Hazard Analysis
SFWRENG 4G06 - Capstone Design Project

Team #7, Wardens of the Wild
Felix Hurst
Marcos Hernandez-Rivero
BoWen Liu
Andy Liang

Table 1: Revision History

Date	**Developer(s)**	**Change**
2025-10-09	All	Created Document

Contents

1 Introduction

The primary goal of this Hazard Analysis is to identify, categorize, and evaluate potential risks and flaws in the game system and its development process. This proactive approach ensures that we address potential issues before they translate into significant problems during development or post-launch. By maintaining a high standard of quality assurance, we aim to deliver a stable, secure, and enjoyable product. A hazard is a flaw in the system that could lead to the user not enjoying our game, inhibited usability, or security/privacy risks.This document will proceed by identifying various hazards across technical, design, and operational domains. It will then analyze the severity and likelihood of each hazard, assigning a resultant risk level. Finally, for all non-trivial risks, concrete mitigation and contingency plans will be proposed to ensure project integrity and the quality of the final product.

2 Scope and Purpose of Hazard Analysis

This hazard analysis covers the game software on Windows 10+ (Minimum Spec): input handling (keyboard/mouse; optional gamepad), rendering/audio, terrain & debris, field publication (water/heat/light), organism traversal, and file I/O (saves/settings/logs). It excludes OS/hardware faults, the Steam client, mods, and online services. Potential future features (e.g., microphone input, external weather) are noted but out of scope for R1.

Purpose

- Identify hazards arising from normal play and operation
- Rate severity and likelihood to prioritize mitigations
- Turn mitigations into verifiable requirements/tests (tie to SRS E.3/E.6 constraints/invariants and S.6 acceptance).
- Maintain traceability from hazards → requirements → tests.

Potential losses if hazards occur

- **Human health/safety:** photosensitive seizures (flashing), hearing damage (sudden loud audio), motion/visual discomfort. → Loss: injury or illness.
- **Accessibility/usability:** unreadable UI at 1280×720; non-functional keyboard/mouse paths. → Loss: players can't play/progress.
- **Data integrity:** corrupted/partial saves/settings/logs. → Loss: progress/time loss; inability to resume.

- **Privacy (future features):** unintended microphone capture/storage. → Loss: personal data exposure; reputational harm.
- **Stability/performance:** crashes, hangs, excessive resource use, high latency. → Loss: denial of service; unplayable experience.
- **Legal/compliance:** mis-licensed assets; writing outside user-writable paths. → Loss: policy violations; takedowns.
- **Project/academic outcome:** failure to meet Minimum Spec or acceptance criteria. → Loss: missed milestones; grading/demo impact.

Method (summary)
Use a checklist + what-if sweep and a simple severity/likelihood matrix (Low/Med/High). For each high/medium risk, define a concrete mitigation that becomes a testable requirement in S.6 (e.g., "atomic save writes," "publish $\leq$ 1.0 s," "readable at 1280×720").

3 System Boundaries and Components

1. Debris System
2. Inventory System
3. Player Controller
4. Organic Pathway System

4 Critical Assumptions

This section outlines the foundational assumptions made about the project's development environment, technical stability, and external dependencies. A failure in any of these assumptions must trigger an immediate re-evaluation of the project plan and hazard analysis.

A core assumption is the technical feasibility of the core requirements. We assume the specialized systems, the destructible environment and the slime mold traversal system, can be implemented successfully using the chosen engine and tools within the timeframe established by Sprint 0. Discovering fundamental limitations after that initial sprint will necessitate a critical reduction in the project's overall scope.

We also assume the stability of the development environment. This means the chosen game engine and all associated third-party components (such as physics libraries) will remain stable, compatible, and maintainable throughout the entire development cycle, from Sprint 0 through Sprint 3. We assume there

won't be any major, unscheduled, or breaking changes to the engine or dependencies during this time.

The project relies on the availability of human resources. We assume that all core development team members will be available and committed to the project according to the schedule in P.4 of the SRS document. The successful delivery of key sprint items would be compromised if any critical team member experienced a prolonged, unplanned absence.

We make assumptions regarding user system specifications. We assume the final game will be run on hardware and operating systems that meet the minimum specifications to be defined in the Design Document (Revision 0). We manage performance hazards based on the expectation that end-users possess at least these minimum required specifications.

5 Failure Mode and Effect Analysis

Component	Failure Modes	Effect of Failure	Severity (1-10)	Cause of Failure	Likelihood (1-10)	Recommended Action	Ref #
Game design and implementation	Player is softlocked	Player cannot progress; forced to restart the game	8	Poor puzzle design	2	Simple, linear puzzle design	N/A
				Unexpected game / object behaviour	4	Include the ability to reset a puzzle room	F214
	Game crashes	Play session is interrupted, unsaved progress is lost	10	Fatal bug	1	Thoroughly test each level; introduce auto-save for key checkpoints; include contact information for bug reports in the crash window	F213, NF232
	Not fun to play	Player loses interest, stops playing. Low game review scores	6	Clunky, unfair gameplay; puzzle solutions that do not make sense realistically	2	Playtesting with a variety of stakeholders	N/A
Interfaces	Controller not supported	Player unable to play with the controller of preference	7	Controller device is not recognized	5	Add support for a wide variety of controllers	N/A
	Player does not know how to play the game	Player gets confused and frustrated	7	Unintuitive controls and/or directions	7	Include a tutorial level demonstrating the controls and objectives; include a HUD	N/A
	Not visually appealing	Product does not sell	10	Poor art direction	5	Art direction overseen by one team member	N/A
Security	Program is vulnerable to malware	Player's device is vulnerable to malware	9	Over time, cybercriminals find new software vulnerabilities in all kinds of software	3	Offer security updates post-launch to patch game and Unity vulnerabilities	NF234
	Program causes harm to the user's device	Player's device is damaged	10	Program causes memory leaks, bypasses operating system security, etc.	1	Test extensively on all supported operating systems; design game to run without requiring administrator privileges	NF233
Message	Message is offensive	Non compliance of Steam rules	5	Messages in the game include offensive / derogatory, language	2	Avoid offensive language	N/A

6 Safety and Security Requirements

- **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).

- **Reset Puzzle:** The player has the option to reset a puzzle room to start over, in case they got lost or softlocked. (F214).
- **Stability:** The system shall experience crashes no more frequently than once every two hours on average. (NF231).
- **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).
- **Privileges:** The system shall be able to run without requiring administrator privileges. (NF233).
- **Software Vulnerabilities:** Any Unity security vulnerabilities shall be patched in a timely fashion and distributed in a software update. (NF234).

7 Roadmap

All requirements in regards to validating correct and expected behavior of the destructible environment and organic pathway system should be implemented and met by the proof of concept stage in addition to the basic UI organization as well as making sure any hardware and/or potential internet vulnerability countermeasures are implemented.
Once the core mechanics of the project are solidified, the game levels can be built and tested over top of that and evaluated on usability and adherence to the central theme of the game. Each level should also be built based on the core mechanics of the game

Appendix — Reflection

Team:
3. Before beginning this deliverable, we thought of risks such as:
-The game not being fun, or not being visually appealling
-Controller support
The rest came about by imagining ourselves playing our game and thinking about what could go wrong at any step of the gameplay loop. It also came from thinking about our past experiences playing other games that had bugs or other issues. This led to thoughts of softlocks, crashes, lost save data, and so on.

Felix Hurst

1. After defining what a "hazard" is in the context of our project, it became easy to come up with potential hazards, creating the FMEA table smoothly with a few categories for organization.

2. Because our project is a game, it is difficult to define "hazards" for a system that has no chance of physically harming the user. We resolved this by redefining what a "hazard" is in the context of our system. Now they focus on security risks, usability, and user enjoyment. This allows us to have a better perspective of "what could go wrong" in the development of our system.

3. See team answer above.

4. Two other types of risks in software products include:
   Data privacy: Will the system collect a user's personal information? If so, it should be kept safe, protected from being leaked to malicious third parties.
   Malware vulnerability: Does the system have any extreme weaknesses that could be exploited to enable an attacker to cause harm to the user's device? Care should be taken to keep a program as secure as possible.

BoWen Liu

1. Our team was able to efficiently identify what would be defined as a "hazard" in our project and set out a plan based on our deliverable schedules as to the concrete implement details.

2. In contrast to projects that require internet connection and/or multiple users, our game is fairly closed off as a package with minimum possible network weaknesses, however we were able to identify unique failure points that are unique to our game project such as game breaking bugs and potential third party vulnerabilities with Unity.

3. See team answer above.

4. Risks may include for our case security vulnerabilities in 3rd party software that we use in our case Unity, Another risk could be game breaking bugs

resulting in the player unable to proceed to the next level thereby making the player stuck at a level.

Marcos Hernandez-Rivero

1. Many of the sections were not too hard to write, and I found that after we met up as a group and discussed some things (like defining what we consider to be a hazard), it became a lot easier to write a given section, since this document is fairly straight forward.

2. For us, "Hazards" sounded a bit harsh, and given that we are making a game, we did not really see (at first) how a game could have "hazards" apart from a memory leak affecting the computer, especially since we do not intend to make this an online game, so the user's communication with external parties is VERY limited.

3. See team answer above.

4. One such risk is a memory leak, where a program consumed more and more memory due to not closing off access to the memory, eventually affecting the entire computer's performance if it goes on long enough. Another suck risk would be malware. We plan to release this game on the Steam platform which does verify files for malware and is widely known and trusted, but at the end of the day, we are not the ones hosting the game, and a malicious actor in Steam's system may be able to change our game files and add in malware. These risks are important to consider because they are sometimes invisible, it's almost impossible to tell just from looking at a screen if a memory leak is happening or if a virus is on your computer until it has already acted and caused severe damage, thus we must ensure that we are very careful to not introduce these risks.

Andy Liang

1. Once we reframed "hazard" for a game (beyond physical harm) to include usability, data integrity, stability, and player enjoyment, ideas flowed quickly.

   Breaking the system into components (controller/input, debris/destructible world, organism traversal, UI/save I/O) made the FMEA systematic instead of ad-hoc.

   Turning mitigations into testable requirements was smooth (e.g., autosave at level starts/checkpoints, room reset to avoid softlocks, readable UI at 1280×720, wide controller support, informative crash log). This improved traceability into our SRS/tests and gave us clear acceptance criteria.

2. Quantifying "fun" as a risk was tricky; "not fun" isn't directly testable. We resolved this by proxying it with measurable hazards: softlocks, unclear puzzle affordances, and poor onboarding. That led to concrete mitigations (tutorial level, HUD cues, reset options, playtest checkpoints).

   Assigning severity/likelihood scores without real data caused disagreement. We aligned by anchoring to minimum spec and failure impact (e.g., save corruption $\geq$ high severity, single-room art glitch $\leq$ lower severity) and recording rationales next to each score.

   Scope creep (future features like mic input) kept sneaking in. We explicitly bounded R1 to our offline Windows build and noted future risks separately, which kept the FMEA focused and implementable.

3. See team answer above.

4. Data integrity risk (e.g., corrupted/partial saves/settings): threatens player time/progress and trust; mitigated via atomic/defensive writes and checkpointed autosaves.

   Accessibility/usability risk (e.g., unreadable UI at 1280×720; unsupported controllers): can make the game unplayable for part of our audience; mitigated via readability constraints, input remapping, and controller coverage.

   Stability/performance risk (crashes, hangs, resource spikes): directly blocks play and generates negative sentiment; mitigated with crash logging, performance budgets, and soak tests.

   Legal/compliance risk (licensing, writing outside user-writable paths): can cause takedowns or user hostility; mitigated via asset audits and correct file I/O locations.