McMaster University | ENGINEERING Computing & Software

Ad Natura

Requirements Standard Plan

Felix Hurst, Marcos Hernandez-Rivero, BoWen Liu, Andy Liang

Version 1, 2025-09-30

Table of Contents

Academic Integrity Disclaimer

We would like to acknowledge that as a dedicated students of McMaster University, we have thoroughly read and comprehended the Academic Integrity Policy published by the university. We are committed to upholding the principles of academic honesty and integrity in all aspects of our educational journey. We understand the importance of acknowledging the work and ideas of others, and we pledge to ensure that all our academic endeavors are conducted with the utmost originality and compliance with the university's policy.

We affirm that the content presented in this document is entirely our own, and any external sources used have been appropriately cited and referenced.

Felix Hurst

As I submit my work, I, **Felix Hurst**, take full responsibility for the integrity of my work and promise to avoid any form of plagiarism, cheating, or dishonest behavior. This acknowledgment serves as a testament to my dedication to academic excellence and the fostering of a trustworthy academic community at McMaster University.

Marcos Hernandez-Rivero

As I submit my work, I, **Marcos Hernandez-Rivero**, take full responsibility for the integrity of my work and promise to avoid any form of plagiarism, cheating, or dishonest behavior. This acknowledgment serves as a testament to my dedication to academic excellence and the fostering of a trustworthy academic community at McMaster University.

BoWen Liu

As I submit my work, I, **BoWen Liu**, take full responsibility for the integrity of my work and promise to avoid any form of plagiarism, cheating, or dishonest behavior. This acknowledgment serves as a testament to my dedication to academic excellence and the fostering of a trustworthy academic community at McMaster University.

Andy Liang

As I submit my work, I, **Andy Liang**, take full responsibility for the integrity of my work and promise to avoid any form of plagiarism, cheating, or dishonest behavior. This acknowledgment serves as a testament to my dedication to academic excellence and the fostering of a trustworthy academic community at McMaster University.

Control Information

Version	**Delivery**		**Feedback**	
	Deadline	*Delivered*	*Received*	*Integrated*
V1	Oct 10, 2025	Oct 9, 2025		
V2				
V3				

(G) Goals

Control Information

Table 1. Ad Natura — Versionning Information — Goal Book

Section	Version	Lead	Delivered on	Reviewer	Approved on
G.1	1.0	Felix Hurst	2025-10-01	BoWen Liu	2025-10-07
G.2	1.0	Felix Hurst	2025-10-01	BoWen Liu	2025-10-07
G.3	1.0	Felix Hurst	2025-10-01	BoWen Liu	2025-10-07
G.4	1.0	BoWen Liu	2025-10-01	Andy Liang	2025-10-07
G.5	1.0	BoWen Liu	2025-10-01	Andy Liang	2025-10-07
G.6	1.0	Felix Hurst	2025-10-01	BoWen Liu	2025-10-07
G.7	1.0	Felix Hurst	2025-10-01	BoWen Liu	2025-10-07

(G.1) Context and Overall Objectives

In today’s society, a large portion of the population is not well educated regarding the impact humans have on our natural environment. By launching the video game Ad Natura, we will spread awareness about the importance of restoring harmony with nature. This will be done through the simulated organisms modelled by the slime mold algorithm, which the player must work with in order to solve puzzles and make progress, slowly leading them from a wasteland to a utopia.

Because the slime mold algorithm is not entirely predictable, the gameplay may vary a lot even when the same actions are taken. This contributes to the enjoyment and replayability of the game, by encouraging the player to keep trying new things to solve puzzles differently than before. Through high engagement, our message will get across even more strongly.

(G.2) Current situation

There is a wide variety of media, including video games, which present a message promoting environmentalism. However, these media only incorporate characters and objects with highly predictable and limited behaviour. They have yet to introduce a game entity that uses a sophisticated algorithm to mimic the behaviour of real world organisms, which are chaotic, though still able to be influenced in mostly predictable ways. Through our slime mold simulation and its highly organic behaviour, the player will form a stronger connection to the organism they are assisting. This will offer more insights to the player than existing media does currently.

(G.3) Expected Benefits

Ad Natura is expected to enhance the realism of a video game experience in two specific ways:

1. **The deformation of terrain.** The player will have access to a few specialized tools, each affecting a differently-sized and shaped chunk of terrain. For instance, a shovel will allow the deformation of a small square chunk of terrain at the player's feet. The deformed terrain is replaced by small and loose pixel particles, each with collision. The pixel particles can be moved or manipulated to eliminate or introduce obstacles in the environment. This may alter the position of water sources, light sources, etc.
2. **The slime mold organism.** There will be an organism (such as a plant or fungus) that is modelled by the slime mold algorithm. It will react to atmospheric conditions, such as light, heat and moisture. It will grow towards favourable conditions, and retract away from unfavourable conditions. The organism's growth can influence other parts of the environment, which can cause a pathway to open up that the player can traverse across to make progress. For example, the plant may grow large leaves the player can jump on, if the plant is given enough favourable conditions to grow in.

In summary, the gameplay involves the manipulation of terrain to move sources of favourable or unfavourable conditions in such a way that directs the slime mold organism to grow in the direction that allows the player to traverse forward.

The gameplay will teach the player that they must work with nature to succeed, and it will do so in ways other games have not done before. This will promote our message of environmentalism in a unique and deep way that will make a lasting memory for our players.

(G.4) Functionality Overview

The following are the functional requirements for our 2D pixel platformer game in no particular order.

1. Destructible environment. Players are able to destroy environment objects and terrain to shape the landscape to clear and/or traverse past obstacles. Certain destroyed objects and terrain can also release water, which the slime mold is attracted to.
2. Organic dynamic paths (slime mold). By shaping the environment and thereby water sources, the player is able to guide the slime mold to create dynamic paths to bypass obstacles.
3. Replayable and reproducible levels. Due to the indeterministic nature of slime mold and the unpredictable nature of the destructible environment, every playthrough of the game will introduce new challenges to the player. While being challenging and replayable, it is also always possible to complete each level with enough thought.
4. Verticality as design and game mechanic. Not only is verticality central to the theme of the game, it also influences how the player navigates levels and utilizes the game's mechanics to traverse height while watching out for falling objects.

(G.5) High-Level Usage Scenarios

UC1: Destructible Environment

1. Player aims their mouse at the destination of their tool strike.
2. When clicked, the struck part of the game object breaks apart, creating debris.
3. When an obstacle is destroyed or partially destroyed, the player can navigate past it or use the debris to their advantage.

UC2: Organic Dynamic Paths (Slime Mold)

1. Player manipulates the environment with tools or their body, causing water debris to accumulate in the intended location.
2. The slime mold traverses from its starting point to the location of the water debris.
3. The player can walk over the slime mold to traverse past obstacles.

UC3: Replayable and Reproducible Levels

1. The player clicks "Play" in the main menu and is taken to the first level.
2. The player gradually acquires all tools, with tutorials provided for each tool and mechanic.
3. The player uses available tools and the organic environment to traverse obstacles.
4. When the player reaches the designated point in the level, the next level is loaded.

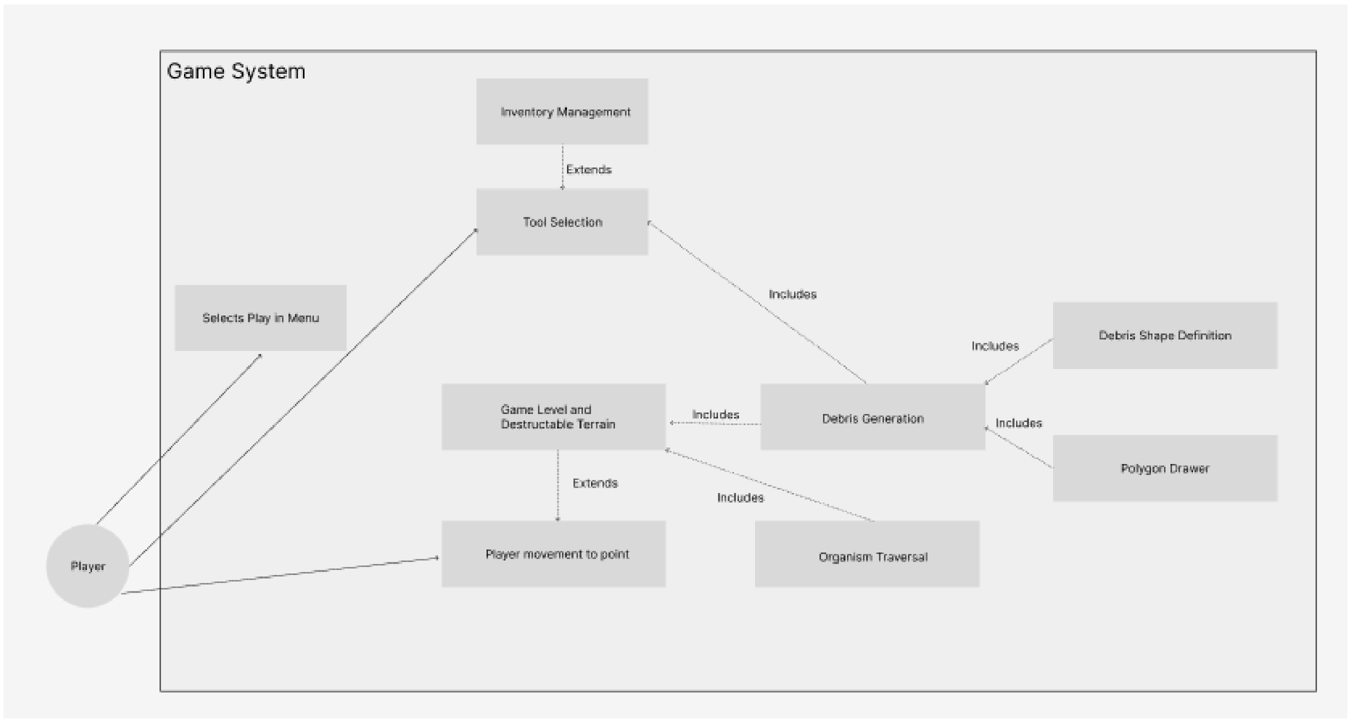

Figure 1. High Level Usage Diagram

(G.6) Limitations and Exclusions

Below is a list of limitations that the game will not do:

- The game will not be designed for multiplayer. It will be single player only.
- The game will not strive to be completely and utterly realistic in player capabilities, environment behaviour, etc. Creative liberties will be taken to adapt the concept into an enjoyable 2D platformer game, while maintaining core features that simulate the real world closely enough for our message to get across as intended.

(G.7) Stakeholders and requirements sources

(G.7.1) Direct Stakeholders

Casual Gamers

Gamers who play games in a non-competitive capacity and as such value the experience rather than solely a challenge. Being familiar with similar games, basic game controls should be intuitive to them. These players may enjoy the game just for its puzzles, or may also spread our message after playing the game.

Environmentalists

Activists whose main purpose is to spread awareness of the importance of protecting our natural environment, habitats and wildlife. They would be interested in the success of our game.

Educators

Instructors may use technology to assist in the teaching of their classes. Our game may be used for educational purposes in a classroom environment.

(G.7.2) Indirect Stakeholders

Valve

Valve operates Steam, the shopfront and regulator of the sale of our game. Its algorithms and policies can determine the success or failure of our game.

Entertainment Software Rating Board (ESRB)

This organization will assign an age rating to our game, determining what age range our audience will be.

(G.7.3) Requirements Sources

- **Reviews** for games sharing the same genre(s) as ours, will be read, with strong praises and criticisms being noted.
- **Steam's Terms of Service:** These terms must be followed in order for us to sell our game on the Steam platform.
    - https://store.steampowered.com/eula/2585120_eula_1
- **ESRB's Rating System:** To ensure our game meets an "E for Everyone" rating to maximize our potential audience, we will need to follow their rating guidelines.
    - https://www.esrb.org/ratings-guide/

(E) Environment

The Environment book describes the application domain and external context, physical or virtual (or a mix), in which the system will operate. [BM22]

Control Information

Table 2. Ad Natura — Versionning Information — Environment Book

Section	Version	Lead	Delivered	Reviewer	Approved
E.1	1.0	Andy Liang	2025-10-07	Marcos Hernandez-Rivero	2025-10-07
E.2	1.0	Andy Liang	2025-10-07	Marcos Hernandez-Rivero	2025-10-07
E.3	1.0	Andy Liang	2025-10-07	Marcos Hernandez-Rivero	2025-10-07
E.4	1.0	Andy Liang	2025-10-07	Marcos Hernandez-Rivero	2025-10-07
E.5	1.0	Andy Liang	2025-10-07	Marcos Hernandez-Rivero	2025-10-07
E.6	1.0	Andy Liang	2025-10-07	Marcos Hernandez-Rivero	2025-10-07

(E.1) Glossary

Terms (alphabetical)

Benchmark Scene

A simple test level used to check performance and behavior consistently.

Cell

The smallest terrain unit the game reasons about (a tile in the grid).

Development Scene

A flexible test scene used for debugging and iteration; not used for performance acceptance or timing. Distinct from Benchmark Scene.

Field

A number stored per cell that influences behavior (e.g., water, heat, light).

Frame
: One step of update/render in the game loop.

Heat Field
: A field where higher values mean “hotter” areas that the organism tends to avoid.

Input Device
: Keyboard + mouse or a standard Windows gamepad.

Latency
: Time between an environment change and when that change is made available to the system.

Light Field
: A field representing illumination intensity across the level.

Minimum Spec
: The lowest hardware/OS we support for stable play (see Problem Statement for the numbers).

Organism
: The slime-mould-like entity that moves through levels.

Publish (Environment)
: Making updated fields/terrain available to the system after a change.

Player Traversal
: How the player character moves and navigates levels (separate from Organism Traversal).

Stimulus
: A player/scripted action that changes fields or terrain (e.g., reveal water, add heat).

Terrain
: The destructible/editable level geometry made of cells.

Traversal
: How the organism moves through the level under the influence of fields and terrain.

Water Field
: A field where higher values mean more water availability that tends to attract the organism.

Acronyms

CPU — Central Processing Unit GPU — Graphics Processing Unit FPS — Frames Per Second I/O — Input / Output CI — Continuous Integration PRNG — Pseudo-Random Number Generator

(E.2) Components

Components (things outside the software that we read from or affect)

Player (human)

The person solving puzzles and triggering changes in the world.

Input devices (Input Device)

Keyboard + mouse; standard Windows gamepad (XInput-style).

Display & audio

Monitor and speakers/headset the game renders to and plays through.

Platform & OS

Windows 10+ (x64). Provides process, windowing, timers, and device drivers.

Hardware resources

CPU, GPU, RAM, and storage available to the game.

File system

Location for saves, settings, and logs (user-writable path).

Clock/time base

Wall-clock time used to measure delays/latency and advance frames.

Terrain (grid of cells)

Destructible/editable level geometry the player modifies.

Environmental fields

Water, heat, and light values defined per cell that influence behavior.

Stimulus sources/tools

In-level items or actions that change fields or terrain (e.g., "reveal water", "add heat", "adjust light").

Obstacles/material states

Passable vs. blocked regions, hazards, and surfaces that constrain movement.

Collision/physics space

The world’s physical layout (static/dynamic) that defines contact and blocking—described here only as facts of the world.

(E.3) Constraints

Constraints (non-negotiables from the environment)

Platform & OS

Windows 10+ (x64) only for release. No macOS/Linux requirements.

Minimum hardware floor

Must operate on the project's Minimum Spec. All targets/acceptance numbers live in S.6.

Input availability

Keyboard + mouse must be fully usable. XInput-style gamepads are supported but not required for play.

Display baseline

Game must be playable and UI text readable at 1280×720 (windowed or fullscreen).

Graphics API class

Target a DX11-class GPU typical of the Minimum Spec era; no ray-tracing or VR-specific features are assumed.

File system & permissions

Saves/settings/logs must live in a user-writable path. No administrator privileges required.

Time base

All delays/latencies are measured in wall-clock seconds; the game must tolerate normal OS scheduling jitter.

Numeric sanity

Environment values (fields, timers, counters) must be finite (no NaN/Inf) and within project-defined safe ranges.

World envelope

Scene size, active objects, and editable terrain density must stay within bounds that keep playability on Minimum Spec (exact budgets verified in S.6).

I/O devices

Support standard Windows audio output and the primary display device; no external sensors or network devices are required.

Determinism for tests

For verification scenes only, a seeded run must be reproducible on the same machine; live gameplay is allowed to be non-deterministic.

(E.4) Assumptions

Assumptions (helpful truths we will treat as given)

GPU drivers

A DX11-class GPU driver is installed and working on the machine.

Offline play

The game is fully playable without an internet connection. (Future online features, if any, are optional.)

Display

At least one display is available at 1280×720 or higher. OS DPI scaling is within common ranges (100–150%).

Audio device

A standard Windows audio output device exists; if none, the game continues without audio.

Input

A keyboard and mouse are available. A gamepad may or may not be present; all actions remain doable on keyboard/mouse.

File system

The user has a writable, non-admin path for saves/settings/logs (e.g., under their profile). No administrator rights are needed.

Clock

A stable wall-clock/monotonic timer is available for measuring delays/latency and frame timing.

Locale

UI is English for MVP; number/date formatting does not depend on system locale.

Color space

Displays run in SDR/sRGB. HDR or wide-gamut displays are treated as SDR.

Power/thermal

Normal OS scheduling/throttling can occur (laptops on battery). The game does not require real-time guarantees.

Controller API

If a controller is used, it follows standard XInput mappings (no custom vendor remaps required).

CI test host

A Windows runner with a DX11-class GPU can execute automated scenes (CI); seeded runs are reproducible on the same machine using a PRNG.

(E.5) Effects

Effects (what changes in the environment when the system acts)

"Publish" = the environment makes updated data available to the system. Exact system responses live in S.* and will be verified in S.6.

Trigger	Environment property changed	Publish ≤	Scope	Persistence
Reveal water	Increase Water Field on affected cells (clamped to safe range)	1.0 s	Local (tool AOE)	Until overwritten or cleared
Add heat	Increase Heat Field; mark "hot zones" above threshold	1.0 s	Local	Until overwritten or cleared
Adjust light	Modify Light Field intensity on cells	1.0 s	Local/Global (per tool)	Until overwritten or cleared
Carve terrain	Change cells from blocked → passable in terrain; update collision space	1.0 s	Local	Persistent until edited again
Build terrain	Change cells from passable → blocked in terrain; update collision space	1.0 s	Local	Persistent until edited again
Place obstacle / hazard	Mark cells as blocked / hazardous (non-traversable)	1.0 s	Local	Until removed
Reset / undo	Revert the most recent environment change (fields or terrain)	1.0 s	Local	Restores prior values/states

Platform / OS facing effects (non-level): * Saves & settings: write/update small files under a user-writable path. * Logs: append diagnostic entries to a text log (rotation defined in Project). * Display: present frames to the primary display device. * Audio: play sound on the current output device; if none, continue silently. * Controller rumble (if present): activate vibration for a specified duration.

(See E.6 for invariants like "no NaN/Inf" and "a cell cannot be both passable and blocked in the same frame".)

(E.6) Invariants

Invariants (facts that must always hold)

Finite fields

All fields are finite numbers (no NaN/Inf) and kept within configured safe ranges each frame.

Atomic terrain edits

A terrain edit applies atomically within a frame; readers see either the pre-edit or post-edit state, never a mix.

Publish bound

After a valid trigger, environment updates are published within ≤1.0 s (see latency).

Frame immutability

Once a frame advances, past environment values are immutable.

Collision coherence

Collision/physics space matches the terrain state for the same frame.

Effect scope

A local change only affects its declared scope (no unintended global spillover).

Seeded reproducibility (tests)

In verification scenes, a fixed PRNG seed produces the same published environment sequence on the same machine.

File hygiene

Environment file writes (saves/settings/logs) target user-writable paths and complete atomically (no partial/corrupt files).

Min-spec gating

Levels labeled “min-spec” never exceed content/material/asset budgets required for Minimum Spec playability.

Bounds & clamps

All field values are clamped to their min/max before publish.

Monotonic time

Timers used for publish/latency derive from a monotonic clock and do not go backward.

World envelope respected

Scene size and editable terrain density remain within the bounds defined by constraints in E.3 during play and tests.

(S) System

the System book refines the Goal one by focusing on more detailed requirements.

Control Information

Section	Version	Lead	Delivered	Reviewer	Approved
S.1	1.0	BoWen Liu	2025-10-01	Andy Liang	2025-10-07
S.2	1.0	BoWen Liu	2025-10-01	Andy Liang	2025-10-07
S.3	1.0	BoWen Liu	2025-10-01	Andy Liang	2025-10-07
S.4	1.0	BoWen Liu	2025-10-01	Andy Liang	2025-10-07
S.5	1.0	Andy Liang	2025-10-07	Marcos Hernandez-Rivero	2025-10-07
[s6]	1.0	Andy Liang	2025-10-07	Marcos Hernandez-Rivero	2025-10-07

(S.1) Components

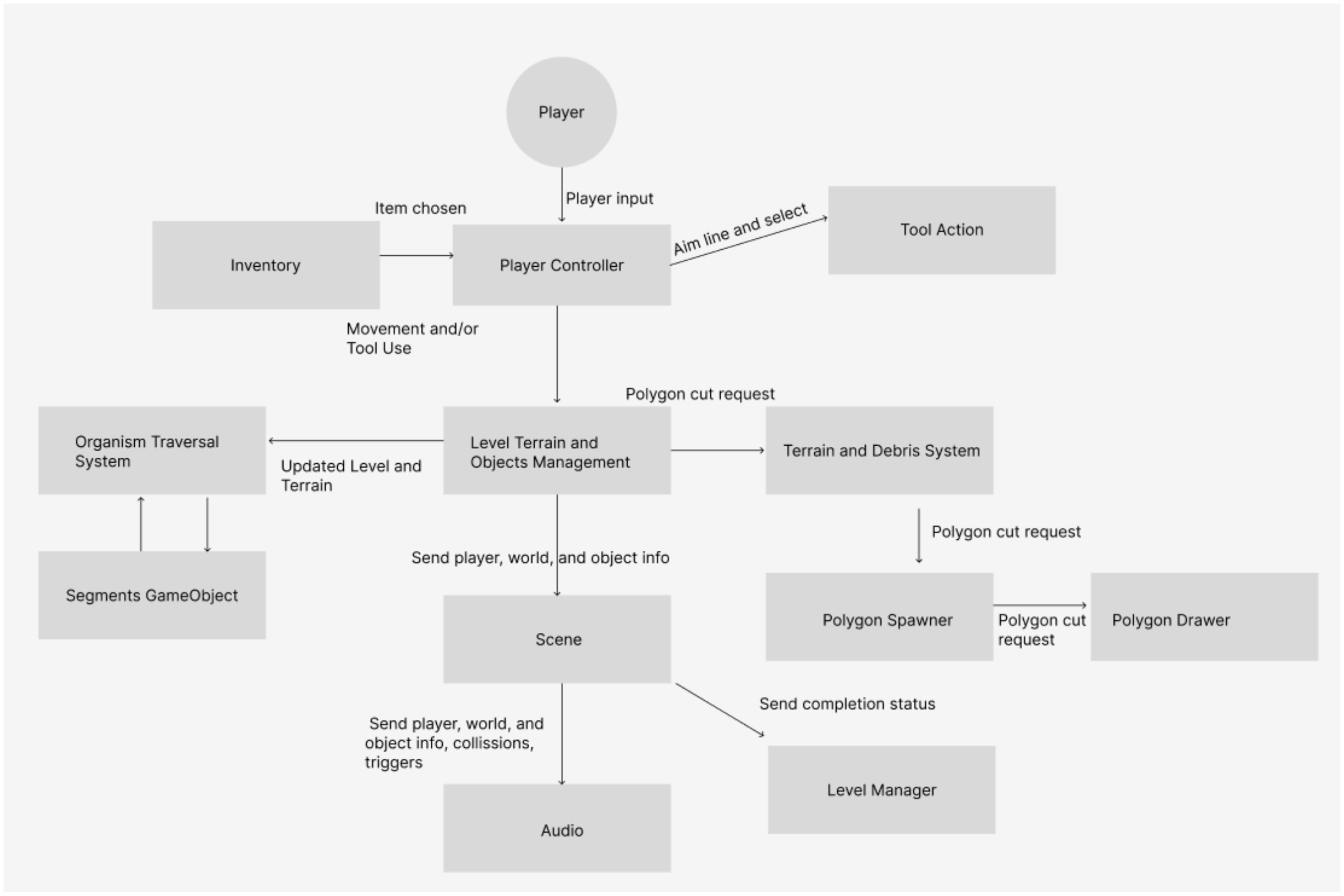

Figure 2. Components Diagram

The major components of our project are as follows:

1. Debris System
   Generates debris objects and reshapes preexisting objects for terrain traversal and environmental manipulation. It is essential that this component behaves as expected, and that level design is built with this system in mind to avoid game-breaking bugs.
2. Inventory System
   Selects tools, each of which has a unique effect on objects and terrain. Game levels and other systems must be designed with each tool type in mind.
3. Player Controller
   Handles player input and movement. It should be designed in coordination with the slime traversal and debris systems. Considerations such as jump height and terrain scaling should be factored in.
4. Organic Pathway System
   Determines how the slime mold traverses the game level. The system should be tested and refined to ensure effective pathways are generated.

(S.2) Functionality

(S.2.1) Save Data Manager

Functional Requirements

1. **Create save data:** Upon launching the game, if there is no save file found, a new save file will be generated on the user's computer file system. (F211)
2. **Restore from save data:** Upon launching the game, if there is an existing save file found, then that file will be loaded to restore the player's progress from their previous session. (F212)
3. **Auto-save:** Each time the player completes a puzzle and progresses to the next scene, the game will automatically save their progress, updating the save file. (F213)
4. **Reset puzzle:** The player has the option to reset a puzzle room to start over, in case they got lost or softlocked. (F214)

Non-Functional Requirements

1. **Data privacy:** Only data related to game progression is saved to the save file; personal information is not collected. (NF211)
2. **Compression:** The save file will be in a compressed, binary format to minimize the file space required to store it on the player's computer. (NF212)

(S.2.2) Game Objects & Environment

Functional Requirements

1. **Destructible environment:** The system shall simulate the usage of destructive tools, which, when interacting with terrain designated as destructible, will decompose the terrain game object into several smaller blocks of terrain. The sum of the areas of the decomposed blocks is roughly equal to the area of the terrain that was removed. The system shall correctly identify and produce logical debris amount, colour and shape, based on the attributes of the original object. (F221)
2. **Slime mold simulation:** The system shall simulate an organism whose behaviour is modelled by the slime mold algorithm, which shall be nondeterministic but largely predictable and manipulable, based on attraction or repulsion to/from water, light and heat. (F222)
3. **Puzzle progression:** The system shall contain environmental puzzles with a definite start and end which the player can traverse through by solving the puzzle; each puzzle requires the player to assist the slime mold organism in order to reveal a path forward. (F223)

Non-Functional Requirements

1. **Game difficulty and length:** The puzzles in the game shall be presented in chunks (referred to as "levels"), such that each level can be completed within a reasonable time frame (an estimated 15-20 minutes). (NF221)
2. **Replayability and randomness:** The nondeterministic behaviour of the slime mold organism and the

decomposed terrain generation shall create variance in a way that improves replayability of the game, while the puzzles themselves can still be solved regardless of the random elements of game behaviour. (NF222)

(S.2.3) Safety & Security

Non-Functional Requirements

1. **Stability:** The system shall experience crashes no more frequently than once every two hours on average. (NF231)
2. **Crash logs:** Should a crash occur, the system shall display as much detail about the crash as possible in a pop-up window, for diagnostic purposes. Developer contact information will be provided for the purpose of bug reporting. (NF232)
3. **Privileges:** The system shall be able to run without requiring administrator privileges. (NF233)
4. **Software vulnerabilities:** Any Unity security vulnerabilities shall be patched in a timely fashion and distributed in a software update. (NF234)

(S.3) Interfaces

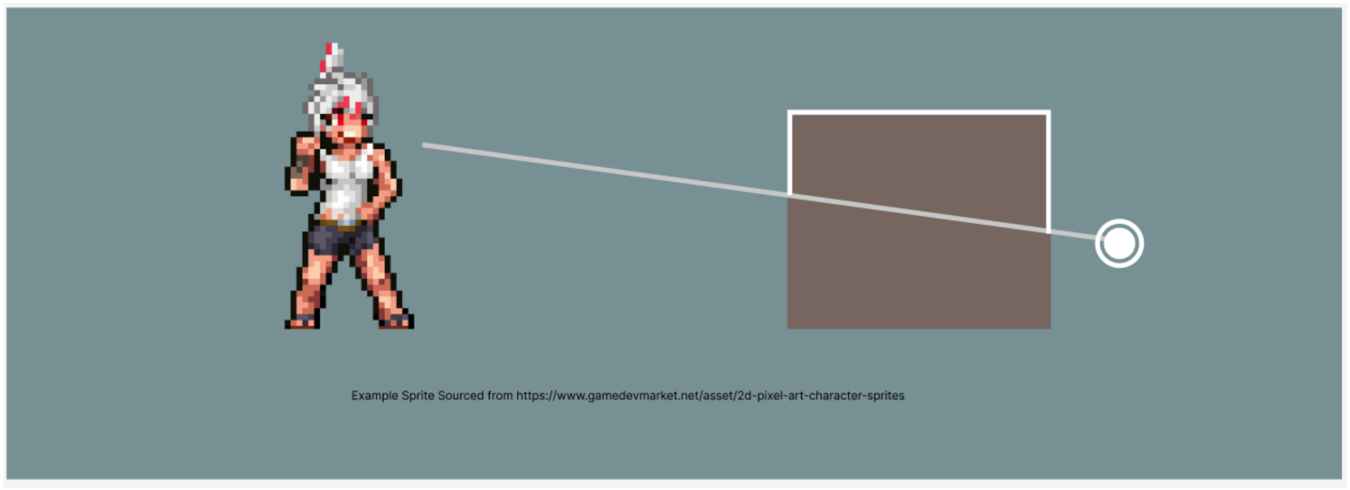
Example Sprite Sourced from https://www.gamedevmarket.net/asset/2d-pixel-art-character-sprites

Figure 3. Aiming Interface

Players mouse draws a line with it and the character symbolizing the path in which the strike it will take from the character to the mouse. Any objects that will be destroyed by the strike will be highlighted in an outline. Player will click to strike which is always on the circular button.

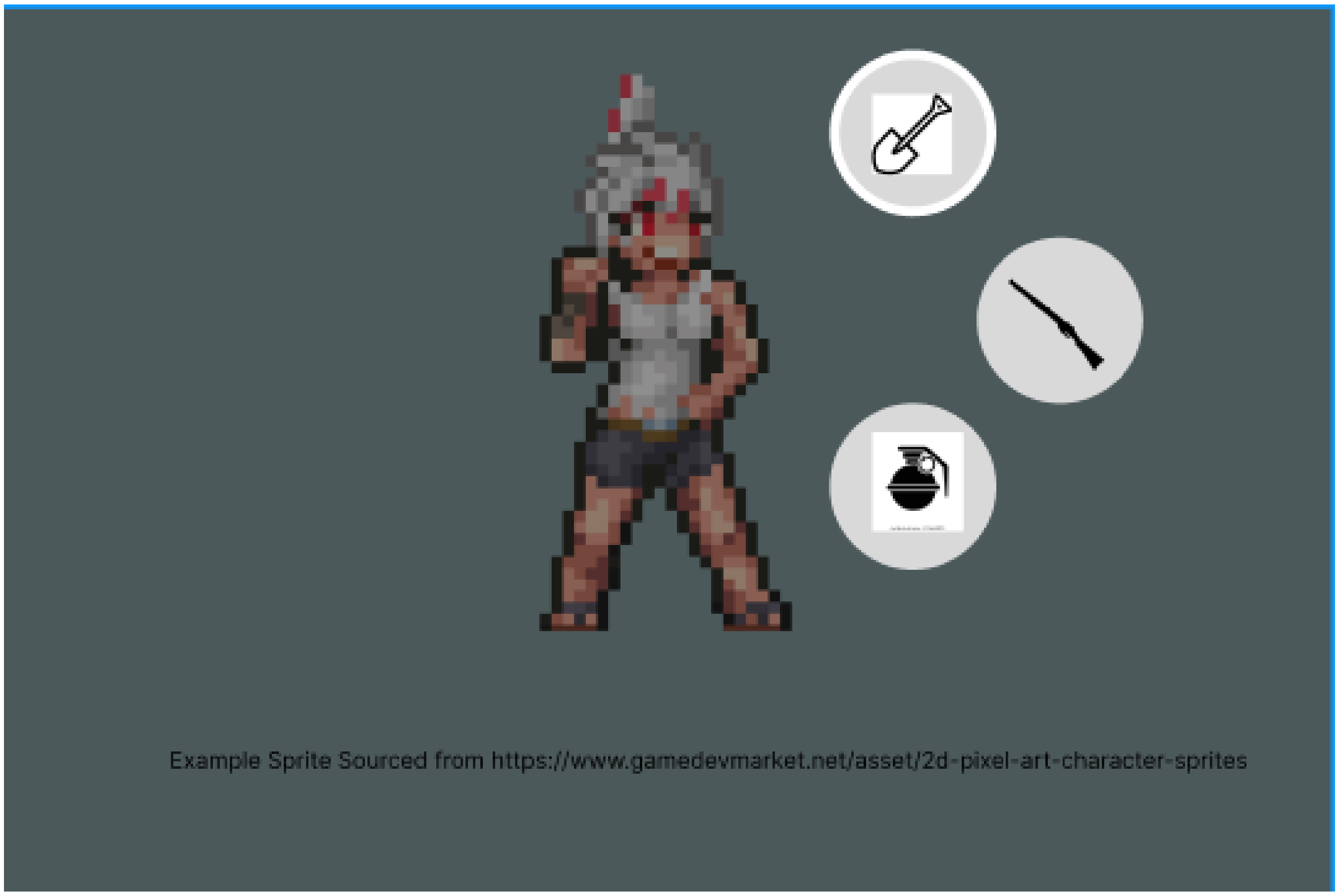
Example Sprite Sourced from https://www.gamedevmarket.net/asset/2d-pixel-art-character-sprites

Figure 4. Tool Selection Interface

Player presses the CTRL button which will pause the game and can use the mousewheel to scroll to the correct tool in which will be equipped once they let go of the button.

(S.4) Detailed Usage Scenarios

Destructible Environment

Use Case
UC1

Primary Actor
Player

Precondition
Player has access to a reachable destructible game object.

Trigger
Player strikes the object with their tool, causing damage above a certain threshold.

Main Success Scenario

- Player aims the tool at a game object or terrain.
- Objects in the aim path are highlighted to show the chunk that will be broken off.
- Player clicks the left mouse button to execute the strike.
- The chunk is broken off and breaks down further into smaller debris that can be manipulated and/or traversed by the player.

Secondary Scenario

- The object or terrain the player carves off contains water, which is released.
- Water accumulates at a point that can be utilized by the slime mold as a destination to traverse toward.

Success Postcondition

The player can traverse obstacles by breaking up game objects and debris, and/or by guiding water flow to unlock pathways and guide slime mold.

Organic Dynamic Paths (Slime Mold)

Use Case
UC2

Primary Actor
Player

Precondition
Slime mold sources exist in the game.

Trigger
Flowing water is released or guided by the player.

Main Success Scenario

- The player releases a water source at the correct location by striking an object or solving a puzzle.
- The slime mold finds a path from its original location to the water source.
- The player walks over the organic pathway to traverse previously unreachable areas.

Secondary Scenario

No alternate scenario applies.

Success Postcondition

The player can traverse obstacles by breaking up game objects and debris, and/or by guiding water flow to unlock pathways and guide slime mold.

(S.5) Prioritization

Prioritization approach (MoSCoW)

We classify features by criticality to gameplay and to meeting constraints on Minimum Spec: * **Must** — core to puzzle loop and acceptance; cannot ship without. * **Should** — important quality/usability; de-scope only if Musts are safe. * **Could** — nice to have; first to drop under time pressure. * **Won't (this release)** — explicitly out of scope.

ID	Item	Priority	Why it matters	Notes / Dependencies
S5-PR-1	Stimulus → field updates ([e5-reveal-water], [e5-add-heat], [e5-adjust-light])	Must	Core loop: player applies stimuli that change fields the organism follows.	Bound by publish ≤1.0 s (Publish bound).
S5-PR-2	Traversal reacts to fields	Must	Without traversal responding to fields, puzzles do not function.	Acceptance in S.6; uses benchmark scene.
S5-PR-3	Terrain edit tools (carve/build)	Must	Player agency over terrain is required to solve puzzles.	Keep atomic per frame (Atomic terrain edits).
S5-PR-4	Finite fields & clamps	Must	Prevents corrupt state / crashes.	See Finite fields, Bounds & clamps.
S5-PR-5	Publish latency ≤ 1.0 s	Must	Feedback must feel immediate and testable.	See Latency and Publish bound.
S5-PR-6	Playable on Minimum Spec with ≥30 FPS (benchmark)	Must	Meets platform promise; grading/acceptance hinge on this.	Targets verified in S.6 using benchmark scene.
S5-PR-7	Keyboard+mouse fully playable (rebindable)	Must	Accessibility and lab machines; no controller required.	Maps defined in S.3; saves to user path.
S5-PR-8	Saves/settings/logs to user path	Must	Persistence and debugability.	Atomic writes (File hygiene).

ID	Item	Priority	Why it matters	Notes / Dependencies
S5-PR-9	Seeded reproducibility in test scenes	Must	CI/marking require repeatable runs.	See Seeded reproducibility (tests).
S5-PR-10	XInput gamepad support + rumble toggle	Should	Improves experience for many players.	Falls back to K/M (S5-PR-7).
S5-PR-11	Undo/redo last environment change	Should	Reduces frustration while experimenting.	Applies to fields and terrain; respects invariants.
S5-PR-12	Readable UI at 1280×720; basic color-safe palette	Should	Meets display baseline and color-vision needs.	Numbers in S.6; palette toggle optional.
S5-PR-13	Logging with rotation	Should	Helps QA; prevents unbounded disk use.	Project section defines policy.
S5-PR-14	Benchmark scene + CI smoke test	Should	Keeps perf and publish bounds from regressing.	Uses fixed seed and scripted inputs.
S5-PR-15	Extra stimuli (e.g., wind/toxin)	Could	Increases puzzle variety.	Only if Must/Should are solid.
S5-PR-16	Visual polish (advanced VFX/HDR)	Could	Aesthetic improvement; not required for loop.	Ensure does not harm Min Spec perf.
S5-PR-17	Photo/replay mode	Could	Portfolio value; non-essential to puzzles.	Defer if schedule tight.
S5-PR-18	Online features / networking	Won't	Out of scope for this release.	N/A.
S5-PR-19	macOS/Linux builds	Won't	Platform scope is Windows-only.	Could revisit post-release.
S5-PR-20	VR / ray tracing	Won't	Incompatible with perf/Min Spec constraints.	N/A.

De-scope order & guardrails

- Drop **Could** items first; then **Should** only if **S5-PR-1..9** remain intact.
- Never violate E.6 invariants when de-scoping; Must items are non-negotiable.
- Any change to **S5-PR-6** (perf target) or **S5-PR-5** (latency bound) requires team approval and rubric impact review.

Verification setup (baseline)

- Test scene: Benchmark Scene with scripted inputs.
- Hardware target: Minimum Spec (lab box or equivalent VM).

- Timing: wall-clock; use monotonic timers for latency (see latency).
- Seeds: fixed PRNG seed for reproducible runs in CI (CI).
- Data sources: in-game counters/markers and lightweight logs; no debugger required.

Acceptance matrix — Must items

ID	What we verify	How we test	Measure	Accept
S6-LAT-1	Publish latency for field changes ([e5-reveal-water], [e5-add-heat], [e5-adjust-light])	In the benchmark scene, trigger each effect; record time from trigger to environment publish event	Δt (trigger → publish)	≤ 1.0 s (see Publish bound)
S6-TRV-1	Traversal reacts to published fields	After S6-LAT-1, observe organism path change under fixed seed	Time to first path deviation	≤ 1.0 s after publish (bench scene)
S6-PERF-1	Playability on Minimum Spec	Run benchmark scene 5 minutes	Avg FPS, 99th-pct frame time	Avg ≥ 30 FPS; 99th-pct ≤ 60 ms
S6-MEM-1	Memory ceiling during gameplay	10-minute play session with edits and stimuli	Peak RAM (task/process)	≤ 3.0 GB
S6-LOAD-1	Scene load time budget	Cold-start load of 3 representative levels on HDD-baseline	Time to first interactive frame	≤ 10 s each
S6-IO-1	Keyboard + mouse fully playable	Complete tutorial goals with no gamepad; rebind all actions	Completeness; bind persistence	100% actions rebindable; bindings persist across restart
S6-FILE-1	File hygiene for saves/settings/logs	Save, restart, reload; inspect files during write	Path + atomicity	User-writable path; no partial/corrupt files (see File hygiene)
S6-DET-1	Seeded reproducibility in test scenes	Run headless twice in CI with same seed	Event/log hashes	Identical outputs on same machine (see Seeded reproducibility (tests))
S6-COLL-1	Collision matches terrain edits	Carve/build then sample collision that frame	Contact vs. cell state	Match within same frame (see Collision coherence)
S6-FIN-1	Finite/clamped environment values	Sweep stimuli to extremes; log field samples	NaN/Inf count; clamp range	Zero NaN/Inf; all within configured bounds (see Finite fields, Bounds & clamps)

Acceptance — Should items

ID	What we verify	How we test	Measure	Accept
S6-UNDO-1	Undo/redo last environment change	Perform edit; undo; redo; sample fields/terrain	State equivalence	Restores exact prior values/states
S6-LOG-1	Logging with rotation	Generate 30 min of logs; trigger rotation	Max file size/count	Within configured cap; no data loss
S6-UI-1	Readability at 1280×720	UI audit in bench scene at 1280×720	Text height/contrast	Meets project baseline (readable without scaling)

Gating & traceability

- A build **passes** S.6 only if all **Must** IDs above are met on Minimum Spec in the benchmark scene.
- Failures on **Should** items are acceptable only if all **Must** remain green and the de-scope order in S.5 is followed.
- Relevant invariants and effects referenced: Publish bound, Finite fields, Bounds & clamps, Atomic terrain edits, Collision coherence, Seeded reproducibility (tests), [e5-reveal-water], [e5-add-heat], [e5-adjust-light].

(P) Project

Control Information

Section	Version	Lead	Delivered	Reviewer	Approved
P.1	1.0	Marcos Hernandez-Rivero	2025-10-03	Felix Hurst	2025-10-07
P.2	1.0	Marcos Hernandez-Rivero	2025-10-03	Felix Hurst	2025-10-07
P.3	1.0	Marcos Hernandez-Rivero	2025-10-03	Felix Hurst	2025-10-07
P.4	1.0	Marcos Hernandez-Rivero	2025-10-03	Felix Hurst	2025-10-07
P.5	1.0	Marcos Hernandez-Rivero	2025-10-03	Felix Hurst	2025-10-07
P.6	1.0	Marcos Hernandez-Rivero	2025-10-03	Felix Hurst	2025-10-07
P.7	1.0	Marcos Hernandez-Rivero	2025-10-03	Felix Hurst	2025-10-07

(P.1) Roles and personnel

- **Team Leader:** Responsible for keeping the team informed and reminded of deliverable deadlines, communicating with instructors, and having final say on topics the team cannot agree on. This role typically requires organizational and leadership skills, with experience in coordinating collaborative projects.
- **Developers / Programmers:** These individuals are responsible for developing and maintaining the application, ensuring it functions properly and that it matches the requirements of what the product owner wants. Within this group, programmers specifically write the game code. Together, they ensure all game systems work as expected. They are usually required to have prior development experience, ideally in collaborative teams and with relevant game engines or frameworks.
- **Testers:** Responsible for testing the functionality of all the game systems, including performance,

graphics, sound, gameplay, and ensuring that both the mechanics function as intended and that usability requirements (such as accessibility and clear player communication) are met. They typically have QA testing experience, ideally in the context of video games.

- **Product Manager:** Oversees the project from planning to deployment, coordinating between teams to ensure progress is on schedule and risks are managed effectively. They also communicate with stakeholders to obtain requirements and provide progress updates. They typically have project management experience, ideally in game development contexts.
- **Level Designers:** Responsible for creating puzzles and challenges for the game. They need a strong understanding of what captures player attention, how to communicate game information effectively, and how to work with developers to ensure player experience is prioritized. They typically have experience in game design or interactive storytelling.
- **Art Director:** Determines the aesthetic direction of the game's artwork, ensuring a cohesive visual style. This role requires strong artistic vision and experience in guiding creative teams.
- **Character Artist:** Creates sprites of the player character, requiring both artistic skill and familiarity with pixel art or digital illustration techniques relevant to the project's style.
- **Environment Artist:** Responsible for drawing assets of the game world, such as backgrounds, terrain, and interactive objects. This role requires creativity and experience in environment or asset design.
- **Music Director:** Determines the aesthetic direction of the game's soundtrack, ensuring that the music aligns with the game's mood and gameplay. This role requires musical expertise and familiarity with game audio design.
- **Composer:** Composes original music for the game, requiring strong composition skills and the ability to create pieces that enhance immersion and match gameplay pacing.
- **Meeting Chair:** Keeps the team on track during meetings and ensures that all agenda topics are discussed. This role requires organizational and communication skills, as well as the ability to maintain meeting focus.
- **Reviewer:** Reviews pull requests and determines whether they should be accepted or rejected. This role requires technical knowledge of the project's codebase, attention to detail, and experience in peer code review.

(P.2) Imposed technical choices

- **Game Engine:** The Unity 6.1 game engine has been chosen. It is highly documented, widely used in both academic and professional contexts, and offers extensive support for complex physics, AI behaviors, and modular world-building. Its cross-platform deployment capability ensures the game can later be expanded to different devices if required.
- **Programming Language:** C# has been selected as the primary scripting language. Unity natively supports C#, making it the most efficient choice. It provides strong performance, clear syntax, and robust libraries, ensuring that game logic and physics calculations can be implemented efficiently.
- **Version Control:** Git and GitHub will be used for collaborative development and version control. This choice ensures proper tracking of contributions, clear project history, and facilitates pull request

reviews, continuous integration, and rollback when necessary.

(P.3) Schedule and milestones

Sprint 0 (September 2 – December 4): Foundation and Planning - Technical Environment setup (Destructible environment, player character, player tools, Slime mold traversal) - V&V Plan - Design Document Revision 0 - Proof of Concept Demonstration

Sprint 1 (January 5 – January 16): Level Design and Demonstration - Level Design Document - Level Design Demonstration

Sprint 2 (January 5 – February 28): Integration and Documentation - System Integration - Design Document Revision 1

Sprint 3 (February 16 - April 7): Finalization and Delivery - Bug Fixing - V&V Report and Extras - Final Demonstration (Revision 1)

(P.4) Tasks and deliverables

Sprint 0 (September 2 – December 4): Foundation and Planning

- **Technical Environment Setup:** Establish the game's foundational systems, including destructible environment mechanics, player character implementation, player tools, and slime mold traversal. **Expected Outcome:** A working prototype environment where all fundamental gameplay mechanics are present in basic form, serving as the baseline for future iterations.
- **V&V Plan:** Create the first version of the Verification and Validation (V&V) plan, outlining methods for testing functionality, performance, and usability. **Expected Outcome:** A documented strategy that guides how the game' features will be tested throughout development.
- **Design Document Revision 0:** Produce an early revision of the game design document capturing initial concepts, gameplay vision, and scope. **Expected Outcome:** A draft design framework that communicates the project's direction to stakeholders and supports proof-of-concept work.
- **Proof of Concept Demonstration:** Build and present a minimal implementation that demonstrates feasibility of key gameplay ideas. **Expected Outcome:** A validated concept showing that the proposed mechanics and tools can be implemented effectively.

Justification: This sprint establishes the technical and conceptual foundation of the project. By demonstrating feasibility and producing the first design materials, the team ensures that subsequent sprints have a stable baseline to build upon.

Sprint 1 (January 5 – January 16): Level Design and Demonstration

- **Level Design Revision 0:** Develop the first set of puzzles, environments, and challenges for players. **Expected Outcome:** A playable draft level that demonstrates intended design philosophy and player experience goals.

- **Level Design Demonstration:** Present the initial level to gather early feedback on gameplay flow, challenge balance, and communication of mechanics. **Expected Outcome:** Feedback that validates or informs revisions of level design practices.

Justification: This sprint introduces the first playable content, allowing stakeholders and testers to evaluate how players will experience the game. Early validation of level design ensures that future development stays aligned with gameplay goals.

Sprint 2 (January 5 – February 28): Integration and Documentation

- **Integration:** Combine different game systems (art, code, mechanics, audio) into a unified build. **Expected Outcome:** A cohesive and playable version of the game that integrates all developed components.
- **Design Document Revision 1:** Update the design document to reflect changes made during integration and align it with the current build. **Expected Outcome:** A refined and consistent design document that matches the state of the game and guides further development.

Justification: This sprint focuses on combining individual contributions into a working whole. It ensures that design documentation stays current with implementation, reducing the risk of misalignment as development progresses.

Sprint 3 (February 16 - April 7): Finalization and Delivery

- **Bug Fixing:** Identify and resolve defects across systems, including gameplay, performance, audio, and usability. **Expected Outcome:** A more stable and polished game experience, ready for final testing and demonstration.
- **V&V Report and Extras:** Conduct structured testing according to the V&V plan, and produce the first version of the report. Additional features or enhancements are also included if feasible. **Expected Outcome:** Verified evidence of functionality and a clear report documenting testing activities, along with any implemented extras.
- **Final Demonstration:** Deliver the game in its most complete and stable form for final stakeholder review. **Expected Outcome:** A fully playable game with documented testing results, representing the final milestone of the project.

Justification: This sprint ensures the project reaches a stable and polished final state. Verification and validation activities confirm functionality, while the final demonstration provides stakeholders with confidence in the product's quality and readiness.

(P.5) Required technology elements

The success of the project depends on the availability of the following external technologies and systems:

- **Unity 6.1:** The core game engine. Availability and continued support from Unity are critical. Any disruption to licensing, version stability, or major API changes would directly impact the project timeline.

- **C# Language and Unity Libraries:** The scripting environment, including Unity's physics and rendering libraries, is essential for implementing game systems. If updates deprecate critical libraries, significant refactoring may be required.
- **GitHub:** The project depends on GitHub for version control, issue tracking, and collaboration. Extended downtime or loss of access would risk development continuity.
- **Hardware:**
    - Development workstations capable of running Unity 6.1, Visual Studio, and testing builds efficiently (Our own computers).
    - Target platforms (PC with sufficient specifications) for testing destructible environments, traversal, and player tools.
- **Testing Frameworks:** External Unity libraries and Unity Test Runner will be required for automated unit testing. Their availability ensures reliable verification of mechanics, performance, and stability.
- **Asset Tools:** External tools such as digital art programs (e.g., Krita, Photoshop alternatives), audio editing software (e.g., Audacity), and 3D modeling tools may be required for assets. Availability of license-free resources is also important.

(P.6) Risk and mitigation analysis

Obstacles and Mitigation Strategies by Sprint

The overall critical risk is Scope Creep and uncontrolled technical debt, which could compromise the final delivery quality. The primary mitigation strategy is a rigorous adherence to the initial design scope and a strict policy of refactoring code before major new features are added.

Sprint 0 (Foundation and Planning)

- Obstacle: Technical Environment Setup takes longer than anticipated due to complexity of core mechanics. **Prioritize Core Gameplay:** If specialized mechanics (e.g., advanced destruction physics) prove too time-consuming, reduce their scope to a "minimum viable feature" set to ensure the prototype is finished. Defer advanced elements to the Extras list in Sprint 3.
- Obstacle: Proof of Concept Demonstration fails to validate feasibility of key technical risks. **Immediate Re-evaluation:** Implement a 1-2 week engineering halt to re-engineer the failed component or pivot the technical approach. This may require a reduction of feature scope to keep the overall schedule intact.

Sprint 1 (Level Design and Demonstration)

- Obstacle: Level Design Demonstration feedback is overwhelmingly negative, suggesting a major flaw in the design philosophy. **Focused Redesign:** Dedicate the first half of Sprint 2 entirely to a major redesign of the level structure and mechanics based on feedback. Accept that this will compress the time available for the Integration task in Sprint 2.

Sprint 2 (Integration and Documentation)

- Obstacle: Integration reveals significant technical incompatibilities or performance issues between game systems. **Systemic Debugging:** Enforce a strict code freeze on new features. Reallocate all resources to integration and bug-fixing. Simplify or temporarily remove any component deemed too unstable to preserve the overall build's playability.
- Obstacle: Design Document Revision 0 is delayed, leading to a lack of clarity for the final push. **Decouple Documentation:** Prioritize the Integration and stability of the game build. Accept a delay on the documentation revision, aiming to finalize it early in Sprint 3, ensuring it still accurately reflects the final game state.

Sprint 3 (Finalization and Delivery)

- Obstacle: Bug Fixing effort reveals high-priority defects that are difficult to resolve late in the process. **Critical Path Focus:** Categorize bugs by severity. Only fix absolute highest priority bugs that most affect functionality. Lower-priority bugs are to be documented as Known Issues in the final V&V Report to ensure the project meets the Final Demonstration deadline.

(P.7) Requirements process and report

The requirements elicitation process will likely be done using open-ended surveys for large populations and open interviews for smaller, more technical stakeholders, since they offer a greater amount of possible responses compared to open interviews, but still allow for the user to answer in a way that they feel is the most correct, as opposed to forcing arbitrary choices on them. If the stakeholder is a non-human entity, they likely have documentation for which document analysis will be our primary elicitation method.

- Surveys with some open-ended answer options should be used when eliciting requirements from our casual gamer stakeholders. With this stakeholder, it's important to remember that everyone games in a different way, and what might be normal for one person might not even be considered by another. Closed-form questions will not accurately capture the requirements that we need from this stakeholder.
- Open interviews should be used when eliciting requirements from developers. Unlike casual gamers, developers can provide highly technical and design-oriented feedback, often rooted in best practices, prior experience, and knowledge of what makes games successful. Open interviews allow for in-depth conversations where specific points can be expanded on, and where follow-up questions can be asked if developers identify potential risks, implementation challenges, or opportunities for improvement. This approach ensures that critical insights aren't lost in rigid survey structures and instead are captured through detailed discussion.
- Document analysis is the primary requirement elicitation method for Valve, as they are an established marketplace and regulator of content, they have certain standards which they have documented. Valve is generally not represented by just a few individuals so our normal requirements elicitation process does not work, but if after the document analysis we still have questions, asking questions to their support is the way to elicit requirements.
- Document analysis is the primary elicitation method for Government Organizations, as they are

literally required to have their standards in writing. This will include studying regional legislation in places where we plan to make the game available, or other rating standards. In cases where regulation is unclear, formal correspondance or consultations with legal experts or compliance officers may be necessary

References