HaskellQuest: a game for teaching functional programming in Haskell

Eva Bogomil

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

Functional programming is notably different from procedural or object-oriented programming paradigms, and many students and developers find it challenging to understand its main concepts at first. While traditional learning methods remain a readily-available option, the primary goal of this project was to make the learning process easier and more entertaining by developing an educational game for teaching functional programming, using Haskell as an example.

HaskellQuest is a classic-inspired 2D RPG that gamifies the process of learning the functional programming language Haskell and makes it a highly engaging, accessible and rewarding experience. It is primarily aimed at undergraduate Computer Science students who have to undertake a functional programming course as part of their degree curriculum, as well as anyone who would like to learn or improve their Haskell skills.

The game is set in a fantasy world and follows a pilgrim who is recovering the long-lost knowledge of the ancient art of functional programming. The player travels through different locations of increasing difficulty, exploring the world and collecting fragments of Haskell knowledge to fight off enemies. The hero's journey continues until the world's biggest mystery is finally solved.

The project has succeeded in the implementation of all the core mechanics necessary for a functional game. Subsequent user tests have yielded overwhelmingly positive reviews, validating the concept behind the game and justifying the project's main purpose.

The first release of HaskellQuest is currently publicly available on GitHub. Should further development continue, the game can easily be scaled-up to accommodate further content, and target to fill the currently existing gap in publicly available production-grade interactive resources for Haskell education.

Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee.
Ethics application numbers: 7094, 7256
Dates when approval was obtained: 2022-11-29, 2023-01-31

The participants' information sheets and a consent forms are included in the appendix.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Eva Bogomil*)

Acknowledgements

I would like to express my gratitude to my project mentor Professor Don Sannella for remaining invariably supportive of my efforts.

A special thank you to the JDoodle team for their collaboration and making the open user testing more accessible, as well as Jacob Gibbins, Adrian Hill and everyone else for providing me with invaluable advice at the start of the project.

Table of Contents

Chapter 1

Introduction

HaskellQuest is an educational game that gamifies the process of learning the functional programming language Haskell and makes it a highly engaging, accessible and rewarding experience. It is primarily aimed at undergraduate Computer Science students at the University of Edinburgh (”UoE”) who have to undertake the INF1A “Informatics 1 - Introduction to Computation” course as part of their compulsory first-year curriculum, as well as anyone else who would like to learn or improve their skills with Haskell. This paper explores different approaches to gamifying programming education and its potential positive impact on students’ engagement with Haskell.

The gamification of programming is not inherently a new idea. Various existing examples of it can be found on platforms such as CodinGames [3] for many popular programming languages. However, HaskellQuest mechanics goes above and beyond its peers in terms of flexibility, range and variety of usage. Instead of restrictive methods of education, HaskellQuest users are encouraged to explore Haskell concepts in a way that stimulates both creativity and curiosity, which facilitates a truly unique and engaging learning experience. While this game is developed mainly based on the information gathered from students at the University of Edinburgh, it is not institution-specific and can be used as a practical support tool by anyone who is learning Haskell.

1.1 Motivation

As someone who initially struggled to understand the principles of functional programming, I am very passionate about this project. My motivation for HaskellQuest was to build a valuable and accessible tool that not only educates, but motivates its users, and would fill the currently existing gap in publicly available production-grade interactive resources for Haskell education.

Haskell as a language does have some interesting applications that include Barclays Capital’s Quantitative Analytics group where Haskell was used to develop a domain-specific functional language FPF to specify exotic equity derivatives as mathematical functions. Companies such as Facebook, Bank of America, BAE Systems and Klarna also use Haskell [12]. And while it is not one of the top most sought-for languages

in the industry, I believe it can still serve as a valuable introduction to the functional programming paradigms used in other programming-related domains. For example, MapReduce - a programming paradigm used for handling big data - was inspired by ”map” and ”reduce” functions from functional programming. Haskell also teaches higher-order functions and lambda functions, which are a common feature of many other popular languages such as Java, Python, Scala, Kotlin and JavaScript.

1.2 Project goals

At the start of the project, the following three main goals were identified.

1. Throughout its development cycle, the project should follow industry standards and best practises discussed in the Approach Chapter.
2. HaskellQuest should cover at least 5 Haskell concepts discussed in the Concept Chapter.
3. HaskellQuest should provide an engaging and unique user experience that motivates players discussed in the Design Chapter.

1.3 Project structure

- Chapter 2. Background discusses functional programming education and gamification of teaching programming languages.
- Chapter 3. Approach describes the project development methodology in accordance with the industry-standard game production cycle. It also covers the game target audience study and decisions behind the chosen game technology.
- Chapter 4. Concept is dedicated to the main game concepts and mechanics, as well as description and evaluation of players' usage of Haskell.
- Chapter 5. Design explains all major creative processes and decisions made regarding user interface, user experience and game plot.
- Chapter 6. Implementation includes key technical details behind the core game mechanics.
- Chapter 7. Evaluation summarises the results of the user testing, future improvements based on the user feedback and current game limitations.
- Chapter 8. Conclusion describes the planned future development and main achievements of HaskellQuest.

Chapter 2

Background

In the first half the chapter discusses the importance and relevance of functional programming education, specifically Haskell. The second half discusses the advantages of the gamification approach in education by exploring available existing interactive resources for learning programming languages.

2.1 Functional programming education

The initial attempt at teaching functional programming languages dates back to 1980s at MIT. ”The structure and interpretation of computer programs” was especially designed as a compulsory entry-level course to focus on the ”techniques used to control the intellectual complexity of large software systems” rather than on syntax, algorithms or mathematical analysis. A ”dialect of the programming language Lisp” was used for the course and the course organisers claimed that they did not have to teach it to students in a formal way but rather expected students to easily pick it up.[43]

In 2001 Dijkstra mentioned in his letter to the Budget Council of The University of Texas that the ”practical reason for preferring functional programming in a freshman course is that most students already have a certain familiarity with imperative programming” and that teaching Haskell to first-year students would introduce novelty compared to any school programming experience and thus boost interest and engagement. Additional points were made about the improved perception of functional programs as mathematical objects and the advantages of lazy evaluation concepts. [51]

M. M. T. Chakravarty and G. Keller talk about their preference for functional languages as the latter support the three principles the authors believe students should get introduced to first: elementary techniques of programming (the practical aspect), essential concepts of computing (the theoretical aspect), and the development of analytical thinking and problem-solving skills (the methodological aspect). [47]

2.1.1 Why Haskell

Haskell is a modern, standard, non-strict, purely-functional programming language. [15]

According to HaskellWiki, Haskell is currently being taught at a number of UK universities, as well as overseas. Among those, the University of Edinburgh, the University of Nottingham, the University of Oklahoma and several others teach Haskell as the first language in their Computer Science curriculum. [56]

M. M. T. Chakravarty and G. Keller believe that first year courses should focus on teaching foundational knowledge rather than individual languages. They state that Haskell syntax helps to ”concentrate on general programming concepts” and allows the introduction of complex structures, such as expression trees and memory trees, to beginners. ”Rigorous static type discipline” also assists with emphasising types as another fundamental concept. At the same time, ”topics specific to functional programming”, such as list comprehensions, currying, higher-order functions and lambda expressions, can initially be omitted. [47]

2.1.2 Teaching Haskell to first-year students at the University of Edinburgh

The University of Edinburgh does not require previous experience in Computer Science to undertake an undergraduate degree in Computer Science. In fact, at the time of writing, UCAS does not list A Level Computer Science as a prerequisite for any undergraduate Computer Science course at any UK university. [32]. Instead, entrance requirements often lay emphasis on Mathematics.[29] Such an approach is advocated for by M. M. T. Chakravarty and G. Keller who believe that ”a modern functional language makes it easier to appeal to the mathematical background of these students and serves to neutralise some of the advantage that their peers have due to prior programming experience.”[47]. The belief, however, somewhat contradicts the original idea by MIT course authors (later expanded by Dijkstra) that teaching functional programming in the first year should be based on the expectation that first-year students already have some experience in either programming or computer hardware systems.[51] [43]

In 2022, BCS, The Chartered Institute for IT (”BCS”), recorded 158,340 Computer Science applicants in the UK [39]. That same year, only 15,210 students took Computer Science at A Level in England [40]. According to the ”Computing Qualifications in the UK” report [38], only AQA included functional programming in their ”Computer Science and IT” A Level syllabus. This was addressed by the INF1A course organisers at UoE. D. Sannella, M. Fourman, H. Peng and P. Wadler in their ”Introduction to Computation” book, written specifically for the course, acknowledge that ”Students who have never programmed before need to work a little harder. But students who have programming experience often have just as much difficulty understanding that Haskell is not just Python with different notation.” [68].

Having done a thorough investigation of the students’ feedback [25] on INF1A course between 2017-2021, I have identified the following common patterns.

Firstly, the majority of people find tutorials very useful for understanding the functional programming part of the course, meaning that students seem to enjoy practical exercises in general. However, students also mentioned that ”It would be helpful to have time with similar problems that are not assessed so we have freedom in our questions.” Secondly, there was a large number of responses stating that previous programming experience would have been useful for the course, as well as those advising future generations to pick up Haskell before starting the course. Some students felt particularly negative about this issue: ”You must have computer knowledge before starting this course. It is not advertised accurately.” Since INF1A is the very first programming course of the UoE CS degree, for some students it effectively becomes an introduction to the whole area. Thirdly, some students complained that the course was too hard for complete beginners and that they felt discouraged by how poorly they were performing compared to students who had more programming experience.

2.2 Gamification in education

While, as mentioned above, there are clear advantages in teaching functional programming in general, and Haskell in particular, it can also prove challenging for students for reasons discussed below.

2.2.1 Why Haskell may be difficult to pick up

Functional programming is notably different from procedural or object-oriented programming paradigms. M.Lipovača, the author of ”Learn You Haskell for Great Good!”, states in the first chapter, ”I failed to learn Haskell approximately 2 times before finally grasping it because it all just seemed too weird to me and I didn’t get it.” [61]. P.Wadler mentions in his paper that programmers who are familiar with imperative programming languages often find functional programming odd at first. He also states that for popular problems, imperative solutions are more likely to exist and be readily available than functional ones, even if the latter would be more elegant [76]. This view is supported by this project’s own research Questionnaire.

Some authors believe that the difficulties can at least partially be attributed to the need for better Haskell teaching resources, especially for the medium level of expertise, since a lot of those currently available are made for either complete beginners or expert-level programmers [69]. I tend to agree that one of the biggest challenges for learning Haskell is the lack of interactive resources. For the UoE Inf1A courses the main resources are reading materials, lectures and tutorials. Among those the latter are being more practical, but are limited to a few sessions per week. The lack of outside resources could be explained by languages popularity which is dictated predominantly by industry needs [74]. Some of the largest platforms on the internet, such as codecademy [2], do not even offer any courses on functional languages. LeetCode supports only Erlang, Scala, Elixir [19] despite numerous requests from users [36] to cover Haskell. HackerRank[10] supports Haskell as well as a broader ”Functional Programming” course[11].

2.2.2 Previous work

As was mentioned before, the gamification of programming is not inherently a new idea. There have been many advancements in introducing games to programming education. Among some of the more interesting examples is the article by Almeida et al. [45] that describes a successful redesigning of an introductory functional programming laboratory course at the University of Minho. During the course the students were asked to ”build a complete game from scratch in their first encounter with a programming language”. The assignment was done in Gloss (an FRP-inspired Haskell library). Authors conclude that ”the restructuring kept students motivated to work actively on their project” and consequently observed an increase in student approval over the years.

Dicheva et al. [50] arrive at a similar conclusion in their paper, in which they conduct a thorough analysis of the gamification approach in education. They argue that compared to traditional means of education, games reinforce skills such as problem-solving, collaboration and communication. Games are described as highly motivational by inducing the joy of gameplay and a thrill from the possibility to win. Games are also a medium that can utilise various mechanisms to encourage people to engage with them.

There are other examples of similar attempts at educational games. Many can be found on CodinGames[3], an online resource that provides ”challenge-based training platform for programmers” and has a wide range of games adapted for many programming languages, including Haskell. The common limitation, however, is that a lot of these games present users with only one challenge for each designated topic such as specific algorithms, optimisation or concepts. Most are also designed primarily for practice and competition rather than teaching.

Indeed, ”gamification”, as defined by Deterding et al. [49], is “the use of game design elements in non-game contexts”. However, unlike the aforementioned examples, for this project I, in some sense, aimed to go beyond this standard approach by building a fully functional game around educational concepts, instead of merely ”gamifying” selected aspects of the course. Such an approach is referred to as ”Serious games” by S. Dreimane[55] and is described as being ”more effective in achieving goals, which might include developing cognitive abilities and knowledge”. Examples of such games for functional programming include The Soccer-Fun project and ScalaQuest. Soccer-Fun was developed for an introductory course in functional programming at the Radboud University Nijmegen [44]. Similarly to INF1A, its authors noticed that ”Every year... there is a group of students who have a hard time understanding functional programming”. In Soccer-Fun, players ”program the brains of football players in the functional language” Clean. The project was deemed successful at improving student involvement and motivation. Similarly, ScalaQuest was an online game aimed at Scala education. It fell into the same category of fantasy RPGs as HaskellQuest, with the core difference being the gameplay - ScalaQuest taught Scala predominantly through quizzes and challenges. Unfortunately, the project was shut down on May 8, 2019[24].

Chapter 3

Approach

3.1 Industry development cycle

As mentioned earlier, I decided to adopt an industry-standard game production cycle. At the beginning of the project, I participated in an intogames[16] online conference, where I had an opportunity to discuss work routines and cycles with both industry and indie game developers. This, combined with additional research gave me great insights into the common development stages and the typical activities that take part in each of them.

Since my project is a one-person project, I had to take into account the time constraints and adapt the typical development process to fit my goals. I have omitted the pre-launch stage that is described in ”The 7 Stages of Game Development” [65] by combining it with the testing. I also did not include the post-launch stage due to its falling outside the time constraints. However, potential post-launch activities are covered in the Retrospective and further development section.

As a result, I separated my work project into the following stages:

1. Planning
2. Pre-production
3. Production
4. Testing
5. Launch

The planning stage began during the summer of 2022 and lasted up to week three of the first semester. This stage included detailed planning of the project time frames and milestones, determining the game target audience, creating the general game idea and its essential features, analysing the targeted platforms and identifying the necessary tools for development. I utilised tools like Gantt charts [28] for planning and a Trello board[31] for tracking the progress.

The pre-production stage lasted around five weeks and included activities such as

finalising the game plot and interface, structuring the gameplay and familiarising myself with the game engine and other decisions. Following industry standards, I produced a Game Design Document (”GDD”) (included in Appendix D Haskell Quest Game Design Document) with a detailed description of game mechanics, game flow, player goals, level descriptions, characters, game map and in-game items. This document was later used as guidance for the production stage. It also played a crucial role in maintaining the game's integrity and consistency, and, most importantly, it allowed the identification of design flaws and early introduction of major changes. This stage also coincided with learning the necessary skills for Unity game engine which was the engine selected for the project, developing the user interface prototype, as well as completing the background research discussed in the Background section.

Production was the longest of all stages and lasted from the fifth week of the first semester to the second half of the second semester. My goal was to incorporate Agile and DevOps practices to which I have been exposed to during my time in industry, but adapt them to my needs. I have separated this stage into 2-week sprints with a milestone at the end of each sprint to reflect on the achievements and identify any blockages. Each sprint was dedicated to a particular part of the game. While it was not always the case that I followed this 2-week structure as sometimes it was difficult to predict how long certain tasks would take, it nevertheless provided vital structure to the process.

A detailed GDD from the pre-production stage helped to identify the key mechanics of the game that needed to be prioritised over other ideas, thus the first major milestone for the production stage was the development of a ”vertical slice” [42]. A ”vertical slice” helps developers evaluate the initial game flow and user experience (”UX”) before adding the rest of the game features. It included implementing the simplest version of each major game mechanic and integrating them into a small prototype. For HaskellQuest, such a ”vertical slice” was finished in January and featured some basic game scenes including a menu, one territory and one playable level. Afterwards, I integrated Haskell (see Haskell integration) and improved the battle algorithm, thus completing the Minimum Viable Product (”MVP”) described in Appendix C Minimum Viable Product Criteria. This was followed by the implementation of additional game mechanics described in Design that concluded the development of the game's currently released version.

The last cycle of this stage coincided with the first testing activity. Originally, the plan included two testing phases: Alpha and Beta. Alpha was supposed to focus on finding and eliminating bugs, generating additional game content and improving both the gameplay and user experience. Beta was intended to serve as the main game evaluation activity. However, due to the time constraints, both testing phases had to eventually be merged into one, resulting in a Beta version of the game that was used for all the enumerated testing activities. The outcome of this stage is covered in detail in the Evaluation chapter.

The final stage of development was launch. As of 26th March 2023, a complete and playable version of HaskellQuest is available on GitHub for Windows: `https://github.com/somethingololo/HaskellQuest-for-Windows` and macOS: `https://github.com/somethingololo/HaskellQuest-for-MacOS`.

3.2 Game engine

One of the core decisions was the choice of the game engine. Game engines are responsible for such aspects as game physics, rendering, development tools etc. While there are many available game engines, I was specifically choosing between the following: Unity [33], Unreal [35], Godot [9], RPG Maker [23] and GameMaker Studio [7]. My criteria were that it must support 2D rendering, have a suitable learning curve, be relatively inexpensive, be applicable to the industry and allow me to gain transferable skills. At this stage, I had a brief game concept which was a 2D Role Playing Game (RPG) with turn-based battle mechanics. Intuitively, RPG Maker would have been an appropriate first choice, however, it is limited exclusively to RPG mechanics which would not allow much flexibility if I wanted to add other mechanics to the game. Furthermore, RPG Maker had a smaller developer community in comparison to other engines and a high initial cost. Unreal engine is a well-established industry tool used for popular games such as ”Fortnite”, ”Sea Of Thieves”, ”Final Fantasy 7Remake” and ”Ark: Survival Evolved” [54]. However, even though it is possible to make 2D games in Unreal, its prime application is 3D. At this point, my choice was narrowed down to Unity, Godot and GameMaker Studio, of which I chose Unity for the following reasons. Firstly, it provided a large ecosystem that, apart from the engine itself, contained Unity Asset Store, Unity Play and the Unity Learn platform with many courses and solutions. It is also worth mentioning the large community of developers and indie creators providing tutorials and answers to many problems. According to multiple sources, more than 50% of games are made with Unity[66] [71] including titles like ”Genshin Impact”, ”Ori and the Will of the Wisps” and ”Among Us” [41]. This made Unity the best choice for my project.

Alternative options included writing the game from scratch in Haskell [75], similar to the ”Unity Tutorial Project in Haskell with Apecs and SDL2” [58] or ”HSRogue: A roguelike in Haskell” [70] since the main HaskellQuest concept was to allow players to write Haskell code in-game. I decided not to use this approach because I wanted to use the more advanced and readily available features of established game engines. Another idea I discarded for the same reason was using CodinGame Game Engine (sdk) by CodinGame, which at the start of HaskellQuest development also lacked detailed documentation and resources.

3.3 Questionnaire

To better design HaskellQuest for its target audience, I have conducted a study (see Appendix A Questionnaire) on the experience of the University of Edinburgh students studying Haskell. The aim was to understand students' behaviour when they start learning new programming languages. Overall, out of the 46 participants, 36 (78%) had no experience with functional programming before starting their undergraduate degree. Out of the 10 (22%) who did have previous experience with functional programming, half mentioned Haskell as their first functional language. 30 participants (65%) strongly agreed that Haskell helped them understand other programming paradigms and logic better. 19 (41%) mentioned using Haskell in their further careers and studies. This

covered cases from using Haskell for other courses to internships, personal projects and competitions. Some participants stated that Haskell helped them with learning Scala, Agda and using functional concepts in Python and Java.

Participants ranked Haskell at 4.83 on the difficulty scale from 1 to 10 (1 - very easy, 10 - very hard). On the multiple choice question on the preferred learning techniques (see Appendix A Questionnaire), “Self-practise and experiments” was ranked as the preferred method of studying new programming languages (39% - 18 participants ranked as the top), followed by ”Video materials” (13 - 28%). The least preferred method was ”Quizzes” with 35% of students (16) placing it in the lowest tier. Overall, there was a clear preference towards more active and visual activities with “Self-practise and experiments”, ”Video materials”, ”Courses with guided instructions” and ”Interactive platforms” all placed above more passive categories such as ”Blogs”, ”Lectures”, ”Books” and ”Quizzes”. This further influenced the choice of in-game activities towards a classic-like RPG (see Concept).

Participants were also asked about which Haskell topics they found the most difficult. The most common replies were monads, recursions, functions and thinking in a functional programming way. The latter was addressed by designing the game around the concept of functions (see Design and Concept), while the topics of functions and recursions were included in the current version of the game.

Chapter 4

Concept

4.1 Game main concept

HaskellQuest is a 2D RPG set in a fantasy world. The player takes on the role of a protagonist recovering long-lost knowledge of the ancient art of functional programming. The player will have to travel through different locations and explore the world to find pieces of knowledge. Each game location is dedicated to a specific Haskell topic. Once the player feels comfortable with their understanding, they can proceed to complete game levels of increasing difficulty where they have to apply their knowledge to progress further.

The game is based on three main game mechanics: exploration, coding and repetition. Players have to explore the world by interacting with in-game objects and non-playable characters (NPCs). These interactions provide the player with both Haskell knowledge and background story. Once the player feels comfortable with their level of Haskell, they can proceed to the next territory by following the main story quest. To unlock a new territory the player has to complete multiple battle levels by coding their actions in Haskell. By completing each battle level the player starts a new game loop with new story content, different Haskell topics and unexplored territories as shown in Figure 4.1.

4.2 Game mechanics

The game includes multiple core mechanics for different scenes that have varying purposes. The main story is presented in the story mode shown in Figure 4.2. The story mode introduces the player to the world setting and their role in the game. This mode contains dialogue presented against a static background. The dialogue can be progressed with a click of the mouse. It is heavily inspired by visual novels.

Story mode is followed by open-world mechanics. The player is presented with a territory that they can navigate freely. The territory contains various interaction points such as in-game objects and NPCs. The player sees a dialogue box when interacting with them. There are two types of interactive objects to be found: ”main story” and ”general”. ”Main story” interactive objects navigate the player through the game by

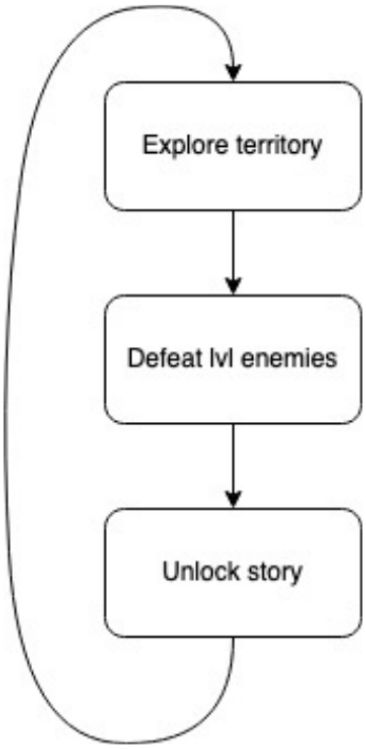


Figure 4.1: Core gameplay loop

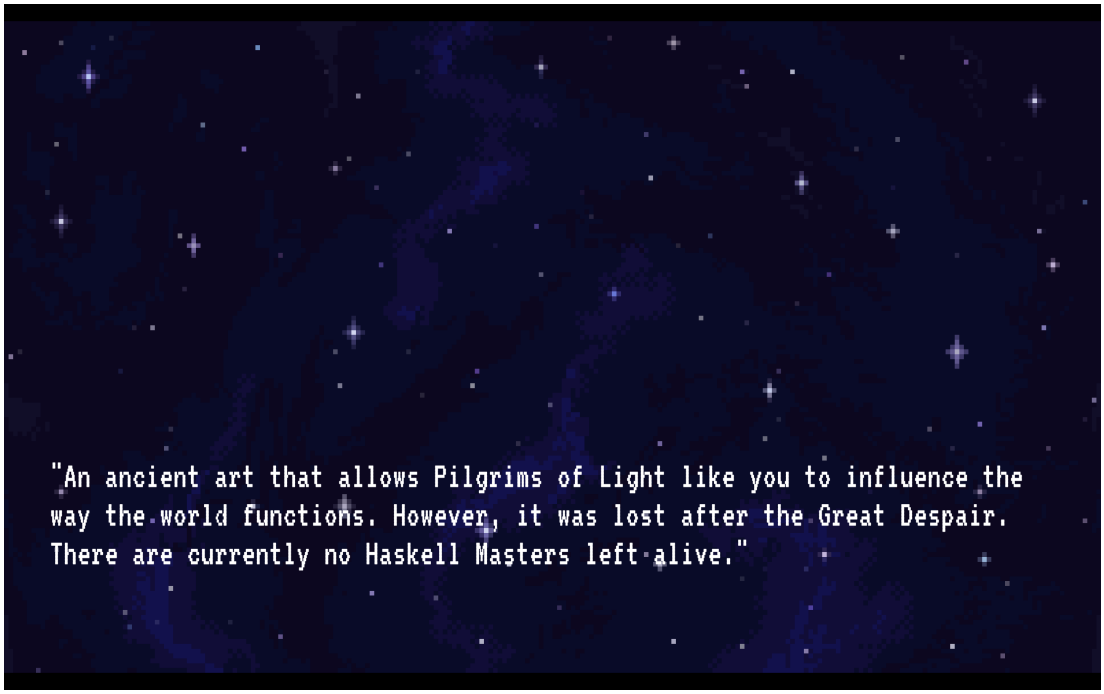


Figure 4.2: Story mode example of introducing player to their role

providing tips on how they can use Haskell knowledge in battle and how to progress to the next territory. The player has to interact with the ”main story” objects in order to understand the game. Interactive objects of the ”general” category provide additional Haskell knowledge and practical examples, as well as character dialogues. Overall, the purpose of the open-world mechanics was to combine the excitement of the unknown and stimulate players' natural curiosity to explore and learn more Haskell tips. The players are rewarded for exploration with interesting and unique tips to complete battle levels. However, in order to encourage free exploration, rather than make it a burden, the player can decide how much they would like to explore. This would depend on the player's experience with Haskell, time available and personal preferences. More experienced players can quickly proceed to the battle levels by playing just the main story. This allows the game to cater to players with different levels of knowledge.

As the player progresses through the main story, they must proceed to the battle mode before navigating to the new territory. The battle mode can be accessed at any stage of the game, however, the main story helps to guide the player to the correct location. The battle mode is a turn-based strategy mechanics. Every enemy and player on the battlefield is referred to as a ”unit”. Units take turns performing an action. Enemies' actions are attacks directed towards the player. The player in their turn needs to write Haskell code that describes their intended action in the built-in text editor. In order for the actions to be executed, the code should compile and be of the correct form described in more detail in Writing Haskell, otherwise the game displays an appropriate error. If the player defeats all the enemies, the game progresses to the next level of increasing difficulty, otherwise the player is returned to the previous territory. The code written by players is preserved in the editor throughout the same level but resets to default as the player progresses to the next one. This is done with the intention of encouraging players to try different strategies for different levels and memorise the syntax.

4.3 Writing Haskell

My goal for the game was to encourage creativity in players when approaching battle levels. As with many complex programming problems, battle levels are not limited to one correct solution. Players write Haskell code in an in-built text editor in a similar way to writing a regular Haskell source file. To execute an action, they need to specify the function in the console just like calling functions from GHCi. This setup allows players to obtain transferable skills in a familiar environment. There is only one restriction - the executing function should have a return type of list of type Turn. Turn is one of the predefined types in HaskellQuest. Type Turn describes which playable character performs which action on whom.

```
type Turn = ((Char, Int), Action, (Char, Int))
```

Each character and each enemy have a unique index of type (Char, Int) that can be found next to their names.

For example the simplest attack could look like this:

```
attack :: [Turn]
```

```
attack = [(('p', 0), SwordSlash, ('e', 0))]
```

Other pre-defined types in HaskellQuest include:

1. Player
2. Enemy
3. Action
4. Units as variables

Definitions of Player, Enemy, Action and Turn appear in the text editor during the entire duration of the level. Action contains different actions a player can perform during the level. For example, SwordSlash would allow the Player to deal damage and Heal would restore health. Some actions are only available during certain levels or characters. For example, RestoreMana action can only be used from the second game level. Player and Enemy are types describing different attributes such as name, index, health, damage and mana. Each unit can be passed to a function as a variable of the corresponding type. For example:

```
attackGuard :: Player -> Enemy -> [Turn]
attackGuard p e
    | hp p > 5 && mana p == 10 = [(index p, SwordSlash, eindex e)]
    | hp p > 5 && mana p < 10 = [(index p, RestoreMana, index p)]
    | otherwise = [(index p, Heal, index p)]
```

To execute this function on an enemy unit ”shadowPuddle” a player ”playerExample” would have to use

```
attackGuard playerExample shadowPuddle
```

in the console. It is also worth mentioning that the player variable would be the name provided by the user at the start of the game, prefixed with the ”player” string literal, for example, ”playerAngryPotato123”.

HaskellQuest battle system allows any Haskell syntax and function complexity. It allows players to use two distinct approaches: static and dynamic attacks. Static attacks are more straightforward: players write a function taking into account the state of the battle only before the function execution like in the example above. It could be the case that the function describes multiple actions, some of which become inapplicable during its execution - for example, if the attacked enemy is already defeated. A more complex dynamic function would allow players to track the state of the units. Dynamic functions are not compulsory but can become a rather powerful tool for players who are more comfortable with Haskell.

```
attackDynamic :: Player -> Enemy -> [Turn]
attackDynamic p e
    | ehp e > 0  = [(index p, SwordSlash, eindex e)]  ++
      attackDynamic p (enemyHealth e p)
    | otherwise = []
```

```
enemyHealth :: Enemy -> Player -> Enemy
enemyHealth e p = Enemy (ename e) (eindex e) (edamage e) ( ehp e - damage p)
```

Once the player is happy with their function and console input, they can proceed to compilation and execution using the ”Compile action” button. Any errors with network, compilation or code format are shown in the dialogue window. If a compilation error occurs, the action won't be executed and the player will have a chance to correct their code. This is done in order to encourage players to experiment with code and reduce player frustration. However, if the action compiles, but was wrong for a different reason, the player will lose their turn and see an error message pointing to where they made a mistake. For example, if the return type of the function was not [Turn], the dialogue window would notify the user that ”Your function output should be in the format [((Char, Int), Action, (Char, Int))]” and the game will proceed with the enemies' turn.

Alternatively, players always have a chance to run away from the battle level and return to the previous territory. If they do so during the battle, the battle progress will be lost. After completing any level, the player is rewarded with an upgrade to their level, damage, health or another attribute. These upgrades will be saved even if the player loses or decides to run away during the next battle level.

The current version of the game supports the following Haskell concepts (tested):

- Types including custom types
- Arithmetic and Boolean operations
- Tuples
- Functions
- If-then-else statements
- Lists including list comprehensions
- Guards
- Recursion

4.4 Level difficulty evaluation

Battle levels are a form of checkpoints for users to evaluate their level of understanding. They are designed with increasing difficulty to challenge the players. The current version of HaskellQuest contains three such battle levels. To ensure dynamic gameplay I evaluated the difficulty of these levels in terms of a minimum number of actions a player has to perform in order to complete each level 4.3. The complete data for this graph can be found in 4.1.

The graph 4.3 shows the total number of attacks a player needs to perform to win, the minimum number of actions a player needs to execute including RestoreMana (assuming the player performs them in one turn without taking any damage) and the total number of enemy attacks for the player to lose. With each battle level the number of actions a player needs to perform increases, with Level 3 being the hardest by design.

Game level	Lvl 1	Lvl 2	Lvl3
Player			
Health	15	18	21
Damage	5	7	9
Mana	10	10	10
ManaPerAttack	3	3	3
RestoreMana	5	5	5
Heal	5	5	5
Enemy 1			
Health	10	10	70
Damage	3	3	10
Enemy 2			
Health	NA	15	NA
Damage	NA	7	NA
Total			
Total enemy health	10	25	70
Total enemy damage	3	10	10
Player attacks to win	2	4	8
Mana needed	6	12	24
How many times to RestoreMana	0	1	5
Min player actions to win (excluding Heal)	2	5	13
Enemy actions to lose	5	2	3

Table 4.1: Player and enemy attributes analysis through the game levels

Level 1, on the other hand, is designed to be introductory and thus the number of player actions is much less than the enemies'. No RestoreMana action is needed on this level. This allows the player to get confident with the game and its interface before proceeding to much harder levels. Level 2 introduces two enemies, and the difficulty goes up. At this stage the player is encouraged to write more complex functions.

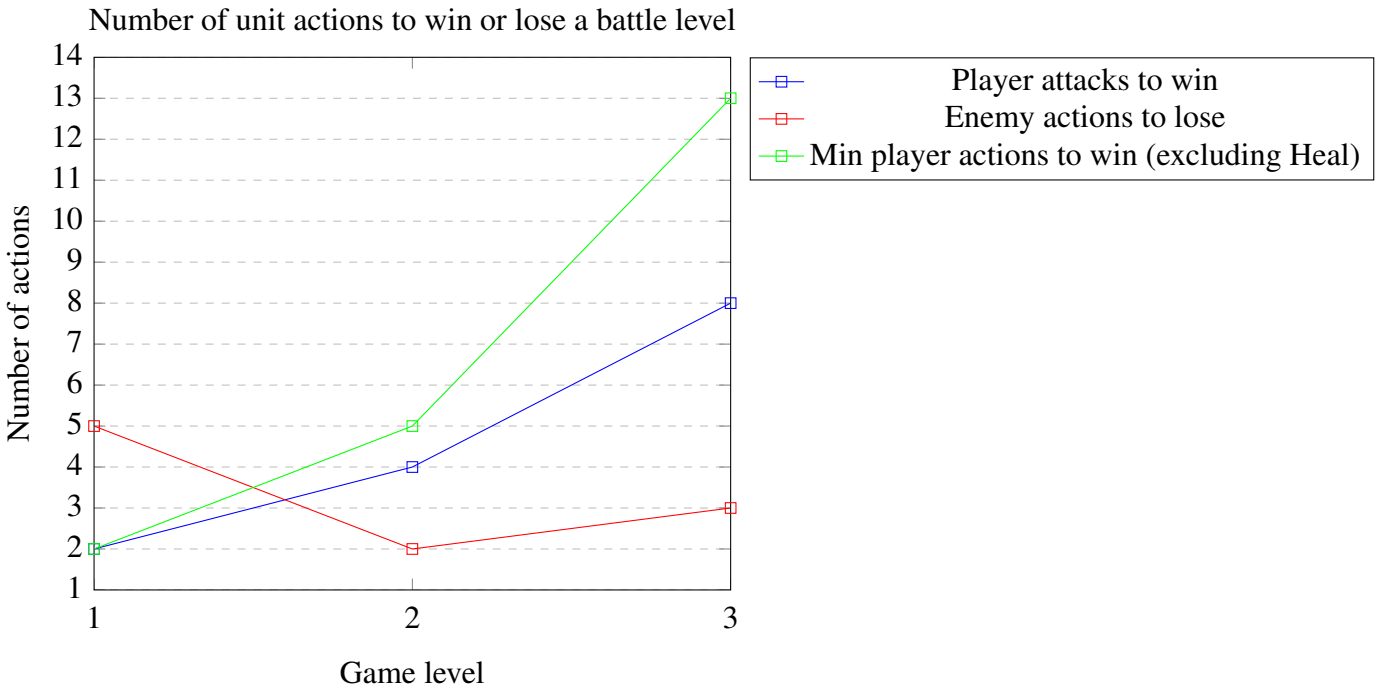


Figure 4.3: Level difficulty evaluation graph

Chapter 5

Design

5.1 Aesthetics and user experience

One of my goals for this project was to deliver a unique and engaging user experience that motivates players. While visuals, music and story were not strictly part of the game mechanics, I believe they were equally valuable for creating a pleasant learning environment.

According to Riot Games' "So you wanna make games?" series [26], game art and UI should aim for clarity and satisfaction when being interacted with. They should have a coherent style and set the right emotional tone. The tone of HaskellQuest is light fantasy and the emotional tone is excitement for adventure and learning new things about the game world. The creative process started off with a visual board of various ideas and images. I wanted the game world to feel coherent with Haskell, so the keywords such as "bugs", "functions", and "recursions" inspired a lot of the world-building. I thought, "Why not implement functions as attacks that the player casts to defeat enemies?" This application of functions felt natural and dynamic. I wanted the player to be able to express their creativity while dynamically writing the code and practising syntax rather than being bound to set options, from where came the idea of the in-game editor. This also provides a much more hands-on practical activity of experiencing various difficulties with writing code rather than a more passive activity like choosing the correct answer out of the given options. This could also prove beneficial for better understanding of the concepts following the common given advice to beginner programmers [52] [48] [62]. The concept of recursions inspired both the main gameplay loop discussed earlier in the Concept and the story. "Bugs" became Corruptions that the player fights. Just like bugs in code, they hide in the world of HaskellQuest breaking its natural processes.

5.2 Plot

The plot of the currently released version of HaskellQuest follows the player who at the start of the game finds themselves looking into the sky lying next to a lake. They then

start a conversation with the Sprite of Light who briefly introduces them to the world. The Sprite explains to the player that they need to fulfil their role as the Pilgrim of Light who needs to recover the long-forgotten art of functional programming. The Pilgrim possesses a rare artefact - a Grimoire that needs to be filled with Haskell knowledge. The player then proceeds to the first territory - the city of Caladail, also called the City of Shimmering Lights, where they meet some of the main characters. After having a dialogue with Stella (an NPC) at the tavern, the player finds out about the Archives that could contain information about what happened to the previous Haskell masters and the event called ”the Great Despair”. The road to the Archives lies through a cave pass where the player takes part in the first series of battle levels.

While the currently released version of the game is limited to the first level only (see User testing), future versions of HaskellQuest would continue the story. As mentioned earlier, the structure of the story is inspired by the concept of recursion. It has multiple endings depending on the character's choices and their performance in story-connected battle levels. These endings include:

1. Good ending
2. Neutral ending
3. Bad ending
4. Secret ending

In good, neutral and bad endings the player returns back to the lake location just like at the start of the game and can replay the game again to explore other endings. When unlocking the secret ending the user finds out about the recursion that was controlling their fate and discovers a secret action that, when activated during the final battle level, completes the game.

For writing character conversations and developing the story, I utilised Arcweave software [1] and Ink Unity Integration asset by inkle[57].

5.3 User interface

The prototype of the user interface (UI) was originally created during the pre-production stage of development. HaskellQuest required robust UI that presents a lot of information to the player during gameplay. The process started with paper sketches that were also used as a story board walk-through [60]. Afterwards, I moved to wireframes construction in Figma[6] to block out UI element layouts for different game scenes. Once I was happy with the balance, I moved onto building the interactive prototype[14]. The prototype allowed early evaluation of the UI usability, consistency and ease of completing different actions. The final version of HaskellQuest UI was designed to be harmonious with the game's main themes and aesthetics for seamless fusion with game sprites and textures [26].

The game consists of seven distinct designs of screens: Main Menu, Last Saved, Credits, Player Name Input, Story, Territory and Battle. All designs were fully implemented

Figure 5.1: Current Territory UI

in the current version of the game with the exception of Last Saved, because the Save-and-Load system was not included in this release. Main Menu and Credits designs were executed closely to the Figma prototype and had very few minor changes to them. Player Name Input and Story were added during the development stage. Later I also added features for the Grimoire (1), Help menu (2) and player statistics (3), such as the name and current level shown in Figure 5.1, which used to be accessible only from the battle level UI. Minimap is currently absent from the Territory UI but will be implemented in future versions of the game.

Designing the battle level interface shown in Figure 5.2 was the most challenging of all due to the large number of elements in it. The design splits the screen into two logical halves: left side for the game visualisation that contains action stage overlayed with the Grimoire (1), help button (2), player statistics (3), enemy statistic (4), dialogue box (5), escape button (6), compile code button (7); and right side for code writing, which consists of a flexible and user-friendly In-Game Text Editor [67] (8) and a console (9). The UI design was created to comply with Jakob Nielsen's general principles for interaction design [64]. For example, the dialogue box (5) guides the player through the battle by displaying the turn, compilation and code format errors and the effects player's actions have on the units. It complies with Neilson's heuristics of "Visibility of system status". The design of the right side is meant to replicate a common setup for writing Haskell according to the second heuristic - "Match between system and the real world", where the real-world is any other IDE. Additionally, every code error produces a detailed error message for the user to help identify the mistake and avoid making it in the future in accordance with the "Error prevention" heuristic. All icons are chosen to be as intuitive as possible, for example arrows for directions, crosses for close and question marks for help, similar to many other common software UIs. Red health and green mana bars are consistent with other games' UIs for "Recognition rather than recall".

The Grimoire element (1) is another major part of the UI/UX. It is persistent through the territory and battle level and contains discovered Haskell information. The Grimoire button opens an overlay panel shown in Figure 5.3. It contains a book-inspired interface that is a horizontally scrollable list with dynamic arrow buttons indicating if more content is available on either side. The first prototype appended new Haskell tips in the order of discovery resembling a queue. Later it was replaced with a version in which new knowledge would appear on the designated page to keep the logical order of Haskell topics, while empty pages would signal to the player that there is still knowledge to be found in the territory.

Future improvements discussed in Improvement suggestions based on user feedback and Retrospective and further development should be done to the interface to make it more intuitive and accessible.

Figure 5.2: Current Battle level UI

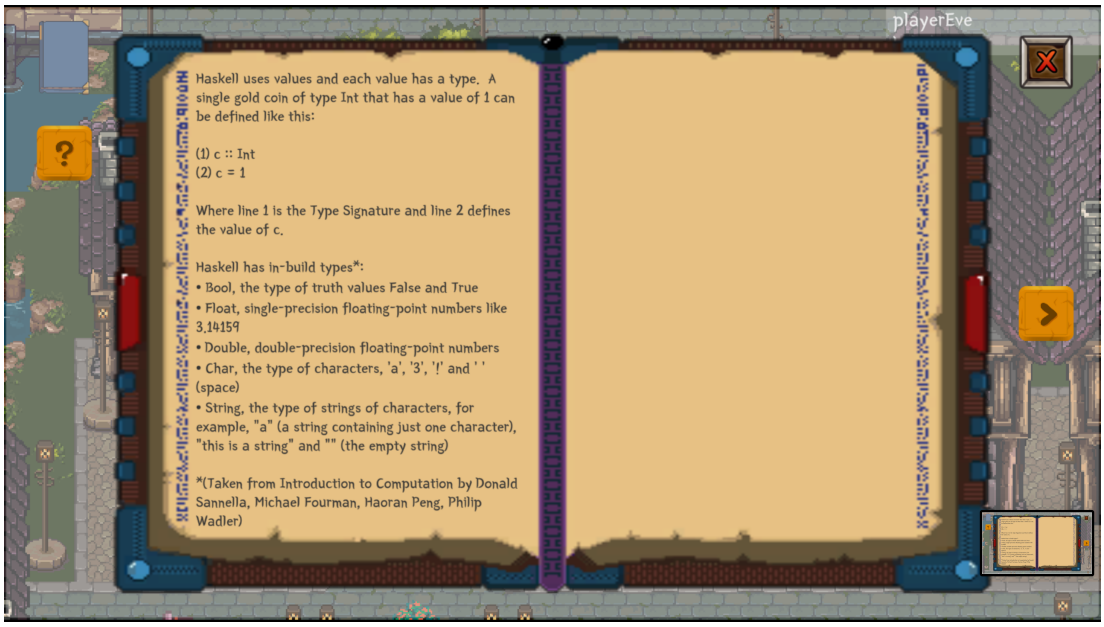


Figure 5.3: Grimoire UI

Chapter 6

Implementation

6.1 Territory

The goal for the territory mechanics was free and interactive exploration that was achieved by sprite/textures levels hierarchy such as water, grass areas and rocks. Trees, houses and other objects have a custom z-axis to allow the player to walk in-front as well as behind them. This achieves a so-called 2.5D [59] feel to the gameplay while still keeping the 2D rendering pipeline [72]. Trees, flames, water and other dynamic elements have added animations to them for aesthetic purposes and a more engaging user experience.

To balance out the game content, I started with multiple map sketches identifying the positions of ”main story” interaction points and other Haskell content. After finalising the draft, I tried to replicate the territory as close as possible with the chosen sprites from Unity Store [34] and itch.io [17]. The final version is shown in Figure 6.1. ”Main story” walkthrough is marked yellow, Haskell content marked blue and other interactions marked green. Haskell content was placed such that more crucial tips were much easier to find, while harder, more specific tips were hidden further away.

One of the challenges was scaling available assets from different packages to realistic sizes compared to each other without compromising their quality. Another aspect was camera movement. As of now, the camera is centered on the player’s sprite and follows its movements. The map has collidable boundaries that do not allow the player to wander off the edge of the map. However, it is a rather restrictive solution, and a possible future improvement for visual consistency would be to prevent the camera from moving beyond the borders, too.

Interactive items display a visual trigger when the player approaches them. There are different visual triggers for the ”main quest” and ”general” objects. The dialogue system is an adapted version of the system described in ”How to make a Dialogue System with Choices in Unity2D” by Trever Mock [63]. Once the player enters the starting territory for the first time, they are also greeted with an instruction on the controls dialogue box. The controls are a common combination of keyboard shortcuts such as ”WASD”/Arrowkeys for player sprite movement and mouse UI interactions.

Figure 6.1: Final territory implementation. Note: during the gameplay the camera zooms in on the player, and you cannot see the whole map at once. Yellow marks show the main story walkthrough and interaction points, where ”start” marks the start and 6 marks the entrance to the battle levels. Blue marks show Haskell content. Green marks show other game content.

6.2 Battle system

Battle system is responsible for sequential events in battle levels. The algorithm flowchart is shown in Figure 6.2. The battle system consists of five states: Start, Player Turn, Enemy Turn, Won and Lost. The original version of the algorithm was based of the ”Turn-Based Combat in Unity” tutorial by Brackeys [46], but was later heavily modified to fit HaskellQuest ideas.

Once the player enters a battle level, they can no longer move their sprite freely. Instead, all player battle actions should be done through the in-game text editor. The player can still interact with the interface using a mouse, and use a keyboard to write code.

The current algorithm supports the ”one player against multiple enemies” case. The player in their turn has to write code that is then sent to JDoodle server for compilation (see Haskell integration). If the code runs into a network error, the player is notified through the dialogue box. If the code compiles correctly, the game proceeds with interpreting it into game actions. Any subsequent runtime errors are displayed to the player in the dialogue box to help the player fix their code. The player can execute multiple actions in one turn, and the effects of each action are processed and tracked sequentially in real time. If the player’s code contains more actions than necessary, they will not be executed, for example if the player defeats the enemy half-way through a series of attacks. When processing individual actions, the algorithm also performs additional checks on the code format described in Writing Haskell and verifies if a given action is allowed, for example if there is enough mana for an attack. Otherwise it intentionally allows any player-coded action as long as the player is the one set to perform the action. This logic holds even if an action is not in favour of the player, for example if the player mistakenly heals an enemy or attacks themselves. During the enemies’ turn, all enemies perform an attack one by one. Similarly to the player turn, all attack effects are tracked in real time. If an enemy gets defeated, it disappears from the scene and is no longer a part of the battle. Every action is followed by a descriptive message in the dialogue box. The player can escape any battle level at any time by clicking the ”Run away” button, and the battle system algorithm will stop.

6.3 Saving data

One of the main challenges with the game was implementing data persistence between different scenes. This was crucial for preserving players data and progress such as discovered Haskell knowledge in the Grimoire, player name, level, damage and other attributes. The current game version does not support saving and loading the overall game progress yet, as that would require extensive additional research on game file encryption and safe storage. Instead, I used a Unity solution with a singleton class that is not destroyed on scenes load [73].

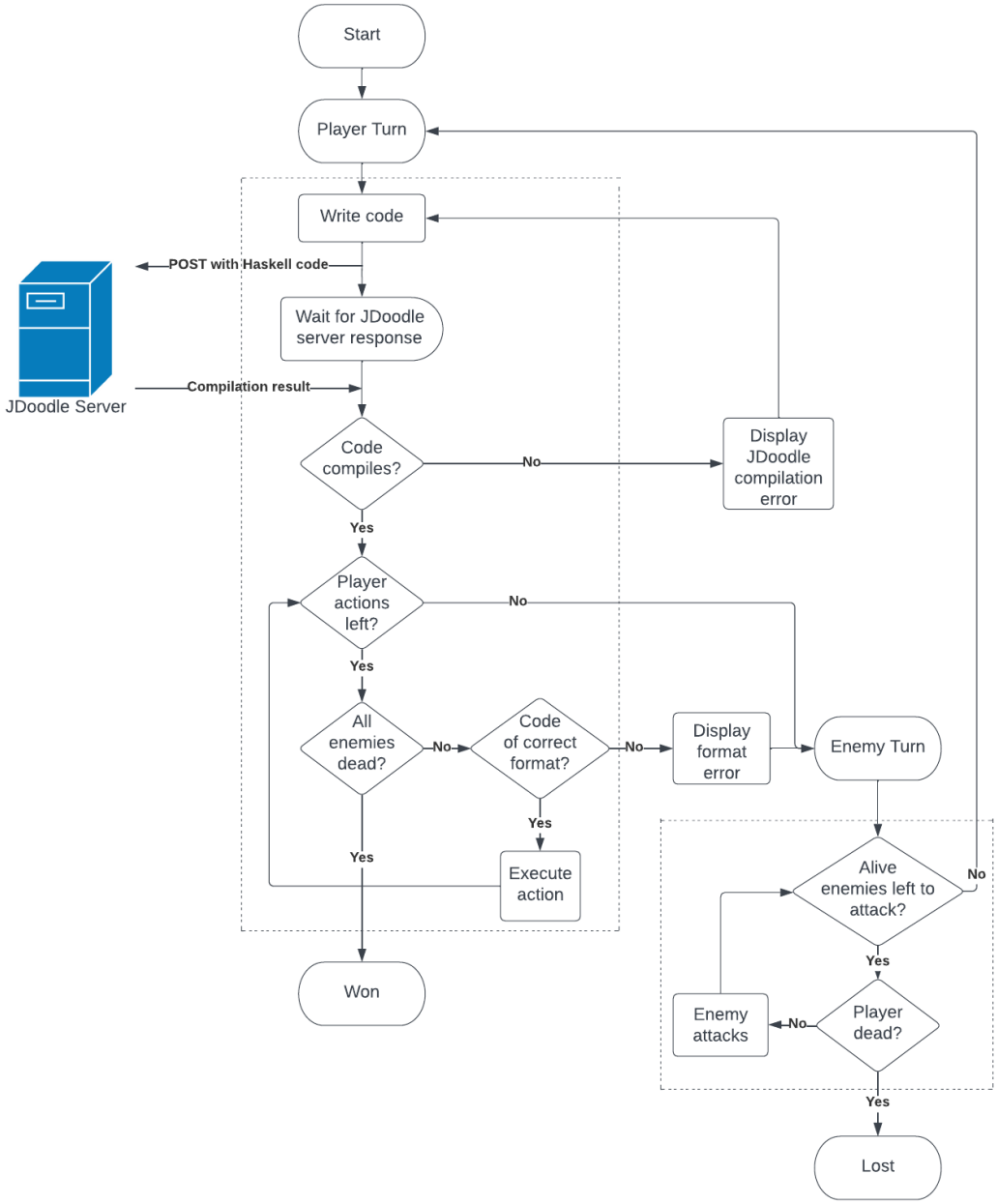


Figure 6.2: Battle algorithm flowchart

6.4 Haskell integration

One of the biggest design challenges for HaskellQuest was integrating the Haskell compiler - GHC. The necessity comes from the user's ability to freely write any Haskell code in the in-game text editor, thus making it one of the core mechanics of the game. There were two ways in which the compiler could have been integrated within the game - internal and external.

Integrating GHC compiler internally would mean either including it with the game files as a part of the build, or suggesting the users install it themselves. The first option means that the entirety of the compiler has to be installed with the game, which adds to the weight of the game, and may cause potential errors from the user environment. The second option adds extra steps to the installation, which can demotivate players and potentially add unnecessary complexity. Difficulty installing Haskell and its compiler in the first place was a common issue mentioned by both experienced developers and UoE students alike. [76] [25] [69] Since the compiler is a crucial mechanic of the game, the players' failure to successfully install the intended version would result in them not being able to access the game at all. Relying on the already installed versions of the compiler means making assumptions about players' environment and also risking introducing errors. But above anything else, HaskellQuest code would likely have to make system calls in order to use GHC which compromises the security. Lastly, this approach would limit the integration of the game to various platforms where GHC can not be installed by default such as consoles [8] if the project were to be extended to those, too.

Taking all of the above considerations into account, I decided to implement the external integration of GHC by using the services of online compilers. An online Haskell compiler meant that the only assumption I had to make about the player's environment was an active connection to the Internet. The latter is a very common game requirement that is easily achievable by the device owner. However, there is a need for maintainability: using GHC internally would be a more stable option while connecting to a compiler through an API means that HaskellQuest needs to be kept up to date with potential updates. The inspiration for this solution came originally from built-in code editors on websites designed to practise coding questions such as HackerRank [10] and LeetCode [19]. The official Haskell website [13] provides a field where site users can try and write Haskell code that then gets compiled and the result is displayed. Tracing outgoing network requests led me to TryHaskell [53] by Chris Done. This website can compile Haskell in the browser, however, it has certain limitations that I discovered later. Since the main purpose of the website is to provide an interactive Haskell tutorial, its online GHCi does not allow the upload of any supporting Haskell source files. Further research led me to multiple other online compilers: JDoodle [18], codingground [4] and replit [22]. JDoodle had the advantage of having a user API, but with a limit of 200 free requests per day. The other two were free but had no API and were aimed at collaborative code writing and coding practice. I also got in touch with RunCode's team who kindly agreed to add an online Haskell compiler in their upcoming service OnlineCompiler [20] that would be available without limitations. Unfortunately, this service could not have been completed in time for the planned user testing, but pending

a successful launch of HaskellQuest, it should be considered as an alternative resource for future work.

JDoodle services provide a REST API to send POST requests containing the code for compilation. JDoodle then responds with the result of the compilation or a detailed error from the GHC compiler as shown in Figure 6.2. Processing these messages allowed me to handle Haskell code as strings in a much more elegant and reliable way compared to calling Haskell directly from C#. Additionally, it acts as an additional abstraction layer such that the game code cannot interfere with player-written code and vice versa. For example, if the player writes an infinite recursion, the service would respond back with an error informing the user that the recursion limit has been reached instead of potentially crashing the game and the platform it runs on.

Chapter 7

Evaluation

I express my gratitude towards everyone who has kindly participated in the user testing phase and provided insightful and thoughtful feedback. Overall, I received many positive comments about the game concept and world structure around Haskell, as well as the gameplay experience and the battle system flexibility 7.1. This validates the success of the game approach towards teaching Haskell and the chosen mechanics systems.

> ”I felt the theme encourages motivation to learn Haskell - some things that have put me off learning Haskell (compared to other languages) before have been that it’s seen as a language more for research than professional use, and that t expects abstract, ’arcane’ knowledge (e.g. the ’A monad is just a monoid in the category of endofunctors’ meme). The fantasy setting reframes these as a positive though, akin to wizardry.”
>
> ”The way you learn about Haskell by looking around the world.”
>
> ”The game concept factors in the core goal of teaching Haskell into all of its mechanics and design. Making it feel very cohesive.”

Figure 7.1: A few of the responses to the form question ”What do you like the most about HaskellQuest?”

7.1 User testing

The majority of the feedback came from the open user testing. Participants were provided with instructions on how to install the game together with two feedback forms. One of the forms focused on user experience with the game (a single submission per user), the other one was for submitting bug reports (unlimited submissions per user). Both of the forms, as well as complete results with visualisations are attached in Appendix B User testing.

User testing of the game included the first territory and three battle levels. As discussed in the Concept chapter, each territory will eventually be dedicated to a single Haskell

topic. However, in the currently released version of the game the first territory contains mentions of all topics listed in the Writing Haskell section. While at the moment this temporarily increases the difficulty level of the starting territory for beginners, this was done for the benefit of more experienced testers to give them a chance to attempt to write more advanced Haskell code and test the core battle system on a wider range of inputs. The difficulty of battle levels as such remained unaltered and they could still be completed by less experienced users.

Overall, 18 people took part in user testing. The participants had a wide range of previous experiences with Haskell, with half of them having little-to-no experience (chose 1-2 on a scale of ”1 - never seen Haskell before” to ”6 - proficient in Haskell”) and 8 (44%) identifying as ”moderately experienced” or ”confident” (4-5 on the scale). However, regardless of their proficiency level, the majority of participants found HaskellQuest not as difficult as I had expected despite the abundance of Haskell topics included in the test version of the game. It was given an average difficulty of 4 out of 6 with 15 (83%) votes falling between 3-5 on a scale where 6 is very easy. The playing experience was rated on average at 4.67 on a scale of ”1 - Non satisfactory” to ”6 - Would definitely play again”. 11 (61%) testers gave a positive rating 5 or 6, and 6 (33%) gave the highest rating of 6, making it the most frequent response. This is a major success for the project for one of its key goals: ”HaskellQuest should provide an engaging and unique user experience that motivates players” (see section Project goals).

The majority of testers also believed that HaskellQuest would be helpful to learning Haskell, with 15 (83%) of participants choosing between 4-6 on a scale of ”1 - Not helpful at all” to ”6 - Very helpful”. When assessing player interactions with the game mechanics, most testers showed good understanding of the game instructions, with the mode rating of 5 (33%) and the average rating of 4.39 on a scale of ”1 - Very hard to understand” to ”6 - Very easy”. However, some clarity should be added in future versions of the game as the second most frequent rating (28%) was 3 out of 6. Haskell content in NPC dialogues was found useful to the gameplay with 15 (83%) of testers rating it above 4 out of 6.

7.2 Improvement suggestions based on user feedback

After analysing all of the responses, I generated a list of improvements based on their level of priority and impact.

7.2.1 High impact/priority improvements

The number of attacks a player can perform during the level is restricted by mana. Each attack costs a certain amount of mana, and when mana level reaches 0, no action can be performed. The intention was to encourage players to optimise their code. During user testing, however, multiple players pointed out that they can still write code similar to lower levels and beat more difficult bosses by simply adding a RestoreMana action. Therefore, one of the biggest improvements should be balancing the player abilities. A better solution for the future would be to limit the usage of RestoreMana action by allowing it only as a consumable item and/or to deplete different amounts of mana

depending on the syntax used. For example, a solution with a hard-coded list of attacks would drain significantly more mana than an elegant list comprehension.

I have also received feedback requesting more Haskell content and use case examples. At the moment in HaskellQuest the user has to enter the battle level to progress to a new territory. In contrast, in ”Octopath Traveller” [27], for example, the player travels between quest zones using roads where they may or may not encounter enemies. Those chance encounters, compared to the more common case of enemies bound to a location, bring an extra dynamic to the game. Additionally, to make the world feel more open, it might be better if the player could encounter enemies in different areas during exploration, too. Some other suggestions included the addition of side quests and a practice arena for ”showing the player that their gained knowledge is applicable to the games challenges early”, as well as adding a more visual tutorial for the first battle level, similar to how it is done for the territory scene.

7.2.2 Medium impact/priority improvements

There were a few requests for saving the written code in the text editor when progressing through levels. As mentioned in Game mechanics, players’ code is not preserved with the intention of encouraging players to try different approaches. It can indeed be argued, however, that this system can make it more difficult for the players to improve the already written code and quickly become an annoyance. A lot of testers admitted that they were manually copy-pasting their code between levels. One of the more interesting suggestions included the introduction of ”spells” - attacks previously written by users that could be pasted into the code during battle. This mechanics could further be evolved into a skill tree common in many games such as ”The Elder Scrolls V: Skyrim” [30] and ”Path of Exile” [21]. Such an addition to the game would facilitate a more dynamic gameplay and game lore while still letting the players practise Haskell.

- A more flexible dialogue navigation
- Adding visual representations of units’ attributes to battle level UI
- More visual content to Haskell content in the Grimoire such as pictograms

7.2.3 Low impact/priority improvements

- Visual tag to identify that new knowledge was added to the Grimoire
- Better terrain delineations in certain places such as stairs
- When the player runs away from the battle level, they need to return to the cavepath entrance instead of the starting location on the territory
- Music and volume controls
- Grimoire section headers
- Preserving console history similar to terminal
- Adding collision boxes for NPCs

7.3 Known issues and limitations

While the current version of HaskellQuest was thoroughly tested during both development and user testing, there is no guarantee that it is completely bug-free. Some areas in need of further testing and potential improvements are:

- Handling of special characters in the in-game text editor
- Image consistency when scaling to different window/screen sizes
- Player text input sanitation on the Name input screen and Battle level
- Jdoodle account scaling

JDoodle service requires a creation of an account with assigned secrets that need to be included within the POST message. The number of available requests would depend on the account tier. At the moment HaskellQuest operates from one account that was given access to a large number of requests for the duration of the user testing. However, this solution is not secure and is not scalable in the long run. One of the potential fixes would be the creation of separate accounts for each new player through a game registration system, however, this solution requires further research on security and player privacy and safekeeping of player data.

In addition to the above, some bugs were also reported that did not restrict the gameplay but had an impact on user experience and should be fixed in future releases, including:

- Some inconsistent collision boxes of items and NPCs affecting smooth interactions with them
- Fixing some order of sprites on the z-axis

Chapter 8

Conclusion

8.1 Retrospective and further development

At the start of HaskellQuest, there were many features and ideas that were falling outside of the time limitations for this project. I had to heavily prioritise the core mechanics and main ideas in order to deliver a playable version of the game. However, there is a list of planned future features that would further enhance the gameplay and user experience.

When working on HaskellQuest, I did research on games like ”Octopath Traveller” [27] and ”Darkest Dungeon” [5], both of which are popular turn-based games. I enjoyed the immersive world and aesthetics of ”Octopath Traveller” as well as the interesting features of its battle system. ”Octopath Traveller” has a unique ”Break system” [37] in which every enemy has a set of attacks they are vulnerable to. This idea could be well translated into HaskellQuest by making enemies vulnerable to specific functions. This coincides with some users' requests to increase the difficulty of the game.

”Darkest Dungeon” has high-paced gameplay, which is very engaging from the start. This thrill is achieved by adding time pressure by limiting the life of the torch that the party carries in a dark dungeon. As the flame grows weaker, enemies are getting stronger and characters are getting stressed. I believe a similar technique could help increase the difficulty and players' excitement for some battle levels. It also inspired me to come up with the following features:

1. Code line limitation
2. Levels that only allow using certain Haskell topics
3. Limitation on available actions
4. Time limit to fix non-compiling code before the enemy attacks

Some other features include:

- More playable characters with unique abilities that can be used to build a team
- A save-and-load system
- Syntax highlighting

- Interactive in-game items such as currency, potions and collectables
- Player inventory
- A more advanced reward system
- Multiplayer support
- Minimap UI to improve exploration and a full map panel
- Character weapons and armor with certain effects on gameplay
- UI panel for level transitioning
- Lighting system
- Character customisation
- Walking inside houses

8.2 Main achievements

The main achievement of this project is the implementation of all the core mechanics necessary for a functional game. This includes story mode, open-world mechanics, dialogues, Grimoire and battle system. This allows any further improvements and features to be built on top of an already existing and functional system. Moreover, the game in its current state can be easily scaled-up to add more levels and content with all major UI elements, animations, mechanics built-in as prefabs. For example, the battle system can be applied to any level of any difficulty, and creating a new enemy or an NPC is as easy as changing the sprite and unit details.

Another achievement is the completion of all the three project goals I set at the start in Introduction. Not only have I gained multiple applied skills with developing the game, but also learnt a lot about the game industry, community, resources, current trends and recent developments. Developing this game and especially its gameplay centred around Haskell helped me understand Haskell concepts in greater depth than before. HaskellQuest was a very engaging project, yet challenging in many areas, from UI/UX, creative writing and animation to architecture, networks and security.

Bibliography

[1] Acweave. https://arcweave.com/. Online; accessed: 9-April-2023.

[2] Codecademy Docs. https://www.codecademy.com/resources/docs. Online; accessed: 9-April-2023.

[3] CodinGames. https://www.codingame.com/home. Online; accessed: 17-December-2022.

[4] codingground Online Haskell Compiler. https://www.tutorialspoint.com/compile_haskell_online.php. Online; accessed: 9-April-2023.

[5] Darkest Dungeon. https://www.darkestdungeon.com/. Online; accessed: 9-April-2023.

[6] Figma: the collaborative interface design tool. https://www.figma.com/. Online; accessed: 9-April-2023.

[7] GameMaker. https://gamemaker.io/en. Online; accessed: 9-April-2023.

[8] GHC Wiki Platform. https://gitlab.haskell.org/ghc/ghc/-/wikis/platforms. Online; accessed: 9-April-2023.

[9] Godot. https://godotengine.org/. Online; accessed: 9-April-2023.

[10] HackerRank. https://www.hackerrank.com/. Online; accessed: 9-April-2023.

[11] HackerRank Functional Programming. https://www.hackerrank.com/domains/fp?filters%5Bstatus%5D%5B%5D=unsolved. Online; accessed: 9-April-2023.

[12] Haskell in industry. https://wiki.haskell.org/Haskell_in_industry. Online; accessed: 9-April-2023.

[13] Haskell.org. https://www.haskell.org/. Online; accessed: 9-April-2023.

[14] HaskellQuest UI Prototype in Figma. https://www.figma.com/proto/06WxmfW89YjV18i00MqSei/Haskell-Quest?page-id=0%3A1&node-id=1-2&viewport=490%2C347%2C0.32&scaling=min-zoom&starting-point-node-id=1%3A2. This is my original work.

[15] HaskellWiki. Introduction. https://wiki.haskell.org/Introduction#What_is_Haskell.3F. Online; accessed: 17-October-2022.

[16] intogames. https://intogames.org/. Online; accessed: 5-June-2022.

[17] itch.io. https://itch.io/. Online; accessed: 9-April-2023.

[18] JDoodle. https://www.jdoodle.com/. Online; accessed: 9-April-2023.

[19] LeetCode. https://leetcode.com/. Online; accessed: 9-April-2023.

[20] OnlineCompiler. https://onlinecompiler.io/. Online; accessed: 9-April-2023.

[21] Path of Excile. Passive skill tree. https://www.pathofexile.com/passive-skill-tree. Online; accessed: 9-April-2023.

[22] Replit. https://replit.com/. Online; accessed: 9-April-2023.

[23] RPG Maker. https://www.rpgmakerweb.com/. Online; accessed: 9-April-2023.

[24] ScalaQuest - the game to learn Scala. https://www.kickstarter.com/projects/andanthor/scalaquest-a-game-to-learn-scala. Online; accessed: 9-April-2023.

[25] School of Informatics Intranet. Informatics Teaching Organisation. Student course feedback. https://web.inf.ed.ac.uk/infweb/student-services/ito/admin/course-survey-reports. Online; accessed: 20-October-2022.

[26] So You Wanna Make Games?? https://www.riotgames.com/en/artedu. Online; accessed: 9-April-2023.

[27] Square Enix Octopath Traveller. https://www.square-enix-games.com/en_GB/games/octopath-traveler. Online; accessed: 9-April-2023.

[28] Teamgantt. https://www.teamgantt.com/h2. Online; accessed: 9-April-2023.

[29] The University of Edinburgh. Undergraduate study - 2023 entry. https://www.ed.ac.uk/studying/undergraduate/degrees/index.php?action=view&code=G401. Online; accessed 13-October-2022.

[30] the Unofficial Elder Scrolls Pages. Skyrim:Skills. https://en.uesp.net/wiki/Skyrim:Skills. Online; accessed: 9-April-2023.

[31] Trello. trello.com/. Online; accessed: 9-April-2023.

[32] UCAS. Computer science 2023. https://www.ucas.com/explore/subjects/computer-science. Online; accessed 13-October-2022.

[33] Unity. https://unity.com/. Online; accessed: 9-April-2023.

[34] Unity Asset Store. https://assetstore.unity.com/. Online; accessed: 9-April-2023.

[35] Unreal Engine. https://www.unrealengine.com/en-US. Online; accessed: 9-April-2023.

[36] Leetcode Support Feedback. https://leetcode.com/discuss/feedback/136097/please-provide-haskell-language-support, October 2018. Online; accessed: 9-April-2023.

[37] Octopath Traveler - Overview Launch Trailer - Nintendo Switch. https://www.youtube.com/watch?v=Fmi8KrntszI, July 2018. Online; accessed: 9-April-2023.

[38] BCS landscape review: Computing qualifications in the UK. https://www.bcs.org/policy-and-influence/education/bcs-landscape-review-computing-qualifications-in-the-uk/, March 2022. Online; accessed: 17-October-2022.

[39] BCS. Record numbers have applied for UK computer science degrees this year. https://www.bcs.org/articles-opinion-and-research/record-numbers-have-applied-for-uk-computer-science-degrees-this-year, February 2022. Online; accessed: 17-October-2022.

[40] BCS. Record numbers of students choose Computer Science A Level in 2022. https://www.bcs.org/articles-opinion-and-research/record-numbers-of-students-choose-computer-science-a-level-in-2022/, June 2022. Online; accessed: 17-October-2022.

[41] Top Games Made with Unity: Unity Game Programming . https://www.create-learn.us/blog/top-games-made-with-unity/, February 2022.

[42] What vertical slicing is and how it's used in project management. https://monday.com/blog/project-management/vertical-slice/, September 2022. Online; accessed: 9-April-2023.

[43] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 1985.

[44] Peter Achten. The soccer-fun project. *Journal of Functional Programming*, 21(1):1–19, 2011.

[45] José Bacelar Almeida, Alcino Cunha, Nuno Macedo, Hugo Pacheco, and José Proença. Teaching how to program using automated assessment and functional glossy games (experience report). *Proc. ACM Program. Lang.*, 2(ICFP), July 2018.

[46] Brackeys. Turn-Based Combat in Unity. https://www.youtube.com/watch?v=_1pz_ohupPs&t=537s, November 2019. Online; accessed: 9-April-2023.

[47] Manuel M. T. Chakravarty and Gabriele Keller. The risks and benefits of teaching purely functional programming in first year. *Journal of Functional Programming*, 14(1):113–123, 2004.

[48] John D. Cook. The value of typing code. https://www.johndcook.com/blog/2012/12/18/the-value-of-typing-code/, December 2012. Online; accessed: 9-April-2023.

[49] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: Defining gamification. volume 11, pages 9–15, 09 2011.

[50] Darina Dicheva, Christo Dichev, Gennady Agre, and Galia Angelova. Gamification in education: A systematic mapping study. *Journal of Educational Technology Society*, 18(3):75–88, 2015.

[51] Edsger Wybe Dijkstra. Letter To the members of the Budget Council. https://www.cs.utexas.edu/users/EWD/transcriptions/OtherDocs/Haskell.html, 2001. Online; accessed 13-October-2022.

[52] Andrea Diotallevi. 5 Benefits Of Typing V Copy Pasting When Working On New Code. https://www.andreadiotallevi.com/blog/5-benefits-of-typing-v-copy-pasting-when-working-on-new-code, September 2022. Online; accessed: 9-April-2023.

[53] Chris Done. Try Haskell. https://tryhaskell.org/. Online; accessed: 9-April-2023.

[54] Jeff Drake. 24 Great Games That Use The Unreal 4 Game Engine. https://www.thegamer.com/great-games-use-unreal-4-game-engine/?newsletter%20popup=1shenmue-3, March 2023. Online; accessed: 9-April-2023.

[55] Santa Dreimane. *Gamification for Education: Review of Current Publications*, pages 453–464. Springer International Publishing, Cham, 2019.

[56] HaskellWiki. Haskell in education. https://wiki.haskell.org/index.php?title=Haskell_in_education&oldid=63321, 2020. Online; accessed 13-October-2022.

[57] inkle. Ink Unity Integration. https://assetstore.unity.com/packages/tools/integration/ink-unity-integration-60055. Online; accessed: 9-April-2023.

[58] K. Galov J. Torrång. Unity Tutorial Project in Haskell with Apecs and SDL2. https://github.com/mewhhaha/apecs-unity-tutorial-haskell, December 2022. Online; accessed: 9-April-2023.

[59] S. Knight. What Are 2.5D Games? How They Differ From 2D and 3D Games. https://www.makeuseof.com/what-are-2-5d-games-2d-3d/, June 2021. Online; accessed: 9-April-2023.

[60] Rachel Krause. Storyboards Help Visualize UX Ideas. https://www.nngroup.com/articles/storyboards-visualize-ideas/, July 2018. Online; accessed: 9-April-2023.

[61] Miran Lipovača. *Learn You a Haskell for Great Good!: A Beginner's Guide*. No Starch Press, 2011.

[62] Joe Mayberry. The Value of Re-Typing Code. https://www.wearediagram.com/blog/value-of-retyping-code, March 2013. Online; accessed: 9-April-2023.

[63] Trever Mock. How to make a Dialogue System with Choices in Unity2D — Unity + Ink tutorial 2021. https://www.youtube.com/watch?v=vY0Sk93YUhA, September 2021. Online; accessed: 9-April-2023.

[64] Jakob Nielsen. 10 Usability Heuristics for User Interface Design. https://www.nngroup.com/articles/ten-usability-heuristics/, November 2020. Online; accessed: 9-April-2023.

[65] Devin Pickell. The 7 Stages of Game Development. https://www.g2.com/articles/stages-of-game-development#5-pre-launch, October 2019. Online; accessed: 23-December-2022.

[66] Egor Piskunov. Unity Vs GameMaker: What to Choose for Game Development? https://ilogos.biz/unity-vs-gamemaker-what-to-choose-for-game-development/, August 2022. Online; accessed: 9-April-2023.

[67] Sandro Ropelato. In-Game Text Editor. https://assetstore.unity.com/packages/tools/gui/in-game-text-editor-199113. Online; accessed: 9-April-2023.

[68] Don Sannella, Michael Paul Fourman, Haoran Peng, and Philip Wadler. *Introduction to Computation: Haskell, Logic and Automata*. Undergraduate Topics in Computer Science. Springer International Publishing, 1 edition, February 2022.

[69] C. Scalfani. Why is Learning Functional Programming So Damned Hard? https://cscalfani.medium.com/why-is-learning-functional-programming-so-damned-hard-bfd00202a7d1, November 2019. Online; accessed: 23-October-2022.

[70] Ashley Smith. HSRogue: A roguelike in Haskell. https://aas.sh/project/hsrogue/, March 2019. Online; accessed: 9-April-2023.

[71] Anastasia Stefanuk. Pros and Cons of Hiring Freelance Unity Developer. https://mobilunity.com/blog/pros-and-cons-of-hiring-freelance-unity-developer/. Online; accessed: 9-April-2023.

[72] Sunny Valley Studio. How to sort sprites by Y axis in Unity 2D. https://www.sunnyvalleystudio.com/blog/how-to-sort-sprites-in-unity, August 2022. Online; accessed: 9-April-2023.

[73] Unity Technologies. Implement data persistence between scenes. https://learn.unity.com/tutorial/implement-data-persistence-between-scenes, November 2022. Online; accessed: 9-April-2023.

[74] Sruthi Veeraraghavan. Top 20 Best Programming Languages To Learn in 2023. https://www.simplilearn.com/

best-programming-languages-start-learning-today-article, March 2023. Online; accessed: 9-April-2023.

[75] Jan Christopher Vogt. Writing 2D games using super simple Haskell. https://www.youtube.com/watch?v=CfaCfQstBIM/, August 2020. Online; accessed: 12-April-2023.

[76] Philip Wadler. Why no one uses functional languages. *ACM Sigplan Notices*, 33(8):23–27, 1998.

Appendix A

Questionnaire

2. Have you had experience with functional programming and/or Haskell before you started Informatics 1?

More Details Insights

Yes	10
No	36

3. If yes, was it at school?

More Details

Yes	6
No	4

4. Could you please share the name of the school and the country where it is located

More Details Insights

6
Responses

Latest Responses

3 respondents (**50**%) answered **England** for this question.

5. Was Haskell your first functional language?

More Details Insights

Yes	41
No	5

6. If not, please specify which functional language was your first

More Details Insights

5
Responses

Latest Responses

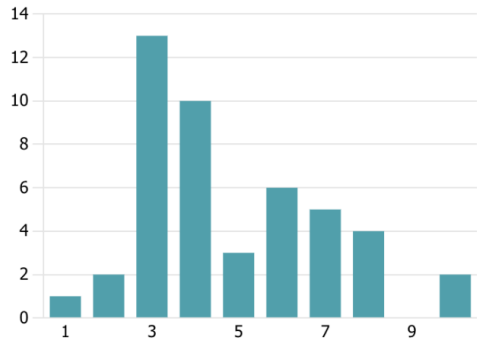
1 respondents (**20**%) answered **Python without classes** for this question.

Scheme
Python without classes
scala **functional language**

7. On a scale of 1-10 how hard did you find picking up Haskell? (1 very easy, 10 very hard)

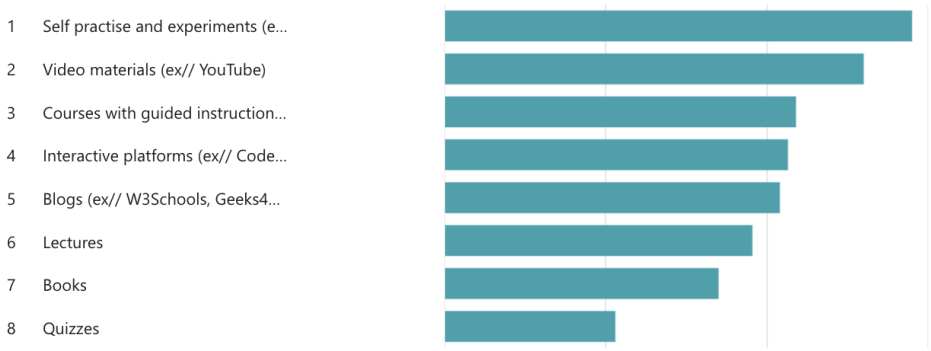
More Details Insights

4.83
Average Rating



8. When starting learning new programming languages, which learning resources or strategies do you prefer? (Imagine all of the resources are equally available. Your raking can be an approximate order)

More Details



9. Which Haskell concepts/topics did you find the hardest?

More Details Insights

46
Responses

Latest Responses
"Monads"
"monad"
"Recursion"

10. Do you think learning Haskell helped you understand other programming paradigms and logic better?

More Details Insights

Yes	30
No	3
Maybe	13

11. Have you used Haskell (or other FP) later on in your career/studies?

More Details Insights

Yes	19
No	27

12. If yes, please specify

More Details Insights

19
Responses

Latest Responses

"Simple testing of mathematical functions in math classes"

"Reasoning and Agents course. I also learned Agda for TSPL course"

A.1 Participants' information sheet

Participant Information Sheet

Project title:	HaskellQuest: a game for teaching functional programming in Haskell
Principal investigator:	Don Sannella
Researcher collecting data:	Eva Bogomil
Funder (if applicable):	NA

This study was certified according to the Informatics Research Ethics Process, RT number **7094**. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

This is an undergraduate student research project conducted by Eva Bogomil and supervised by Donald Sannella.

What is the purpose of the study?

The goal of the project is to create a teaching game especially targeted at students/ developers who are starting to learn Haskell. I would like to learn about students' experience learning Haskell. The results would help me design the game's features to cater to students' needs.

Why have I been asked to take part?

Research target group are Informatics students who have taken Informatics 1 or have other previous experience with Haskell.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time, without giving a reason. Your rights will not be affected. If you wish to withdraw, contact the PI. We will stop using your data in any publications or presentations submitted after you have withdrawn consent. However, we will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part?

Specify:

- Kinds of data being collected: students experience learning Haskell. No personal data.
- Means of collection: questionnaire
- Duration of session: up to a participant, approximately > 5 min
- If participant audio/video is being recorded: No

- How often, where, when: Once

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

If participant are interested, they can take part in the early closed testing to preview the early version of the game.

What will happen to the results of this study?

The results of this study may be summarised in published articles, reports and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. Your data may be archived for a minimum of 2 years.

Data protection and confidentiality.

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will be referred to by a unique participant number rather than by name. Your data will only be viewed by the researcher Eva Bogomil and supervisor Don Sannella.

All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or on the University's secure encrypted cloud storage services (DataShare, ownCloud, or Sharepoint) and all paper records will be stored in a locked filing cabinet in the PI's office. Your consent information will be kept separately from your responses in order to minimise risk.

What are my data protection rights?

The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure and objection. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.

For general information about how we use your data, go to: edin.ac/privacy-research

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Eva Bogomil at s1917296@ed.ac.uk.

THE UNIVERSITY of EDINBURGH informatics

If you wish to make a complaint about the study, please contact

inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint.

Updated information.

If the research project changes in any way, an updated Participant Information Sheet will be made available on http://web.inf.ed.ac.uk/infweb/research/study-updates. [NB: the PI should notify the Ethics panel on inf-ethics@ed.ac.uk to upload any update PIS to the website]

Consent

By proceeding with the study, I agree to all of the following statements:

- I have read and understood the above information.
- I understand that my participation is voluntary, and I can withdraw at any time.
- I consent to my anonymised data being used in academic publications and presentations.
- I allow my data to be used in future ethically approved research.

[Button here named “I agree” or “take me to the survey”]

A.2 Participants' consent form

Participant Consent Form

Project title:	HaskellQuest: a game for teaching functional programming in Haskell
Principal investigator (PI):	Don Sannella
Researcher:	Eva Bogomil
PI contact details:	don.sannella@ed.ac.uk

By participating in the study you agree that:

- I have read and understood the Participant Information Sheet for the above study, that I have had the opportunity to ask questions, and that any questions I had were answered to my satisfaction.
- My participation is voluntary, and that I can withdraw at any time without giving a reason. Withdrawing will not affect any of my rights.
- I consent to my anonymised data being used in academic publications and presentations.
- I understand that my anonymised data will be stored for the duration outlined in the Participant Information Sheet.

Please tick yes or no for each of these statements.

1.	I allow my data to be used in future ethically approved research.		
		Yes	**No**
2.	I agree to take part in this study.		
		Yes	**No**

Appendix B

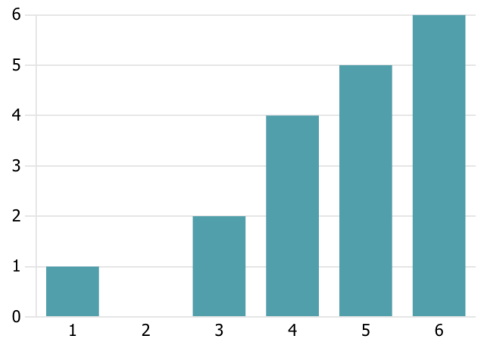
User testing

B.1 Feedback form

2. Please rate overal playing experience (1 - Non satisfactory to 6 - Would definately play again)

More Details Insights

4.67
Average Rating



3. If any what would you change to improve HaskellQuest experience?

More Details Insights

18
Responses

Latest Responses

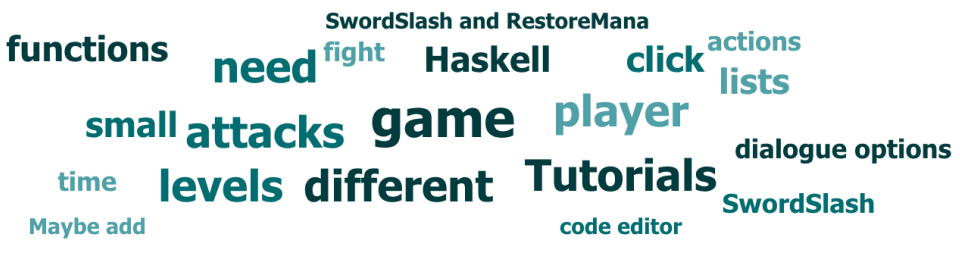
"I made a couple of notes while playing the game, most of them are tiny n..."

"You can't improve the most perfect game ever."

"More levels please, it is so much fun to play!"

Update

6 respondents (**35**%) answered **game** for this question.



4. What do you like most about HaskellQuest?

More Details Insights

18
Responses

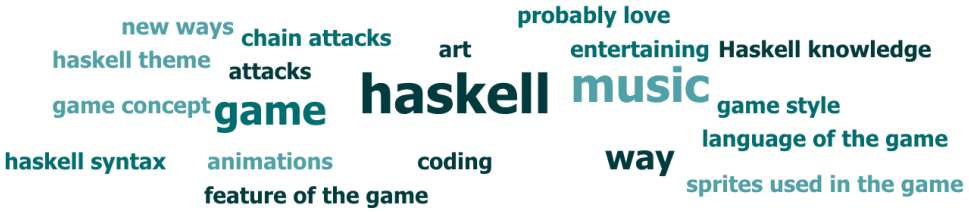
Latest Responses

"The sprites and animations are really really good. Game looks very pretty ...

"Its educational but so much fun to actually play. I am really motivated to f...

Update

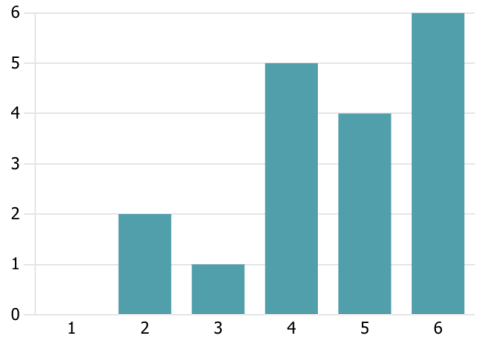
8 respondents (**47**%) answered **haskell** for this question.



5. Do you think HaskellQuest would be helpfull to learning Haskell (1 - Not helpful at all, 6 - Very helpful)

More Details Insights

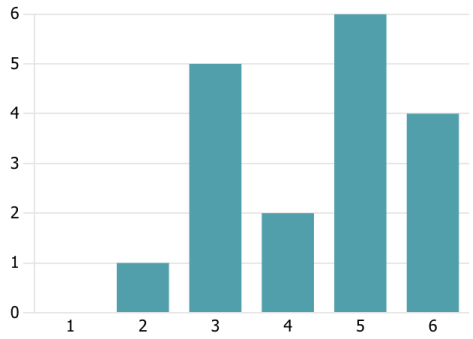
4.61
Average Rating



6. How well did you understand instructions to the game? (1 - Very hard to understand, 6 - Very easy)

More Details Insights

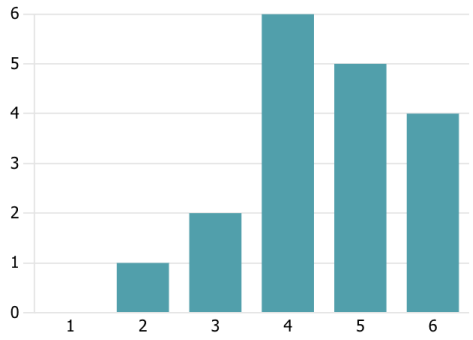
4.39
Average Rating



7. How usefull did you find Haskell content and tips from NPC dialogues within the game? (1 - Not usefull at all, 6 - Very useful)

More Details Insights

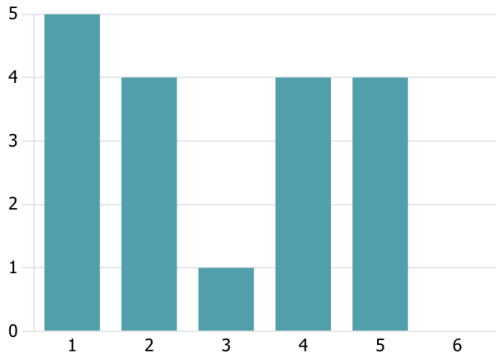
4.50
Average Rating



8. What is your previous experience with Haskell (1- never seen Haskell before, 6 - proficient in Haskell)

More Details Insights

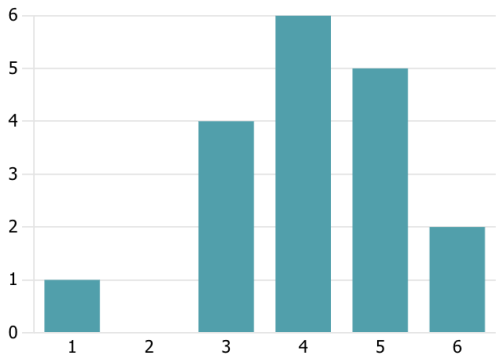
2.89
Average Rating



9. How difficult did you find the HaskellQuest? (1 - very difficult, 6 - very easy)

More Details Insights

4.11
Average Rating

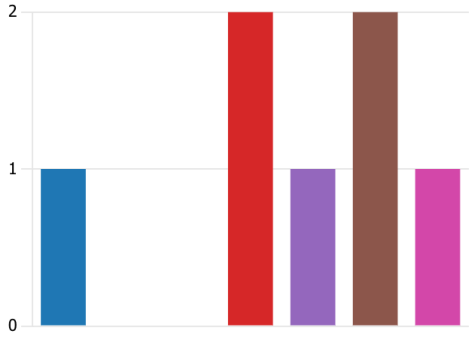


B.2 Bug report form

1. Where did you enounter the bug?

More Details

Installation	1
Main menu	0
Credits page	0
Story mode	2
Exploration/map mode	1
Battle	2
Other	1



2. Describe the bug and the steps to trigger it (the more details the better)

More Details

7
Responses

Latest Responses

"Typo: it should be "lists are" instead of "list as" (see screenshot)"

"In the windows version, it is not possible to enter special characters like "[...

"In the first town, the leftmost villa (with the two twin sisters) has a section ...

3. If applicable, submit the supporting files (ex screenshots would be very usefull)

More Details

3
Responses

Latest Responses

Screenshot 2023-04-04 003303_anonymous.png

B.3 Participants' information sheet

Participant Information Sheet

Project title:	HaskellQuest: a game for teaching functional programming in Haskell
Principal investigator:	Don Sannella
Researcher collecting data:	Eva Bogomil
Funder (if applicable):	NA

This study was certified according to the Informatics Research Ethics Process, RT number 7256. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

This is an undergraduate student research project conducted by Eva Bogomil and supervised by Donald Sannella.

What is the purpose of the study?

The goal of the project is to create a teaching game especially targeted at students/ developers who are starting to learn Haskell. I would like to learn about students' experience learning Haskell. The results would help me design the game's features to cater to students' needs.

Why have I been asked to take part?

Research target group are Informatics students who have are interested in learning Haskell or already know it. You were also likely to expose interest in taking part in early game demo/testing in previous studies.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time, without giving a reason. After this point, personal data will be deleted and anonymised data will be combined such that it is impossible to remove individual information from the analysis. Your rights will not be affected. If you wish to withdraw,

contact the PI. We will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part?

If you decide to take part you will have an option of joining: alpha testing, beta testing or both.

Alpha testing will be conducted in-person during semester weeks 4-5. The session duration will be up to an hour but you can stay for as long as you like. It will include one session where you will be given an alpha-version of the game (still in active development) to play and make as many comments as you would like mainly about the functionality, interface, ease to play etc. You can also engage in a discussion with other participants if you like. You will be provided a sheet of paper to write your feedback. Alternatively you can provide your feedback verbally to the researcher Eve Bogomil or other means like email to s1917296@ed.ac.uk. You are only required to join one session, but you are welcome to come to multiple if you would like to share more feedback.

Beta testing is conducted online during week 6-7 of the semester. You will be provided a beta-version of the game and an online form for feedback. You can complete it in your own time and spend as much time on it as you would like to. You can also submit multiple online forms. You are encouraged to make as many comments as possible.

You are welcome to reach out to me via email **s1917296@ed.ac.uk** if you would like to share more feedback or if you have any questions.

No participant audio/video is being recorded. No personal information will be collected.

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

Thank you in the credits of the game.

What will happen to the results of this study?

The results of this study may be summarised in published articles, reports and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. Your data may be archived for a maximum of 2 years. All potentially identifiable data will be deleted within this timeframe if it has not already been deleted as part of anonymization.

Data protection and confidentiality.

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will be referred to by a unique participant number rather than by name. Your data will only be viewed by the researcher/research team Eva Bogomil and supervisor Don Sannella.

All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or on the University's secure encrypted cloud storage services (DataShare, ownCloud, or Sharepoint) and all paper records will be stored in a locked filing cabinet in the PI's office. Your consent information will be kept separately from your responses in order to minimise risk.

What are my data protection rights?

The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure and objection. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Eva Bogomil at s1917296@ed.ac.uk.

If you wish to make a complaint about the study, please contact

inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint.

Updated information.

If the research project changes in any way, an updated Participant Information Sheet will be made available on http://web.inf.ed.ac.uk/infweb/research/study-updates.

Alternative formats.

To request this document in an alternative format, such as large print or on coloured paper, please contact Eva Bogomil at s1917296@ed.ac.uk.

General information.

For general information about how we use your data, go to: edin.ac/privacy-research

THE UNIVERSITY of EDINBURGH
informatics

B.4 Participants' consent form

Participant Consent Form

Project title:	HaskellQuest: a game for teaching functional programming in Haskell
Principal investigator (PI):	Don Sannella
Researcher:	Eva Bogomil
PI contact details:	don.sannella@ed.ac.uk

By participating in the study you agree that:

- I have read and understood the Participant Information Sheet for the above study, that I have had the opportunity to ask questions, and that any questions I had were answered to my satisfaction.
- My participation is voluntary, and that I can withdraw at any time without giving a reason. Withdrawing will not affect any of my rights.
- I consent to my anonymised data being used in academic publications and presentations.
- I understand that my anonymised data will be stored for the duration outlined in the Participant Information Sheet.

Please tick yes or no for each of these statements.

1.	I allow my data to be used in future ethically approved research.		
		Yes	**No**
2.	I agree to take part in this study.		
		Yes	**No**

Appendix C

Minimum Viable Product Criteria

Minimum Viable Product (”MVP”) version of the game should contain the following:

- 2D plane with player object that can be controlled using keyboard input
- Character can interact with NPCs
- Battle level where enemy object attacks character object in enemy turn
- Code editor panel where player can write Haskell code
- Haskell code should compile and correctly interpret player’s attack on the chosen enemy in player’s turn
- Both the enemy and the character must have health bars
- When either the player or the enemy are attacked, the health bar should go down
- Once the player’s or enemy’s health bar goes to 0, they die ending the battle

Appendix D

Haskell Quest Game Design Document

High Concept Information

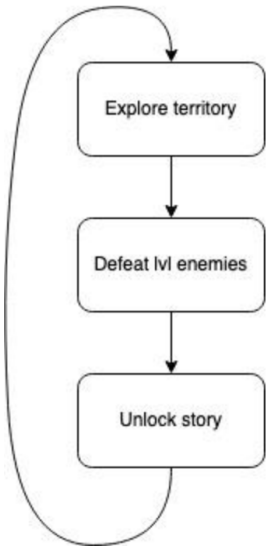
Pitch/Game concept

Haskell Quest gamifies the learning process and teaches players Haskell. The game is about a protagonist recovering long-lost knowledge of the ancient art of functional programming, set in a fantasy world. The player will have to travel through different locations of increasing difficulty and explore the world to find pieces of knowledge and apply it in the editor console to fight off enemies until eventually facing the final boss.

Three Main Game Pillars

Explore hidden ancient Haskell knowledge to fill your grimoire and recover your lost memories. This world is hiding a lot of mysteries!
Code your attacks to defeat enemies and progress through different levels.
Repeat your fate and discover different story endings

Core Gameplay Loop



Game Mechanics

Mechanics Outline

Move around and explore territory, interact with npcs, items and navigate the story with interaction windows (similar to visual novel style)
Turn by turn strategy game when entering level combat. Player writes attack code and selects an enemy to direct it to. The enemy attacks one by one in their turn.

Controls Overview (PC)

WASD - walking movements
Keyboard - code writing
Mouse right click - select game features (code editor, enemies, help etc)
I - inventory access
M - map level
H - open grimoire

Player attributes

Health - each playable character has a health bar that decreases when enemies successfully attack the character. Each battle starts with a full health bar, and it is automatically restored at the end of each level. During the level battle, health can be restored by applying health restoring items from the inventory. These items are acquired either as level rewards or though exploration.

Character specific attack type - each character has a specific role, attack and special ability. Attacks are programmed by the player and are discovered though learning different Haskell concepts. The power and durations of attacks are decided by the player but with certain limits. Special attacks are not prograble and their idea is to improve the gameplay.

Mana - mana caps the possible duration and intensity of the attack. Mana is restored at each player turn, but this could differ on certain levels when specific limitation effects are applied. It also doesn't apply to recursion attacks.

More on combat mechanics and team selection

Player has a choice of character that has different attacks and cover different Haskell concepts. For the mvp the player will only be able to choose one playable character, which is a universal character that is created by the player before the start of the game. In a full version a player can recruit certain characters from the world to their team.

Gameplay

Level Select

Once the game is opened, the player is presented with menu.

- Continue: new window with last saved versions of the game?
- New game: Starts a new game
- Credits: scroll of the people who helped work on the game
- Quit: closes game

Gameplay Goals

- Explore the territory as much as possible by playing the main story (light novel format) to learn the main concepts of Haskell. The player can not progress further unless the main story is covered.
- By interacting with the environment (NPC chats) and world items the player unlocks more tips and facts that will further expand the knowledge. This exploration is done in a free manner.
- Clear the combat level by defeating all the enemies using the skills learnt from main story and exploration.

Gameplay Scenarios

- ☐ Goal
- ☐ Opposition
- ☐ Decisions to overcome oppositions
- ☐ Rules

Think of frustration levels - keeping good frustration that is motivational but be careful not to be unfair. Since HaskellQuest is entirely build on players skill, little should be left to chance. Players should have a lot of chances to learn and improve - need a good support system.

- Open map exploration
  - No coding, just learning
  - Interaction with characters and items
- Battle scene:
  - Battle UI view
  - Turn based mechanics
- Boss battle:
  - Same UI but the gameplay has certain restrictions to make level harder
  - Restrictions:
    - Limited time to make a decision
    - Limitation to lines of code
    - Limitations to use potions (for example to restore mana)
- Dialogue view
  - Story telling, Haskell explanation

Progression Outline

Reward system

- The player is generally rewarded for experimentation and exploration
- The more the player interacts with environment the more Haskelll knowledge they can discover
- Player can find/purchase useful items from merchants, by talking to npcs and finding items in environments
- After clearing the level the player gets rewards such as coins, potions, and mysterious scraps, notes and letters with haskell knowledge. They can get special books from boss fights.

Collectables

- Coins - used to purchahse items from merchants, stored in inventory
- Potions - stored in inventory:
    - Health potions - used to restore health
    - Mana potions - used to restore mana
    - Time potions - used to slow down the timeout for certain level
- Haskell related items - once used, added to the grimoir and removed from inventory:
    - Notes
    - Letters
    - Books
    - Scrolls

Special Collectables

- Player can collect five special items to unlock a secret ending:
    - Golden pendant from Luna
    - Light holder from a creeper in Caladail
    - Piece of white robe stained with darkness in the main Cathedral
    - Funeral locket with a strand of grey hair on a lonely grave in the woods
    - Black dahlia on a porch

Narrative and Aesthetic

KEYWORDS: adventure, everything is a mystery, dark and light

Characters

- ☐ Research other games
- ☑ ~~Make character sheets~~
- ☑ ~~Create mood boards~~
- ☐ Decide iconic elements
- ☐ Think of changes that occur in development/lvl up

Playable characters:

Main character(MC):

- Player
- Player can decide the appearance
- Same uniform
- Pilgrim of Light
- Has connection with the Sprite of Light
- Starting weapon: minor staff of light
- The character doesn't remember anything
- Attack: light casting
- Special ability: potion using

Stella:

- Travelling bard, knows a lot of stories
- Is the first to join MC team
- Can use light powers due to her deep knowledge of old stories and passion for music
- Loves adventures and is here for a good time
- A bit irresponsible, easygoing, can seem selfish but cares about party members
- Wears comfortable traveling clothes: cloak, trousers, boots. Her hair is put in braids and she has scar/s on her face
- Attack: Damage to all enemies
- Special ability: blinding - paralyse shadow creatures they for n turns

Caelus:

- Divine priest
- Support/heal
- Becomes the final boss who gets corrupted by the power
- The closest to the mc, acts as a comfort character
- Gives mc clues on how to progress the journey
- Kind, knowledgeable, powerful, double faced
- Wears robes and a head piece, staff
- Attack: damage to a certain enemy
- Special ability: heal

Breta:

- Former royal knights commander
- Attack/shield role
- Strong, trustworthy, strategic thinking, loyal, straightforward
- After meeting mc thinks that helping them is her duty as a knight
- Has hard time getting used to the team but eventually opens up and is very friendly and loud
- Tall, wears old armour, middleage, brown hair, muscular, weapons: sword and shield
- Attack: usual blade attack to 2 enemies
- Special ability: shield - get enemies attack only her

Serentia:

- Scholar, mage-summoner. Studies in the mage academy but instead of using light as a direct attack, she uses her abilities to open up portals to summon different creatures and command them
- The youngest in the team
- Apathetic, quite, distant, rarely smiles, calm and collected in the face of danger
- Short, thin, pale, long hair, wears dresses and robes; weapons: summoners book/light
- Attack: summon a creature and make them attack in turn
- Specialo ability: recruit an enemy if their hp is lower than ⅛. Doesn't work on bosses

Gerdon:

- Rogue character who used to study in academy too but was expelled due to an incident in which we overused his power
- Got disowned by his family because of that and since then has been travelling the world and taking on different jobs
- Lonelly, rarely speaks, gets in trouble with Serentia, but grows fond of the team and becomes loyal
- Dark hair, hood and cape, dark outfit, weapons: dual blades
- Attack: blades double attack
- Special ability: strangle the enemy for 1 move. Doesn't work on bosses

Notable NPCs

Luna the Merchant:

- Appears in alo cities but with different iteams available to purchase

Meledian:

- Guardian of the library between the worlds that can only be opened with secret ending

Enemie types

The main theme behind enemies is that they are creatures emerged from the darkness: shadows, insect like creatures, etc

In increasing order of difficulty:

- Shadow puddles
- Shadow blobs
- Shadow spiders
- Shadow gloomers
- Shadow creepers
- Light eaters

Bosses:

- The Angler
- Isabella the Reasoner
- The Dark Curruptor

Environment Setting

The world has been submerged in darkness after the Great Despair - an event when all the natural light (ie sun and moonlight) was gone.

Locations:

- Starting lake locations - lake surrounded by blooming daffodils (?)
- Caladail: town on the river. Despite the dark night, the town looks very lively as millions of lights lighten up its streets.
- The Capital:

Possible Strory Endings

- Good ending
    a. Player successfully defeated Caelus
- Neutral ending
- Bad ending
- Secret ending

UI

According to Riot Games “So you wanna make games?” series, game art (Including UI) goals are clarity and satisfaction when interacting with. The art within the game should have a coherent style and set the right emotional tone. The tone of Haskell Quest is light fantasy (no heavy topics or realism that can distract from the learning idea) and the emotional tone should be excitement for adventure and learning new things about the world.

Inspiration games
Octopath Traveller
The Darkest Dungeon:

- High paste gameplay, very engaging from the start

- Thril of getting the torch duration affect the gameplay - enemies are getting stronger, characters are stressed
- Thrill of you can't resurrect the characters if they die. But this won't be good for a teaching game since it should be normal for the player to try again and ask for help

Visual Hierarchy

- Create focus using contrast (shape, colour, saturation, layout, detail placement etc)
- Make sure that the players know which enemies/objects/characters are more important

Audio

Soundtracks should be supporting the majority of the game scenes. I am planning on using resources such as non-copyright music:

- https://www.youtube.com/watch?v=pgLjYsVP4H0&t=198s
- Youtube studio audio library: https://studio.youtube.com/channel/UCg10-zg3ZsFVjaLq_nde1gQ/music
- Bfxr software: https://www.bfxr.net/
- https://www.youtube.com/watch?v=YIcxvt1-Rzw as opening music

Appendix E

Relevant UI sketches and prototypes

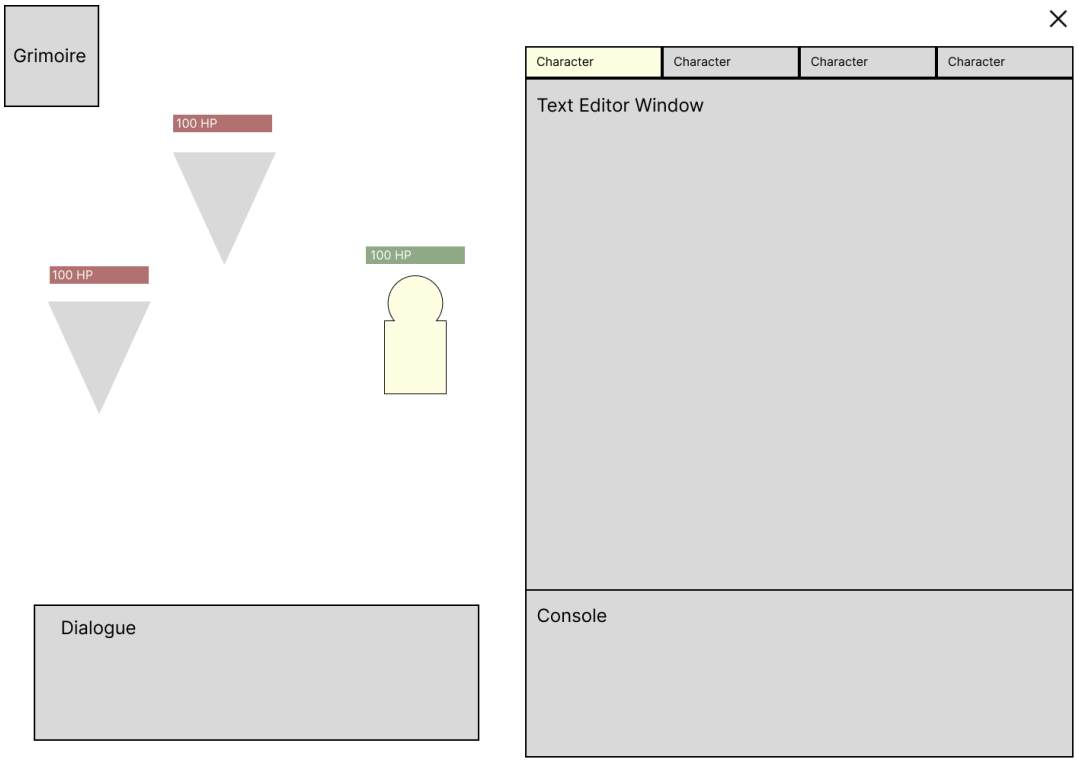


Figure E.1: Battle level UI draft

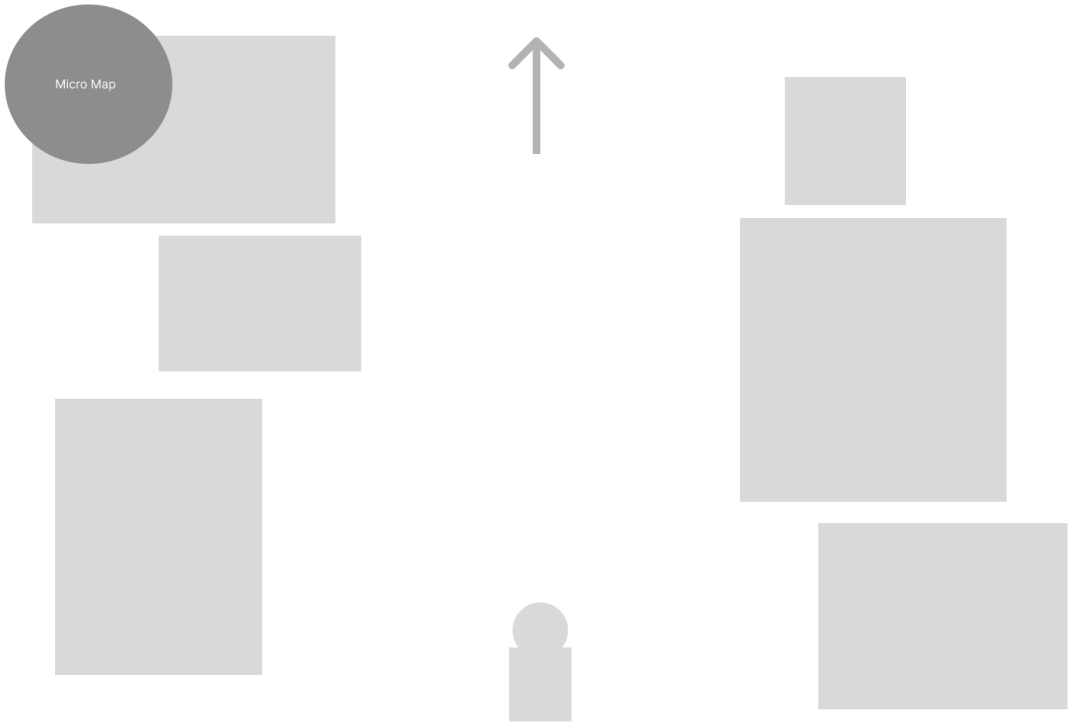


Figure E.2: Territory UI draft

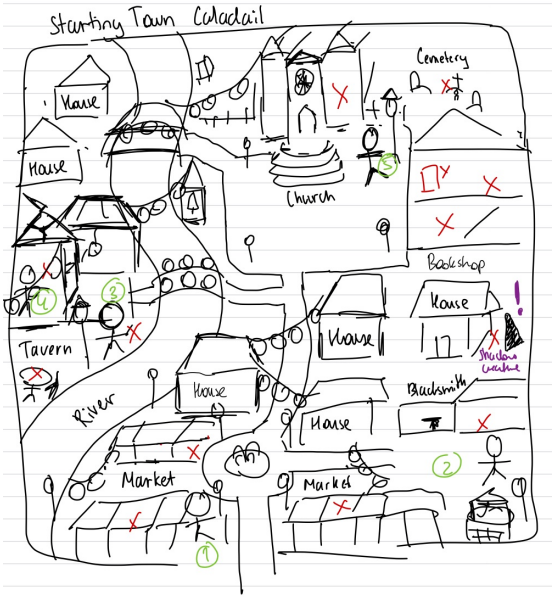


Figure E.3: Caladail map sketch

Appendix F

Game music

Music titles used in HaskellQuest: ”Explorer”, ”Enchanted Forest”, ”Magic Tavern”, ”Battle Loop” by Alexander Nakarada (www.serpentsoundstudios.com). Licensed under Creative Commons BY Attribution 4.0 License http://creativecommons.org/licenses/by/4.0/.