CS310 Project Specification: Exploring the World of Game Development in Haskell

5511814 – Nicholas Mason-Apps

October 2025

1 Problem Statement

Functional programming has become of great interest in recent years, with a primary language being Haskell. Haskell is a purely functional language, with strong static typing, first-class and higher-order functions, type inference, concurrency, and lazy evaluation [4]. Many of these features are of interest to programmers, as they all aid in producing more robust, separated, and re-usable code. Additionally for Haskell, code written in it follows a very clear and concise syntax, which allows code to be easily understood by external users.

Video games and game development have also become an increasingly popular medium; it is estimated that 3.32 billion people play video games around the world [6]. Many video games today are developed using tools called game engines, with the most popular one being Unity [5]. Unity uses C# for any of its code writing, and C# is an imperative programming language which includes functional features for developers to make use of [1]. Therefore, many games developed today make use of functional features to improve their workflow and the development of their game. Despite this, few examples exist of feature-rich games being developed in Haskell and taking advantage of the benefits it provides for game development.

2 Impact

From the problem statement, this project aims to develop a 2D game using Haskell and the `apecs` library. This will showcase the abilities of the Haskell language for game development, and gain insights into the benefits, drawbacks, and intricacies of using Haskell for game development. The game that results from the project will also act as reference material for future aspiring developers, helping to fill the gap highlighted.

The game itself will be a 2D roguelike [2]. The game will allow the user to control a player character, explore procedurally generated levels, and combat enemies in a turn-based style in an attempt to proceed to the next level. The levels themselves will be designed such that each level will procedurally generate different configurations of rooms, and each room will have content for the player to explore. This style of game is not unique; many roguelike games exist already with this style of gameplay, and examples games being *Enter the Gungeon* and *Blazing Beaks* following this template can be found in Figure 1. However, since the aim of the project is to showcase the suitability of Haskell for game development, this only serves as a benefit as many games can be used as reference points, and I also have personal experience developing these style of games, pushing the focus of development onto implementation in Haskell.

Figure 1: Screenshots from *Enter the Gungeon* and *Blazing Beaks* respectively to showcase the gameplay of roguelikes.

3 Objectives

Since this project aims to develop a feature-rich game, there will be many different objectives each designed to improve the quality of the game, each with different priorities.

3.1 Must-have Objectives

These objectives are ones which are needed for both the game to function and needed by all additional features of the game. Therefore, these are mandatory to include:

1. Allow the user to control and move a character in a 2D, top-down game world.
2. Use procedural generation to generate each level such that position of rooms is unique, increasing the replayability of the game.
3. Handle collision detection between the player and any game objects within the world.
4. Ensure that the game runs at 60 FPS 99% of the time, at a minimum. This will be monitored using frame rate capturing software, and the Department of Computer Science's computer systems will be used as the base system.

3.2 Should-have Objectives

The following objectives are not mandatory for the game to function, but their inclusion would significantly elevate the game and its playability:

1. Include enemy characters which inhabit the game world and must be fought by the player in order to proceed
2. A turn-based combat system for fighting enemies. At a base implementation the user will be able to select up to three different attacks to use against the enemy.
3. Use procedural generation for the contents of each room, increasing the uniqueness of each play of the game.
4. At the end of each level, implement a special enemy to add an additional layer of challenge and complexity.

3.3 Could-have Objectives

These objectives include features which would make the overall game feel more complete, however are not important to include if time does not allow:

1. Implement an inventory system for the player character, and allow them to find collectible items throughout their play of the game to empower them.

2. Add different types of levels that change as the player progresses, improving the progression of the game.
3. Extend the turn-based system to include an "affinity" system: each attack will have an affinity and enemies will be resistant or weak to specific affinities.
4. Include a reaction-based gameplay system for combat, where if the player presses an input correctly and within a time window, they are able to dodge enemy attacks.
5. Polish the game by adding visual feedback to the player to improve the feel of the game, as well as polish visuals to be more cohesive. An example of the first would be adding visual shake and particles to shooting a projectile.

3.4 Further extensions

These extensions are ones which go beyond the original scope of the game, and either intend to push the direction of the game further, or aim to do something different along with the original vision of the game.

1. Produce a barebones 3D game in Haskell using an SDL binding library, such as `sdl2`. This will act as a proof-of-concept that 3D games, even simple ones, are also feasible in Haskell.
2. Introduce multiple NPC characters that join the player during combat scenarios, and extend the turn-based combat system to allow for multiple characters to attack the enemy, and conversly enemies can dynamically change the characters they are targeting.
3. Develop a system to allow for dynamic, radial-based lighting to be added to the game world.

4 Technical Interest

This project will utilise the Haskell programming language and the `apecs` entity component system (ECS) library [3]. Due to Haskell being a purely functional language with lazy evaluation and allowing for a high level of concurrency, it will allow for a feature-rich game to be developed whilst still being highly performant. Since the `apecs` library is an ECS rather than a rendering library, it is designed to be easily integrated with other rendering libraries, such as `gloss` [7], complementing the additional objectives of the project well. Additionally, the project would be of interest to anyone eager to learning more Haskell, game development, or both, and the clear and concise nature of code written in Haskell complements this. For version control and history, `git` will be used along with a remote repository served from `Github`. This will allow for secure backups to be created during the continuous development of the game.

All assets for the game will be acquired from public repositories, such as `https://kenney.nl/`, which are designed for developers to use as temporary replacements of assets. For the game, all sprites will be 2D assets, and any audio samples will be gathered from the same repositories.

5 Methodology

The development of a video game typically revolves around incrementally adding features and testing developmental builds of the game as new features are added. Therefore, an extreme programming-based methodology is most suitable, as extreme programming prioritises continuous, but smaller releases which matches the incremental feature development of games. It will differ from traditional XP as it will lack pair programming and a customer always being present. The lack of pair programming and a customer being present always is due to the project being primarily a solo endeavor. However, that is not to say customers will not be used at all. For testing of the game, unit tests will be written where appropriate, but due to the nature of games the majority of tests will come from playtesting. Therefore, playtests will be conducted by gathering a small group of people to interact with the game and provide feedback through a form, which will in-turn help inform future iterations.

6 Timetable

Week	**Events**
T1 W1-2	Writing of and Submission of Project Specification
T1 W2-3	Begin learning `apecs` through literature and online tutorials and adapting to the project. Ensure player movement is implemented fully.
T1 W4-5	Learn about `apecs-physics` for collision detection and implement a rigorous collision detection system.
T1 W5-6	Research how existing solutions for procedural generation are implemented in Haskell, and utilise and adapt for developing the procedural generation compoment of the game's layouts.
T1 W7	Buffer for the CS325 Compiler Design coursework.
T1 W8-9	Continue implementation of procedural generation. Gather feedback from playtests.
T1 W10	Buffer for the CS349 Principles of Programming Languages coursework.
T1 W11	Start implementing the turn-based combat mechanic.
Winter Holidays	Writing of the progress report.
T2 W1	Gather feedback on progress report and improve the report. Continue development of the combat mechanic.
T2 W2	Submission of Progress Report.
T2 W2-3	Continue development of the combat mechanic. Gather feedback through a playtest.
T2 W3-4	Expand on procedural generation to extend to the procedural generation of room elements.
T2 W5	Buffer for Term 2 Courseworks.
T2 W6	Buffer for Societal commitment: running the Real Ale festival for the Real Ale society.
T2 W7	Finalise room-specific procedural generation and prepare for Evaluation day.
. T2 W7-8	Begin writing the Final Report Plan document.
T2 W9	Finalisation of and submission of the Final Report Plan document. After submission begin writing the Final Report and Project Management Report
T2 W9-10	Continued writing of the Final report and Project Management Report.
Easter Holidays	Module Revision and continued writing of the Final report and Project Management Report. Begin preparation of the Project Viva.
T3	Continued module revision and continued preparation for the Project Viva.

The timetable shown is a rough outline of the events which will occur throughout the academic year and how they relate to the development of the project. The timetable is deliberately lenient; multiple buffers have been added to accommodate for other modules and commitments outside of the course, as well as being lenient in the time frames for certain events. Despite this, the timetable is designed to produce, at minimum, a prototype game which incorporates all of the must-have objectives as well as some of the should-have objectives. Therefore, in the worst case scenario, a minimum game will be produced with more than sufficient time being given to the other deliverables for the project. Thus, due to the leniency of the timetable, it is quite possible that work will be ahead of schedule, therefore allowing for more features to be implemented.

Throughout the development of the project, documentation and notes will be written alongside the development in order to aid the writing of the Progress Report and the Final Report, along with recurring meetings with the Supervisor to gather input and feedback on the progress of the project.

7 Risks

Below are the following risks associated with the project:

Risk	**Mitigation**
Personal Computer may break	Utilise `Github` to store the source code and resources such that they can be accessed from other computers.
Unexpected delays from illness	Due to the current timetable being lenient on time, any delays caused will be able to be recovered from.
Asset repository is unsuitable	Replacement repositories will be found, such as `https://opengameart.org/`.
Asset repository is temporarily unavailable	Necessary assets will be saved locally.
`apecs` library is not suitable for the scope of the project	An alternative of just using `gloss` for both rendering and game programming using a loop-based system will be used.

8 Legal, Social, Ethical, and Professional Issues and Considerations

Copyright laws is a big concern when using assets from public domains. For assets uploaded to `https://kenney.nl/`, they all come under the Creative Commons 0 (CC0) license, which places no restrictions on how the assets can be used. Therefore, using them in this game will not raise any issue with copyright.

For playtests, the information will be collected thorugh a form, with the playtesters being friends and colleagues. As a result, the Department of Comptuer Science considers this a low-risk ethical issue, and therefore is not of concern.

9 References

[1] Anon. C sharp (programming language), 2025. [online] Avaiable from `https://en.wikipedia.org/wiki/C_Sharp_(programming_language)#Functional_programming`.

[2] Anon. Roguelikes., 2025. [online] Avaiable from `https://en.wikipedia.org/wiki/Roguelike#Gameplay_and_design`.

[3] J. Carpay. *Apecs: A Type-Driven Entity-Component-System Framework.* 2018.

[4] Inc. Haskell.org. Haskell, 2025. [online] Avaiable from `https://www.haskell.org/`.

[5] Video Game Insights. The big game engine report of 2025, 2025. [online] Avaiable from `https://app.sensortower.com/vgi/assets/reports/The_Big_Game_Engines_Report_of_2025.pdf`.

[6] Konvoy. How many people play video games in 2025? (and most popular genres), 2025. [online] Avaiable from `https://www.konvoy.vc/blogs/how-many-people-play-video-games`.

[7] B. Lippmeier. gloss: Painless 2d vector graphics, animations and simulations., 2025. [online] Avaiable from `https://hackage.haskell.org/package/gloss`.