Game Design Document - *The Cure*

CONTENT DRAFT

Xavier

# User Interface

The user interface will be minimal during exploration and non-combat interaction except for dialogue captions and status updates; it will be reactive, with contextual elements being summoned or dismissed as needed to keep the screen uncluttered and easy to read. During interactions, character portraits will be present to identify speakers and indicate which characters are affected by status changes. The user interface should not overwhelm the player or distract them from gameplay, instead providing whatever information is pertinent at the time and remaining out of the way otherwise.

A familiar control scheme is intended - character navigation will involve standard four-direction key controls such as cursor keys or WASD on mouse-and-keyboard systems and use the primary analogue stick on gamepads, while interaction can be performed through adjacent keys (such as Q, E, R, F, Z, X, C, Shift, Ctrl, Tab, numbers 1 through 4) on mouse-and-keyboard systems and the face buttons (**ABXY** or ✕◻△○) on gamepads. Camera/view controls will use the mouse or numeric pad on mouse-and-keyboard systems or the secondary analogue stick on gamepads. The general controls will be presented to the player in the game menu and the tutorial level, while contextual controls such as interacting with objects will be displayed when relevant. The object of this is to provide an intuitive method of moving around in the world so that the player does not have to learn new controls and can focus on the game.

The menu screens and in-game GUI are laid out as follows:

[wireframes go here]

# Art and Sound

The game makes use of three visual styles: pop-in character busts for dialogue and interaction sections, using caricature depictions of the characters; semi-realistic stylised character models for general gameplay, striking a balance between realism and economy; and simpler, less realistic background art, allowing the characters and foreground to take visual precedence while providing an artistic ambience to scenes. Our visual design is inspired by the art style of *The Legend of Zelda: The Wind Waker*, *Ōkami*, and *Aragami*, using cel-shading similar to *Wind Waker* and painted-style black outlines inspired by *Ōkami* and *Aragami*, lowering the production costs for graphical assets compared to more realistic art styles.

Multiple art styles have been considered, and to expand on this it is important to look at how a balance of graphical economy and visual appeal is achieved in other games of this genre. Possibilities that interest us include 2D-in-3D, in which characters and items are represented as flat sprites moving around in a 3D environment; 3D-in-2D, in which characters and items are represented as 3D models imposed upon a painted or pre-rendered 2D background; and full 3D, which can be achieved economically and with pleasing effect by using stylised low-fidelity 3D models and 3D environment with an aesthetic reminiscent of 1990s console games.

In the case of The Cure, the approach taken is full 3D as this allows for a dynamic and traversable environment without fixed or restricted camera views and characters that can potentially move in arbitrary rather than predefined directions, restrictions of the controller notwithstanding. In addition to this consideration, it is important to define the aesthetics and visual atmosphere such that the artistic design of the game agrees with the narrative direction. In terms of textures and colour scheme, we have elected to use a bright and clean colour palette to achieve a sufficiently friendly look; the lighting and hue-saturation will vary according to location and current gameplay status, with brighter and warmer lighting in more familiar places and when doing well, and dimmer and colder lighting in more remote places and when doing poorly. This will be linked with the Goodwill mechanic, and the intention is to provide a sense of the player’s progression through the game without giving them an explicit performance gauge.

A decision made early in the conceptual stage of design, intended to set this game apart from others in its genre, was to produce a fully diegetic soundtrack for the game, with the bustle of townsfolk going about their business offering an acoustic backdrop in the cities and towns of the land, the ambient sounds of nature in wilder areas giving the player a sense of tranquillity or isolation depending on situation, and the music of birdsong and travelling performers adding atmosphere and reality throughout the experience. A limited amount of incidental music may be included at key moments to supplement the natural soundtrack, but we are avoiding the full symphonic approach that is often taken with fantasy RPGs to create a more grounded and personal experience.

# Tools

For games with a focus on dialogue, story, and exploration, there are many options available for the game engine. As this is a small production it is best to use one of several well-known and widely-supported engines and editor suites, such as GameMaker, RPGMaker, Unity, Unreal Engine, or Godot.

Focusing on the dialogue and visual novel styling, RPGMaker provides an easy way to create retro-styled 2D JRPG-format games including menu-based interaction, combat, and inventory management. This would be useful for prototyping the narrative flow and some simple mechanics in our game, but it would not be a suitable platform for the immersive 3D experience we hope to create.

GameMaker is more flexible but is similarly intended mostly for 2D production and uses an easy-to-learn workflow ideal for small independent game production. While a number of notable games in both 2D and 3D have come from this engine, it has a proprietary scripting language and the 3D aspects can be difficult to work with as it lacks a 3D scene viewer.

Godot is an open-source engine with a similarly intuitive workflow to GameMaker and as such is great for teams without a strong programming side. This is a decent engine to work with as it is very capable of handling 3D game development and works with a range of programming languages, but may struggle when producing large-scale open-world games.

Unity is a highly versatile engine for 2D and 3D work and is among the most common choices in the indie scene. It requires some programming knowledge and various external tools to make effective use of its workflow, but it is a powerful engine suitable for game developers of any size and experience; additionally, the team is already familiar with this engine’s application for 3D game development.

Unreal Engine is optimised for 3D game production and tailored toward larger and graphically intense game production, finding use in a fair number of mainstream and independent games over the last 21 years. It offers a more comprehensive suite of development tools than Unity and boasts outstanding production quality on multiple platforms; a drawback is that the engine may be too powerful or intensive for a small team to work with, and given the limited timeframe this makes it a less appealing option.

For our game, we have decided Unity is the most appropriate option for a game engine. Our reasoning is that the team is already familiar with the editor and workflow and its capabilities are in line with our aims without having extraneous features or technical restrictions.

Other production tools for asset creation include:

* 3D modelling and animation software such as Blender. Blender is available for free and is supported by a large community and dedicated development team, with a vast array of addons and tutorials available. Other 3D modelling software such as 3DS Max, Maya, and Cinema 4D may offer more professional quality and specialised toolsets but cost as much as £1500 per year, making them unsuitable for a small independent development team.
* Substance Designer and Substance Painter, which while expensive can accelerate the production of high-quality materials and textures. These are packaged with a cost of around £200 to £300 for either annual subscription with updates or perpetual licence without; it is possible to work without this suite, but the benefits are worthwhile when realistic materials and high-quality shaders are desired.
* Image editors such as Photoshop, GIMP, Krita, or PaintTool SAI, which can be used for concept art, environmental art and texturing, model texturing, and interfaces. GIMP and Krita are free while PaintTool SAI is priced at 5500 JPY (about £37) and Photoshop is available as a subscription costing £240 per year.
* Vector editors such as Inkscape or Illustrator, which can be used to create clean scalable designs for interfaces and icons. Inkscape is available for free, while Illustrator is available as a subscription costing £240 per year
* A Digital Audio Workstation such as FL Studio, which can be used to produce and sequence music tracks without requiring an actual soundtrack orchestra and recording studio. Depending on the feature set, a DAW is likely to cost between £200 and £1000; additional instruments may be freely available or might cost as much as £500 for a mid-range orchestral set.
* Audio editors such as Audacity, which can be used to process music tracks, sound effects, and voice clips using various filters. Audacity is available for free.

As the team member responsible for art and sound design, Xavier already has many of these asset creation tools available; as such, the initial setup cost for these will not require budgeting. The game engine and any other tools being used by other team members will need to be budgeted for, however.

# Asset Lists

This section contains a set of tables which identify and define the assets required for the development of this game in as much detail as is appropriate at this stage, including textures and graphics, sound effects and voice lines, text strings for dialogue and interface, 3d models and animations, scripts and logic assets, and miscellaneous items.

[tables of assets go here]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CATEGORY | Asset | Description | Application | Priority | Development Cost |
| Audio Assets | Example01 | Dummy sound file | Testing only | High | < 1 hour / Free |
|  | Example02 | Dummy sound file | Testing only | Medium | < 1 hour / Free |
|  | Example03 | Dummy sound file | Testing only | Low | < 1 hour / Free |
|  | Example04 | Dummy sound file | Testing only | Low | < 1 hour / Free |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CATEGORY | Asset | Description | Application | Priority | Development Cost |
| Image Assets | Example01 | Dummy image file | Testing only | High | < 1 hour / Free |
|  | Example02 | UV chequered grid | Testing only | Medium | < 1 hour / Free |
|  | Example03 | Colour test pattern | Testing only | Low | < 1 hour / Free |
|  | Example04 | Dummy image file | Testing only | Medium | < 1 hour / Free |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

[etc]

# Technical Documentation

## Features and Systems

Key technical features for the game include a traditional hit-point system which is controlled by "hurt-boxes" (a collision region that is marked as being able to do a specific amount of damage) intersecting or colliding with player hitbox regions (which is a collision region marked as being able to receive collision from other hitboxes and damage from hurt-boxes.) This method of hit registration is fairly common amongst games with melee combat and will serve the game appropriately.

In order to monitor the player's reputation amongst in-game factions, a global script will be operating in the background that keeps scores based on player actions in-game - each major story event having a numerical value associated with it, or setting reputation flags that automatically set player allegiance or opposition of factions (notably attacking the Fera marks the player as an enemy of the islander faction outright, regardless of their score to that point).

Sound controllers will be attached to two places - trigger boxes in level areas so that region-appropriate sounds are played according to factors like time of day and player actions, and to enemy encounter logic so that various audio stings can be used to alert the player to events that they may not immediately see.

An encounter controller is necessary to govern the rate and difficulty of enemy encounters with the player and will likely entail a region-specific weighted "spawn list" that contains a table of possible enemy encounters and the likelihood of their appearance off-screen or when entering an area. To simplify progression, combatants will be only strong enough that the player can defeat them on their first time through that area, "new game +" play-throughs scaling these values to prevent the experience becoming too easy.

Speechcraft and questing will be tied to the same overall system of branching dialogue selections and flag-setting so that the technical aspects are simplified. Providing enough of the "correct" responses for a conversation will open up Speechcraft possibilities, story progression and player-facing information (rumours, chatter, information on the world, etc.) while a similar system monitors quest progression. Completing events in order throughout a quest or side quest marks an objective complete, while unlocking and/or displaying the next objective in a quest sequence. After all steps in a sequence are complete, the quest or side-quest is marked as fully complete and any tasks linked to that final completion flag will also become available.

## Infrastructure

Assets and other files are sorted into the following category folders: World, Objects, Entities, Interface, Audio, and System.

* World contains the terrain and scenery data for each level as well as level-specific controllers and triggers.
* Objects contains items and objects that don't move, such as scenery props and collectibles.
* Entities contains mobile entities such as characters, creatures, projectiles, and vehicles.
* Interface contains graphics and behaviours for GUI/HUD including menu screens and inventory.
* Audio contains all the audio, including sound effects and dialogue.
* System contains global scripts and logic as well as miscellaneous features.

Asset filenames are presented in the format “TYPE\_OWNER(\_SUBTYPE)\_DATA(\_SUBTYPE)”, meaning that the first word is the overall asset type (a logic system, an entity, a voice line), the middle word or two words describe what the asset does or what uses it (a navigation algorithm, a specific character, a set of greetings), and the last word or two words identify the type of data it is (a script, a 3d model, a sound file).

For example: “mob\_sheep\_white\_tex\_diffuse” refers to the diffuse texture of the white subtype of a sheep in the "mob" (mobile entity/creature) type. Type tags may be more specific than the main categories to which they belong; for instance, the Entities folder contains "CHAR", "MOB", and "ENT" types, referring to humanoid characters, creatures, and non-living mobile entities respectively.

Scripts belonging to entities go inside the subdirectory of that entity, as do the model and texture files and any other data.

## Major Technical Questions & Risks

During the development of any project it is necessary to assess potential risks and take precautions against them to avoid damages and delays. For each risk, I will assess the likelihood and severity of the risk to produce a score using a risk assessment matrix and suggest precautions for each step of the problem resolution process as outlined below.

For the purposes of this design, the risk assessment matrix is a table representing the likelihood of a problem occurring and the severity of that problem if it occurs, ranked according to a five-point scale on each metric. The numerical product of these two scores is calculated to determine an overall risk factor, with the highest risk factor indicating the problem that demands the most attention and extensive safeguards and failsafes.

For this project, we have selected the problem areas to focus on based on their potential to cause damage to the whole project. The persistent risk of data loss is tackled in a broad variety of ways, but our other major risks may be less obvious at first. These include tool failure, sunk cost, tech deficit, and scope creep.

### Overall Project Setback Risk assessment:

**Risk assessment**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Catastrophic | 5 | 10 | 15 | 20 | 25 |
| Severe | 4 | 8 | 12 | 16 | 20 |
| Significant | 3 | 6 | 9 | 12 | 15 |
| Moderate | 2 | 4 | 6 | 8 | 10 |
| Trivial | 1 | 2 | 3 | 4 | 5 |
| *Risk factor:*  ***Moderate-High*** | Very Unlikely | Unlikely | Likely | Very Likely | Near Certain |

As this design document facilitates our pre-production ideals and production goals, it is very difficult to know with certainty how likely or unlikely a particular risk is on a case-by-case basis. The reality of risk assessment is that where insufficient information exists to make a judgement, preparation for the worst case is the second best option and so it should be considered likely that the project will meet setbacks throughout development, but those setbacks will not be as severe as they could be were the risks not considered at all.

Our four-phase approach to resolving risks in our design applies to this assessment also, but with the nuance that it must incorporate new findings as the project evolves.

***Awareness:*** *being informed of a potential problem so that it is possible to respond to it.*

Periodic review of progress in the project is of utmost importance. Finding that a part of the project is delayed should always be seen as a result of potential problems, and not a source of future problems. Early identification allows enough time to be dedicated to a setback or delay that any technical deficit or adjustment in scope around the problem simply does not occur. Allowing the problem to go unaddressed is not appropriate in any circumstance.

***Prevention:*** *setting out precautions to reduce the likelihood or delay the onset of a problem.*

After setting aside the necessary resources to attend to any issues that would arise during development, all team members should take steps to identify the cause as according to the above categories in a manner that does not assign blame but enforces responsibility. The results of preventative actions will invariably lead to new findings or the discovery of difficulties in the project, allowing for refinement of the design document and design goals to occur alongside production holistically.

***Mitigation:*** *utilising damage control methods to reduce the severity of a problem once it occurs.*

Risk assessments of individual project elements prior to this one form the first part of mitigating actions - identification of possible sources of risk. Outside of this, preventative actions before any delays are identified and periodic meetings help maintain the quality of the project, reducing the scale of sources of risk if they did become apparent during production of the game over the course of that year.

***Recovery:*** *returning to a functional state after a problem has occurred and recouping losses where possible.*

Despite the probability of risk and the effects each risk factor may have on the project, each of them can be addressed easily and effectively - data loss, in particular, is high-risk and high-impact but resolved by automatic backup. Provided that the team bears any possible problems with the project in mind at all times and the risks are reassessed semi-periodically, production can be streamlined around the measures that must be taken to prevent and recover from any risks.

### Data Loss

**Risk assessment**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Catastrophic | 5 | 10 | 15 | 20 | 25 |
| Severe | 4 | 8 | 12 | 16 | 20 |
| Significant | 3 | 6 | 9 | 12 | 15 |
| Moderate | 2 | 4 | 6 | 8 | 10 |
| Trivial | 1 | 2 | 3 | 4 | 5 |
| *Risk factor:*  ***High*** | Very Unlikely | Unlikely | Likely | Very Likely | Near Certain |

A data loss event is likely to occur and would have a severe impact on the project depending on what is lost. If not resolved, even a minor data loss could have damaging effects on the end result: for instance, a single missing texture that is not replaced before release would disrupt gameplay and potentially damage the reputation of the developers.

**Awareness**: team members should be aware that data loss may occur due to accident, user error, or technical fault, and should be prepared to negotiate this.

**Prevention**: to avoid a data loss event, team members should make sure to save their work frequently during development and prepare backup copies and old versions of any work produced.

**Mitigation**: by storing backups in a separate location it is unlikely that all copies of a file or project will be lost due to an event.

**Recovery**: load the backup or revert to an old copy to restore work; if this is not possible, recreate the work.

### Tool Failure

**Risk assessment**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Catastrophic | 5 | 10 | 15 | 20 | 25 |
| Severe | 4 | 8 | 12 | 16 | 20 |
| Significant | 3 | 6 | 9 | 12 | 15 |
| Moderate | 2 | 4 | 6 | 8 | 10 |
| Trivial | 1 | 2 | 3 | 4 | 5 |
| *Risk factor:*  ***High*** | Very Unlikely | Unlikely | Likely | Very Likely | Near Certain |

A tool failure is unlikely to occur, but it may have catastrophic impact on the project if it does happen, depending on which tool fails and whether a replacement is available. An example of the significant case would be for an image or model editor to malfunction, causing delays but with recoverable files; an example of the catastrophic case would be for the game engine or world editor to cease functioning, causing a potentially irreversible loss of the game project.

**Awareness**: team members should be aware that production tools may malfunction or cease functioning entirely and should be prepared to make changes to the production workflow.

**Prevention**: to avoid a tool failure, team members should ensure that all production software is on the latest stable version and that there are no version differences between devices being used.

**Mitigation**: in many cases a tool failure can be resolved by reinstalling the tool and/or making configuration changes; tool failures that cannot be resolved may be mitigated by ensuring that backup project files are stored in data formats readable by other tools in case a change is needed, and team members should be competent in multiple tools such that a new tool can be used to continue work.

**Recovery**: if possible, use a new tool to resume work or attempt to reproduce lost work.

### Sunk Cost

**Risk assessment**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Catastrophic | 5 | 10 | 15 | 20 | 25 |
| Severe | 4 | 8 | 12 | 16 | 20 |
| Significant | 3 | 6 | 9 | 12 | 15 |
| Moderate | 2 | 4 | 6 | 8 | 10 |
| Trivial | 1 | 2 | 3 | 4 | 5 |
| *Risk factor:*  ***High*** | Very Unlikely | Unlikely | Likely | Very Likely | Near Certain |

A sunk cost scenario is very likely to occur and may have significant impact on the project depending on the nature of the cut content and the timing in relation to the project schedule. An example of the trivial case would be for an optional costume material to be scrapped; an example of the significant case would be for the flow or balance of a level to be compromised, calling for heavy restructuring of that level.

**Awareness**: team members should be cautious of over-investing in content that may be cut or over-optimising a stage that has not been fully defined and be aware that quality assurance and scope changes may render some production obsolete or unnecessary.

**Prevention**: to prevent sunk cost, production priorities should be clearly laid out before development work begins on each section and formative evaluation should be undertaken during production to identify which aspects are most solid and which are likely to need revision.

**Mitigation**: to limit the knock-on effects of revisions and cut content, it is advisable to “sandbox” tentative production work and use placeholder assets for prototyping.

**Recovery**: recouping losses from sunk cost is generally not possible but cut content may be kept on record for future reference in case it is needed or as bonus trivia for players.

### Tech Deficit

**Risk assessment**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Catastrophic | 5 | 10 | 15 | 20 | 25 |
| Severe | 4 | 8 | 12 | 16 | 20 |
| Significant | 3 | 6 | 9 | 12 | 15 |
| Moderate | 2 | 4 | 6 | 8 | 10 |
| Trivial | 1 | 2 | 3 | 4 | 5 |
| *Risk factor:*  ***High*** | Very Unlikely | Unlikely | Likely | Very Likely | Near Certain |

Tech deficit is very likely to occur and may have significant impact on the project if it is poorly managed. An example of the moderate case would be background actor scripting not being fully optimised; an example of the severe case would be a core system being incomplete at time of release.

**Awareness**: team members should be familiar with the causes of tech deficit and the consequences of letting it go unchecked.

**Prevention**: to prevent a tech deficit developing, regular progress reviews should be made between milestones to ensure that all production is occurring on schedule.

**Mitigation**: to limit the impact of a tech deficit, additional development time should be made available to catch up on any work that is behind schedule when a milestone is reached.

**Recovery**: if sufficient time and resources are made available for catching up and any production that depends on delayed systems or assets is not rushed, tech deficit can easily be resolved.

### Scope Creep

**Risk assessment**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Catastrophic | 5 | 10 | 15 | 20 | 25 |
| Severe | 4 | 8 | 12 | 16 | 20 |
| Significant | 3 | 6 | 9 | 12 | 15 |
| Moderate | 2 | 4 | 6 | 8 | 10 |
| Trivial | 1 | 2 | 3 | 4 | 5 |
| *Risk factor:*  ***High*** | Very Unlikely | Unlikely | Likely | Very Likely | Near Certain |

Scope creep is very likely to occur and may have significant impact on the project depending on the requests made and how far into production this occurs. An example of the moderate case would be the addition of a minor location or background NPC; an example of the significant case would be the addition of a fully interactive NPC or larger side-quest.

**Awareness**: team members should be be familiar with the causes of scope creep and wary of late-production additions or changes.

**Prevention**: to prevent scope creep from affecting development, all additions and changes to the project definition after the preproduction stage should be fully reviewed by all team members to avoid unnecessary workload; no significant changes may be made during the later stages of production or during post-production.

**Mitigation**: to reduce the impact of scope changes, any requirements connected with the change should be carefully considered and minimised through compromise if necessary.

**Recovery**: it is difficult to recover from scope creep, especially if it is not properly controlled, as attempting to revert changes once production is underway leads to sunk cost but allowing scope creep to continue can exacerbate tech deficit.