohesion (soft)
  + Make code reusable if possible (soft)
  + The interaction Architecture (Strategy pattern): Splitting behaviour of interactable objects and their method/pattern of activation, such that they can be recombined as basic building blocks, and possibly more complex interactions can be composited from these (hard)
* 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. We shouldn’t make things from scratch if it is easier to find it made somewhere else (hard)

### 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

* Games:
  + Wandersong
  + Hollow knight
  + Loom
* Technical 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, there is elegance in simplicity.

### Tech & Tools

* Game engine:
  + Unity
* Addons, Libraries, Assets:
  + URP, Input System, 2D Tilemap Extras, FMOD Integration
* Design tools
  + Unity’s builtin TileMap Editor
* Audio, 2D and 3D graphics tools
  + FMOD (audio), Photoshop (sprites)
* IDE(s) (Rider, Visual Studio, VS Code, ...)
  + Whichever IDE the programmers prefer
* Project management tools (i.e. Trello, Notion, ...)
  + Trello
* Communication tools (i.e. Teams, Discord, Slack)
  + Discord
* Version control tools (i.e. GitHub, self-hosted Git, Mercurial, ...)
  + GitHub
* Bug tracking and reporting (i.e. Jira, ...)
  + Google docs
  + Trello
* Asset and file sharing
  + Google drive

### Code Quality

There are a few coding guidelines in place to ensure more robust code (restriction of access-modifiers, and use of enums), though these are not always followed by the programmers, and thus the tech lead will once in a while do some refactoring or clean-up if needed.

Though form is important for code quality, function is much more important, and the coding guidelines are therefore just in place as guidelines, and for the most part not rules set in stone, that at all times must be followed.

First get it working, then make it pretty, if possible.

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 and share these with the rest of the team in a bug-list document where the programmers can see descriptions of what happens, and when it happens.

### Tools for Content Production

Custom inspectors, TileMap editor, and the interaction architecture (decoupled patterns of interaction between the player and objects, and the behaviour of the objects), to make the designers able to build more complex puzzles with these building blocks.

The FMOD project is 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 about it too much about it, and so they don’t need to rebuild it every time they pull from git.

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

The TileMap editor has not yet been used to create any final levels, but the 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 the tech lead just assigns 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 with a flat hierarchy, and programmers and designers primarily talk with each other instead of all of it going through the tech lead.

Tech lead has also made a couple of different documents outlining some of his 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 and keeping bugs to a minimum.

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 things.

Having 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. Also whether or not the currently planned tools for them are enough for them to iterate on content themselves.

### Subsystems

