Having Fun Learning - A Hidden Component to Success

Maxim Despinoy

4th Year Project Report
Software Engineering
School of Informatics
University of Edinburgh

2023

Abstract

Learning new information can be a difficult process. This difficulty can seem to occur for numerous reasons, the foremost of these being a lack of engagement with the subject matter.
Ultimately to learn something new the student must *want* to learn it at some level. This report covers the HaskellQuest project, with the overall idea to help students want to interact with learning Haskell concepts and syntax, or further adding to their interest if it was already there. HaskellQuest is a learning tool in the guise of a video game, trying to take the qualities that make video games such a popular past-time and use them to make learning interesting.
This report details the theory behind major design decisions and of the implementation itself, and some analysis on user data that was gathered from players after they had played the game.

Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee.
Ethics application number: rt #7313
Date when approval was obtained: 2023-02-23
The core information included in the participants' information sheet and a consent form 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.

(*Maxim Despinoy*)

Acknowledgements

I would like to begin by expressing an enormous amount of gratitude to my supervisor Don Sanella for his continuous feedback, support and guidance given every week. It was essential to making this project the best it could be.

A sincere thank you to Andrew Hadaway for his work on the main menu and cutscene artwork, it is stunning and helps push the visual quality of the project with some professionally done artwork.

Another sincere thank you to thank Matthew Layet for his work on the music of the game, it provides exactly the kind of atmosphere I hoped it would.

I would also like to thank my friends and family for their love and kindness to me through several tough patches in the timeline of this project. I couldn't have done it without you.

And finally, I would like to thank the Unity community for their commitment to providing resources to beginners to myself, and providing useful tutorials and packages for free. Special thanks to:

- ChompyLunchBox and Polygonal mind for their free 3D asset packages, which were used extensively to make the 3D aspect of the game more visually appealing.
- Scott Steffes for his Vertex Text Animation demo.

Table of Contents

1 Introduction **1**
- 1.1 Learning Haskell Through Interesting Media . . . 1
- 1.2 Summary of Contributions . . . 2
- 1.3 Summary of Results . . . 2

2 Background **4**
- 2.1 The Importance of Enjoyment in Learning Programming Languages . 4
  - 2.1.1 Enjoyment and Learning . . . 4
  - 2.1.2 Programming Languages Are Hard to Learn . . . 5
  - 2.1.3 Expectancy-Value Theory - Why Difficulty Matters . . . 5
  - 2.1.4 Haskell is Hard to Learn . . . 6
- 2.2 Serious Games as a Solution . . . 7
  - 2.2.1 Enjoyment and video games . . . 7
  - 2.2.2 An Exploration into Existing Literature . . . 8
  - 2.2.3 Learning through Video games: The impacts of competition and cooperation . . . 10

3 Design and Construction **13**
- 3.1 Tools . . . 14
  - 3.1.1 Unity . . . 14
  - 3.1.2 GHCi . . . 14
- 3.2 Content . . . 14
- 3.3 Genre . . . 15
- 3.4 The Learner's Journey . . . 16
  - 3.4.1 Introduction - The First Puzzle (puzzle 1) . . . 16
  - 3.4.2 A Little Further on - List Comprehensions (puzzle 3) . . . 17
  - 3.4.3 A Soft Wall - Multiple Generators (puzzle 6) . . . 17
  - 3.4.4 A Substantial Increase in Difficulty (puzzle 8) . . . 17
- 3.5 Evaluating Answers . . . 19
  - 3.5.1 Options . . . 19
  - 3.5.2 Implementation . . . 20
  - 3.5.3 Checking Correctness . . . 23
- 3.6 Menus and Accessibility . . . 23

4 User Testing **27**
- 4.1 Investigation Goals . . . 27

4.2 Gathering Participants 29
4.3 Enjoyment and Engagement of Participants 29
4.4 Participant Learning and Analysis 31
4.5 Participant Ideas for Further Improvement 35

5 Conclusion **36**
5.1 Lessons Learned 36
5.2 Further Scope and Improvements 36

Bibliography **38**

A Participant Consent Form **42**

B Participant Information Sheet **44**

C Puzzle Questions and Answers **49**

D User Feedback Quiz **55**

E Haskell Quiz Marking Scheme **62**

Chapter 1

Introduction

1.1 Learning Haskell Through Interesting Media

Haskell is a powerful functional programming language used both in academia and industry [36]. At the University of Edinburgh, it is notably the first programming language taught to students pursuing a degree in Informatics. Through these observations one could conclude that it is a worthy language to be learned for anybody studying in the field of computer science.

That being said, Haskell is also not an easy language to learn, and while this metric is mostly subjective, it is listed as being one of the harder to learn languages according to several authors [10][1]. Making Haskell easier to learn through the use of specialised learning tools seems like a worthy goal.

While many such tools exist already in written format - for instance the books: Learn You a Haskell For Great Good [17] and Two Dozen Short Lessons in Haskell [21] both take novel approaches to make learning from the book interesting - they don't present truly interactive options for learning. This is where HaskellQuest extends this idea. Instead of a book, this project aims to create an interactive and engaging learning tool enabling easier learning of a difficult language. Like the aforementioned books, this project aims to be most effective for people with limited or no functional programming experience, but are familiar with some common imperative programming concepts such as types and functions.

In addition, this project aims to test the effectiveness of HaskellQuest (and by extension games like it) at teaching Haskell and contains a section involving feedback from 15 anonymous testers as well as analysis of the implications of that feedback. This is briefly summarised in the section alongside a summary of contributions made during this project.

1.2 Summary of Contributions

HaskellQuest is a large project created from scratch inside the Unity Editor. The list of contributions by the project author include:

- Creating a reliable interface for the Unity Game Engine to interact with an multi-threaded instance of GHCi.
- Creating a system for easily creating and registering puzzles inside the Unity Editor.
- Creating the logic for a multilayered game that allows for gameplay in both a 3D and a 2D environment.
- Designing a post-test research questionnaire to judge the effectiveness of HaskellQuest as a learning tool through user feedback and conducting ethical research using it.
- Creating almost all of the 2D artwork displayed during gameplay from scratch (not including cutscene artwork by Andrew Hadaway).
- Creating several 3D models using Blendr for use inside the 3D portion of the game, that were composed by hand.
- Writing in-character dialogue for prompting the player about Haskell concepts and syntax while keeping them immersed within the game world.
- Writing puzzles to teach the player about Haskell syntax and concepts by exploring them through problems.

Alongside the physical contributions, much time was spent thinking about what sections of Haskell to teach. List comprehensions were settled on because of their easy to read syntax, They make a good starting point to build additional knowledge about Haskell upon later. Lists in Haskell are also a fundamental part of the language, and working with them was judged to be necessary for any student learning the language.

1.3 Summary of Results

HaskellQuest is successful at providing an interactive and engaging game, however much more data would be required to measure the efficacy of its teaching. While participants completely new to programming were able to learn and understand some concepts and syntax that HaskellQuest introduces to them, the learning curve of the game was considered to be too steep by several participants, meaning the challenge of the game is too difficult.

Despite this, almost all study participants were able to learn something and commented positively on the UI design, help menu and story of the game as helping them feel engaged with the material.

While the current form of the game is likely too immediately difficult, this can certainly be rectified with further work and testing in the future. This project provides a good foundation on which to build to understand the place that serious games should have in education systems in the future.

Chapter 2

Background

2.1 The Importance of Enjoyment in Learning Programming Languages

2.1.1 Enjoyment and Learning

One of the most important and historically overlooked aspects of education and learning is the role that enjoyment plays in the students learning of any subject [11]]. Two of the great learning theorists, Piaget and Vygotsky, emphasised the importance of play in promoting learning and in supporting cognitive childhood development. Play in this regard could be defined as any activity that an individual engages in for personal enjoyment or fulfilment. Following on from this, a number of studies have shown that students who enjoy their time learning are much more successful than students who do not enjoy their learning [11][13]. It could be argued that the relationship between enjoyment and learning is explained by backward causality i.e. that a student who is more successful is more likely to enjoy their learning. In fact, instead studies have shown that enjoyment is a strong predictor of success even amongst otherwise motivated students [23]. These findings suggest that the relationship between learning and enjoyment is better explained by overall enjoyment being an important factor in student success rather than successful students enjoying more time learning.

It is important to acknowledge, however, that students do also derive enjoyment from success and in fact their positive experience of successful learning can augment their attitude and approach to learning more generally. The old adage "success breed success" is well recognised in business ventures and seems to apply to learning environments also [12].

Interestingly, students who enjoyed the learning process tend to place higher value on the information they learned. The way a student views their education and the way a student views themselves can be an important factor in learning success and will be explored in more detail when looking at Expectancy–Value Theory in section 2.1.3.

Students who attempt to learn a difficult subject and who are not enjoying or engaged in the learning material are much less likely to be successful in learning that material overall [11].

2.1.2 Programming Languages Are Hard to Learn

Learning a programming language, especially for the first time, is difficult. Programming languages can be unwieldy, their rules may be obtuse and difficult to intuit. They are, in the words of Amy Ko, "The least usable, but most powerful human-computer interfaces ever invented." [16]. Haskell, as a purely functional programming language, is part of a paradigm that, while it is becoming more commonly used, still has a lot of ground to cover before approaching anywhere near the popularity and commonality of imperative programming [2]. Because of the niche that functional programming generally fills in industry, the vast majority of students who enter the informatics world will likely have little or no experience with Haskell or languages like it. Most independent sources suggest learning an imperative language as a "first" language [3][24] and while secondary schools are free to teach whichever programming language they see as fit, UK national qualifications curricula such as those of the Assessment and Qualifications Alliance (AQA) or Eduqas GCSE Computing do not cover functional programming languages at all [31].

Most students will be familiar with imperative programming, so when learning Haskell, in addition to learning a completely new language, they also have to come to terms with the fact that Haskell is a functional programming language with its particular distinctive approach and rules which add to and increase the difficulty of the learning task.

2.1.3 Expectancy-Value Theory - Why Difficulty Matters

One might be inclined to ask "why does difficulty matter to a student's learning outside of the need for a student to apply themselves?" The answer to this can be explained in part by Expectancy- Value theory, a theory that explores underlying motivational factors that affect outcome and is used in both education and economics [5].

According to the Expectancy-Value theory [5][25] when applied in an education environment, a student's success in learning or achievement in education is determined by two main types of factor: Expectancies and Subjective Task Values (STVs). Expectancies are factors that relate to the student and describe an individual students beliefs about how likely they are to succeed in a task, for example, what they believe they can accomplish, how intelligent they think they are and how hard working they believe themselves to be. STVs, on the other hand can best be described as constructs relating to the specifics

of the task and reflect the question. “Do I want to do this activity and why?” [8]. This project draws more from and builds upon consideration of motivating elements of STVs rather than Expectancies in its design and approach. STVs and Expectancies are thought to interact with one another to shape and influence important learning outcomes such as engagement of the student with the learning material, academic achievement and the student’s ongoing interest in the subject matter.

Subjective Task Values can further be separated into Attainment Value of the task (which are important for the student’s identity or sense of self), Intrinsic Value of the task (reflecting students enjoyment or interest), Utility Value of the task (how useful or relevant the learning is to the student) and Cost of the task (how costly the learning is in terms of time, effort, loss of valued alternatives, or negative psychological experiences such as stress) [5].

Considering motivational factors in this way when designing teaching materials may be becoming increasingly important. With the increasing amount of online teaching in very recent years, a growing number of students are struggling to stay motivated [18]. There are potentially many reasons for this which are beyond the scope of this discussion but this concerning development in the educational landscape serves to highlight the important role of well-designed and considered teaching materials in course work and for motivating factors in ongoing student engagement.

2.1.4 Haskell is Hard to Learn

Learning a functional programming language such as Haskell is generally high on cost for the reasons outlined earlier (its approach and rules can be difficult to adapt to and students largely lack previous experience or exposure to functional languages). In terms of STVs then, learning Haskell could represent a high Cost to the student. In addition, while Haskell provides a vital role within its niche area of activity, many subsequent jobs a student may be involved in will not involve functional programming in any way other than potentially dabbling in python [27] and therefore, for many students Haskell might not rate particularly highly on perceived STV Utility Value. The combination of these factors might lead to a perception that learning Haskell will not provide particular future benefit for the student despite the fact that learning Haskell provides a number of indirect benefits. These include firstly, broadening the students view of what Informatics is and thereby allowing potential for future growth and innovation in the discipline; secondly providing the student an opportunity to learner a stricter language with safe assumptions, thus supporting growth in their precision when programming and thirdly further advancing student experience with notation, both mathematical and computational.

In a perfect world lecturers would endeavour to try to ensure that the Intrinsic Value and

Utility Value of learning functional programming should be as high as possible in order to generate interest in the subject and support Attainment Value in students learning of the subject. This would benefit both the unmotivated student who is unsure of the value of learning Haskell and concerned about the Costs involved in its learning, and the already motivated student, further reinforcing their curiosity and drive. Ideally this approach could provide a strong baseline to support either group of students towards learning success.

The natural question following on from these considerations then is "How?" How can teaching materials incorporate elements from the research to optimise student experience and support learning success.

2.2 Serious Games as a Solution

2.2.1 Enjoyment and video games

A serious game is at its core, a game that is useful to its player as more than just a relaxation tool, it has both "serious" and "game" parts to it [7]. While the "serious" section can refer to matters other than learning for educational purposes (for example, the modification of the game Half-Life "Escape from Woomera"[4] raises awareness of living conditions inside an Australian Immigration camp) for the purposes of this paper a "Serious Game" will only refer to their use in an education context. The idea for this comes from the discussion earlier – simply that enjoyment while learning, correlates positively with effort and focus on a task [22] and that effort and focus are essential to engagement – a vital component in student success [14].

While games might seem like an odd consideration when thinking about increasing enjoyment and engagement with learning material, there is a large body of strong evidence showing that children learn extremely effectively through play [40].While the literature surrounding adults learning is less extensive, the evidence that does exist is supportive of using games as a way of teaching adults new subjects [39].

Games, – specifically video games – are an extremely popular method of relaxation and leisure time for a lot of people today [28]. While playing a game, the player engages with what is displayed on screen to perform various activities including information analysis and strategy, problem solving, performing tasks of mechanical skill and engaging cooperatively or competitively with other players. The player does what their designated common noun's etymology suggests, they play.

This observation about the existence and attractiveness of video games to people was succinctly put as "The purpose of video games ... is to enjoy them" [19]. Put together

with the idea that play can be a strong medium to successfully teaching topics, it is easy enough to conclude that the marriage of video games and education should be – on paper – a match made in heaven.

There already exists a large amount of literature suggesting that the "Gamification of Learning" or "Serious Games" can be beneficial for learners either on their own or when combined with more traditional learning methods [38], however this is not always the case. In an article by Erickson and Sammons-Lohse published in 2021 [9], a game was found to be able to keep up with an experienced lecture in rating engagement of learning participants and generating a positive attitude towards the material - but not in post test results. This runs contrary to much currently existing literature, although the quality and rigor of that literature could be considered subpar [20]. Research into serious games as a way to teach adults tends to be sparse.

2.2.2 An Exploration into Existing Literature

This section discusses two articles in more depth to explore some of the approaches taken by researchers when exploring the development of serious games as learning tools in greater detail, paying particular attention to how they might highlight some of the finer points that seem to influence the ability of serious games to teach students. The two articles come to different conclusions, one is enthusiastic about the effectiveness of a serious game in teaching students, the other much more reserved in their conclusions. Given the increasing evidence base on the potential usefulness of serious games as teaching aids, it seems to be the case that greater understanding is needed on the contents of successful games and the relationship between game content/ design and student population rather than studies adopting more general conclusions regarding serious games as a teaching tool less specifically.

2.2.2.1 Mobile serious games: Effects on students' understanding of programming concepts and attitudes towards information technology

Mobile serious games: Effects on students' understanding of programming concepts and attitudes towards information technology [38] is an article on research performed in Turkey investigating the effects of serious games on the education of students around 11-12 years of age. The article was specifically looking at both the performance of students and their attitude towards the subject matter when their learning was reinforced using a serious game delivered through a mobile device instead of the standard instructor-led classroom learning.

The game Lightbot [15] was used to teach 21 students in their "Information Technology and Software" time during school, with 15 students in the same grade acting as a control group. Lightbot was used as a reinforcement tool - being used as a way for students to

visualise and interact with new concepts taught by the instructor instead of the game introducing new concepts to the student.

The authors selected the game Lightbot under the following conditions:

- ”The game should be easy to play”
- ”The game’s languages should be easy to understand”
- ”The game should be free, since it will be played at school”

These factors - particularly the first two, highlight the importance of various principles guiding serious game design. The importance of a successful serious game being accessible to ts user base is clear. A low barrier to entry is considered a must for games where the focus is on learning a subject.

The experiment used a pretest-post-test design to collect data about the performance of students before they were taught the material by the instructor, and then again four weeks later after material had been reinforced, either by Lightbot in the case of the experimental group, or by the instructor in the case of the control group.

Both groups performed similarly in their pretest scores for both categories. The study found, however, that while the student attitude towards the course was not significantly different between the two groups, the difference in performance between the control group and the experimental group were significantly different - students who had their learning reinforced by the serious game performed *much* better in post-test performance than their peers.

While this study supports the potential power of serious games as teaching aids, even with its positive results its findings are limited in what can be concluded more generally about serious games due to the studies design and small scale. It does however demonstrate that in this specific example students could perform better by being coached by a serious game than by more traditional methods. Although the authors of the article were very optimistic about the results of the experiment and what they believe it demonstrates about the scope of serious games in general, this maybe is somewhat exaggerated given some of the drawbacks of the study design. Limitations of the design include its small scale, lack of randomisation and, although the experimental and control groups help equal academic ability, limited information was provided about possible confounding factors that may have influenced the results rather than the effectiveness of Lightbot. The authors, for example do not acknowledge the possibility of elements such as novelty effects influencing motivation for those students in the experimental group - particular since the mobile devices were rented specifically for the experiment and are not available to the students otherwise.

While this is an interesting article on the power of serious games in certain circumstances and it adds to the growing pile of evidence supporting the use of Serious Games

as a teaching aide, it assumes a lot in its conclusion and blows out of proportion the results found.

2.2.3 Learning through Video games: The impacts of competition and cooperation

Learning through video games: The impacts of competition and cooperation [19] on the other hand, comes to the opposite conclusion about its findings when comparing the effectiveness of serious games compared to traditional learning methods when teaching students. This article's idea was to investigate four different teaching methods in an adult demographic.

- A control group - using a traditional slideshow with an instructor.
- Experimental group 1 - using a serious game played individually.
- Experimental group 2 - using a serious game played cooperatively with other players.
- Experimental group 3 - using a serious game played competitively with other players.

While the paper more specifically investigates the effects of cooperative and competitive play in serious games, their overall investigation is into adult learning through play and the study has a much larger number of participants (180 in total) which enables conclusions to be drawn about adult learning outcomes and serious games in general in addition to more specific investigations into the impact of individual versus cooperative or competitive play. Unlike Yallihep and Kutlu's study design, this study more closely relates to the questions this project aims to explore.

This study aims to "explore some of the main educational impacts of cooperation and competition as mechanisms for social engagement and their added value to the transformational play-based educational video game experience." Here transformational play is defined as the process of immersing the learner inside a story and allowing them to interact with the said story. HaskellQuest adopts a similar approach, introducing a character and story line to the learner who then encounters learning material through activities and interactions of the play-based character.

Erikson and Sammons-Lohse's study set out to answer three main questions: whether there was a significant difference, firstly in performance, secondly in engagement and thirdly in attitude between the four participant test groups described above. Importantly the study assumed that there was no significant difference between the four groups prior to the study. This could be considered a weakness of the design, although admittedly one that HaskellQuest shares. The researchers used a post–test design to collect data

and set out to answer their three research questions using the data gathered.

The chosen game in the study was Night of the Living Debt [17] which is a game designed to teach fiscal responsibility and to educate the player about potential dangers such as "payday lenders" and other such financial traps. In the game the players' credit score is important and players' decisions lead to meaningful gameplay consequences on their credit score. These consequences attempt to mirror consequences that similar actions would have in reality.

Participants in the control group received a 40 minute lecture during which time they were also allowed to interact with the lecturer and ask questions. Game players in the experimental group were allocated roughly the same amount of time as the control group and were instructed to either play individually, cooperatively or competitively. After the allotted time had passed all groups were asked to complete a post-test, answering questions specifically designed to explore the three research areas relating to performance, attitude and engagement.

Interestingly the study found that the control group who received the lecture showed no significant difference in their engagement or attitude scores to the learning material, however a statistically significant increase was found when compared against the performance scores of the experimental groups playing the serious game. These findings run contrary to the findings of many other studies in published literature. The authors state that there are many possible reasons for these results. The lecturer, for example, taking the control group, was considered to be a content expert and was able to field questions relating to specific individual student questions on the topic extremely well, something that the serious game lacked the flexibility to do. The paper highlights the fact that in this particular study design the control group could "freely interact with a content expert and experienced lecturer on a topic of interest" where the serious game groups could not. Interestingly, even though there were no significant differences in attitude towards and engagement with teaching material between the game players and the control group, the performance of the control group in this study was far higher.

There are multiple ways to interpret this study's findings and any conclusions it might contribute to the use of serious games as an educational tool. Amongst other things, it highlights the fact that there is still much to learn in terms of using serious games as teaching tools and how best to do that. Taken with Yallihep and Kutlu's findings, it might be the case that serious games, while not necessarily better at teaching the student first time, may be strong reinforcement tools that allow the student to keep engaging with the subject at the same level as with an experienced lecturer. This would have the advantage that it would not require that lecturers time beyond the initial teaching or game material development. If this were the case serious games would have important contributions to make in the educational landscape. Additionally it might draw attention to the need for serious games to be designed so that they have enough flexibility to match well and pitch the educational material to the specific and individual requirements of the student. One

of the important contributions that Vygotsky made to scientific understanding of how individuals learn was to emphasise the need for scaffolding and awareness of the zone of proximal development (ZPD) [6]. Teaching is thought to be most beneficial when it is targeted at the ZPD which describes the zone which is just outside of a learner's knowledge and experience but close enough to be able to be mastered with guidance and teaching. To learn we need to be presented with tasks just outside of our ability range (but not too far) with scaffolding, or supportive activities provided by a teacher, aimed at supporting the student as they grow through the ZPD. As Haskellquest aims to be predominately for students with some imperative programming experience but little or no functional language experience, the hope is that the content chosen is familiar enough to the player that is within their zone of proximal development. Ideally the player would have enough experience with imperative programming that Haskellquestt may partially act as a reinforcement of familiar concepts, before reaching into the unknown and the game content scaffold the growth of skills in the use of Haskell.

Chapter 3

Design and Construction

The central objective of the project was to produce a video game that covered aspects of Haskell concepts and syntax that could be learned through playing it, while maintaining user engagement through a "whodunit" style mystery and an iconic aesthetic. Deciding on a design was a multi step process.

- Firstly, deciding on content to cover. Haskell is a very large language with many core concepts to cover. It was decided that in order to limit the scope of the game to something achievable within the time given, the content would have to be limited to a small subsection of the language.
- Secondly, the genre of the game had to be decided. This was an integral part of how the Haskell content was delivered in an engaging way, as well as providing the scaffolding of how to immerse the learner in the setting.
- Heavily reliant on the previous steps for direction, puzzles presenting problems to the player had to be written and designed to progress the learner through the content, providing enough of a challenge while showing different syntax elements and interaction in Haskell. In practice this was difficult to balance, particularly with the amount of content that the game was proposed to cover.
- Fourthly, the learners had to have their answers evaluated in some manner to provide feedback to them while they made their way through the puzzles. This was the most technically complex area of the project and was where a lot of the core difficulties in implementation were faced.
- Finally, decisions had to be made about any extra information that would need to be provided for learners in addition to how that information should be laid out, as well as decisions and implementation of accessibility features.

These points are covered in sections 3.2-3.6 below, following section 3.1 on relevant tools to the project.

Overall, the creation of a game is a multidisciplinary field, involving many different skills including writing, digital art, software design and programming. All assets bar the ones credited to others were created from scratch in order to better fit the atmosphere

and themes of the story being told through the game medium.

3.1 Tools

The primary tool used to ease the construction of a game was the Unity Editor. This was used for creating the final running game, and was used for the primary game loop, the graphics engine and the game object interaction. Most scripts were written using C# inside visual studio and a mixture of Paint3D, GIMP and Blendr were used to create the majority of 2D assets and some 3D assets too. Most of this work could be considered "routine", but almost all of the parts in the final game were created from scratch. Parts that were challenging - are detailed further in this chapter.

3.1.1 Unity

Unity is a free scene editor packaged with a physics and game engine and has been used in a professional environment to create several award winning games such as Subnautica [35], Outer Wilds [33], Hollow Knight [32] and Overcooked [34].

It allows for painless integration of multiple assets and scripts to create a working game experience, as well as coming with its own physics and real-time rendering engine so that the focus of the project could be on the game design and pedagogical aspects rather than the technical implementation of a game from scratch.

3.1.2 GHCi

GHCi - short for Glasgow Haskell Compiler interactive - is a shell program that runs an interactive Haskell environment. This is very similar to the python interpreter. This was incredibly useful for debugging any problems relating to puzzles in Haskell and became an integral component of evaluating answers provided by the learner later on in the game's construction.

3.2 Content

HaskellQuest is aimed at beginners to Haskell and to a lesser extent, beginners to programming. It teaches the basic syntax of Haskell types and lists, familiarising the learner with variables and functions as concepts (for a full list of syntax introduced by HaskellQuest see Figure 3.1). This decision to focus on beginners was made in order to be able to more effectively test the effectiveness of engagement in learning. The idea was that people who have had less exposure to programming and Haskell would provide data that is easier to analyse - someone who is already familiar with Haskell would know many of the concepts demonstrated and trivialise a lot of the challenges that a beginner learning Haskell for the first time would face.

Syntax	Specific	Haskell Examples
Types	`Integer`	`var::Integer, var::Int`
	`Double`	`var::Double`
	`Bool`	`var::Bool`
	`Char`	`var::Char`
	`String`	`var::String, var::[Char]`
	`List`	`var::[a]`
	`Tuple`	`var::(a,b), var::(a,b,c,d)`
List Comprehension	Output	`[2 * a | ... ]`
	Generator	`[ ... | a <- as, ... ]`
	Guards	`[ ... | ..., a == 1, ... ]`
Pattern Matching	List Items	`a:as, a:b:cs, [], a:[], a:_`
Infix Functions	`+, -, /, *`	`2 + 2 == 2 * 2`
	`==, /=, <, >, <=, >=`	`3.4 > 2.0, 200 <= 200`
	`++`	`("as" ++ "bs") == "asbs"`
Functions	`head, tail, init, last`	`(head [1, 2]) == 1`
	`length`	`(length [1, 2, 3]) == 3`
	`zip`	`(zip [1, 2] "ab") == [(1, a), (2, b)]`

Figure 3.1: HaskellQuest Syntax List

While it would be possible to create alternate problems for more experienced survey participants, this idea was rejected due to the difficulty in assessing experienced participants' knowledge bases, as well as the additional time that creating such problems would take.

3.3 Genre

The chosen genre of HaskellQuest was a puzzle game. Puzzle games vary wildly, however the genre is generally well received amongst people otherwise unfamiliar with video games - the Candy Crush Saga for instance, is one of the most played games in the world and has a very varied demographic, with its typical player being in their mid 30s [29].

After deciding on the form that the game would take, a further decision on aesthetic and atmosphere was made in order to immerse the learner in the game environment [26]. The game has a "film noir" theme - partly this was chosen because of my own tastes at the time, however it is also a stunning and recognisable film movement which is strongly associated with mystery and intrigue - perfect for a game demanding its players solve mysteries.

3.4 The Learner's Journey

The mysteries in HaskellQuest take the form of puzzles that are displayed through a puzzle screen. The learner is encouraged to seek out these puzzles inside a 3D environment, this is by itself a mini-game, demanding the players pay some attention to their environment to seek out puzzles to solve. Once the player finds a puzzle, they are taken into the puzzle screen (Figure 3.2) and introduced to the concepts needed in the puzzle via animated dialogue in game. The dialogue introduces all the intricacies of the puzzle menu and different Haskell concepts.

The puzzle screen contains several elements to make playing around with the data the player is given more intuitive. The Data Element (Figure 3.3) provides information on a variable that is held within memory and describes the typing of the variable and the contents of it (if it is a list or tuple) in additional detail - these are held in the Data Panel. There is also a Help screen (Figure 3.4) that the player is pointed to if they need more information. This contains further in-depth information on all topics that the game teaches, but disables topics that the player has yet to be introduced to through dialogue, in order to prevent confusion.

These displays exist to reduce the "cost" of completing the puzzle, the information that is needed to work out and solve the puzzle is readily available within a few clicks - so the player need not hold that information all in their head, they can simply focus on the method of how to solve the puzzle.

The game contains 10 puzzles in total - 7 "tutorial" puzzles, introducing syntax through dialogue and puzzles and 3 "mystery" puzzles, where the player uncovers the tracks of the murderer. While this is too much to go over all of them in detail, the following puzzles are important milestones for the player and so will be dived into to give an idea of the questions players are asked (all questions along with example solutions can be found in the appendix C).

3.4.1 Introduction - The First Puzzle (puzzle 1)

The first puzzle involves a large amount of dialogue introducing how Haskell handles variables and types to the player. To gently introduce the player to the concept, an example of a list of Strings is given and the player is asked to return a list containing the strings "hello" and "world".
As expected the player's answer must be of the following form:

```
["hello", "world"]
```

3.4.2 A Little Further on - List Comprehensions (puzzle 3)

The next milestone occurs after the introduction of list comprehensions, Asking the player to add the first element of a list of integers to every element of a list of doubles. Like the previous question, the player *may* decide to do this manually, either using a pen and paper, or in their head - this is aided due to the user interface design giving access to the variables in the data.
The list of doubles is only 4 elements long and so it is reasonable to complete the question like this. This potential "manual completion" is intentional and is left in in case the player needs more time to acclimatise to the notation and type system. If the player wishes to answer using a list comprehension, their expected answer would look something like this:

```
[head(list1) + a | a <- list2]
```

3.4.3 A Soft Wall - Multiple Generators (puzzle 6)

The third milestone during the learning stage of the game introduces multiple generators in a list comprehension, and is constructed in such a way to prove unintuitive for those solving puzzles manually (in fact, this seems to be where several participants in the survey portion of the project were unable to continue). The puzzle asks to - using multiple generators in a single list comprehension - append the elements of the list `[1, 14, 144]` to the lists within `[[2, 3, 4], [15, 16, 17], [145, 146, 147]]`. The player is asked to use multiple generators here, the expected result would be a list 9 elements in length, counter to the intuition that the answer would be something akin to `[[1, 2, 3, 4], [14, 15, 16, 17], [144, 145, 146, 147]]`
This puzzle is important because it allows those who may have paid a little less attention to recognise that they have missed some information and that they require it to continue, it also mentions that answer will likely be unexpected within the puzzle description.
The expected answer is in the form:

```
[a:as | a <- list1, as <- list2]
```

3.4.4 A Substantial Increase in Difficulty (puzzle 8)

By this point in the game, the player is expected to have learned all the material featured in Figure 3.1, as this is the "story" portion of the game, and the player is supposed to be solving a mystery, the difficulty substantially increases from the tutorial puzzles. The player is introduced to size 4 tuples and given several elements to filter on. The player must use multiple guards (which is directly encouraged in game), or multiple stages in order to successfully find the correct answer. This pattern of using guards over different types and elements persists throughout the rest of the questions.
The expected answer is in the form:

```
[ a | (a, b, _, d) <- knives, guard1 b, guard2 d]
```

Figure 3.2: The in-game Puzzle Screen

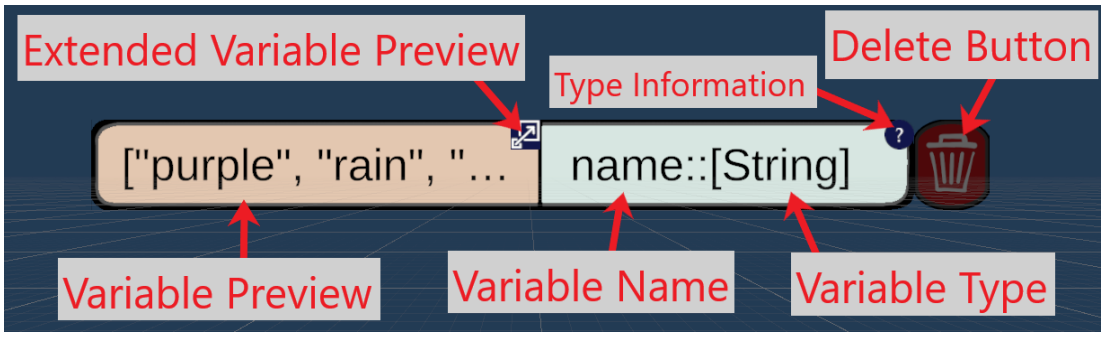

Figure 3.3: Data Element

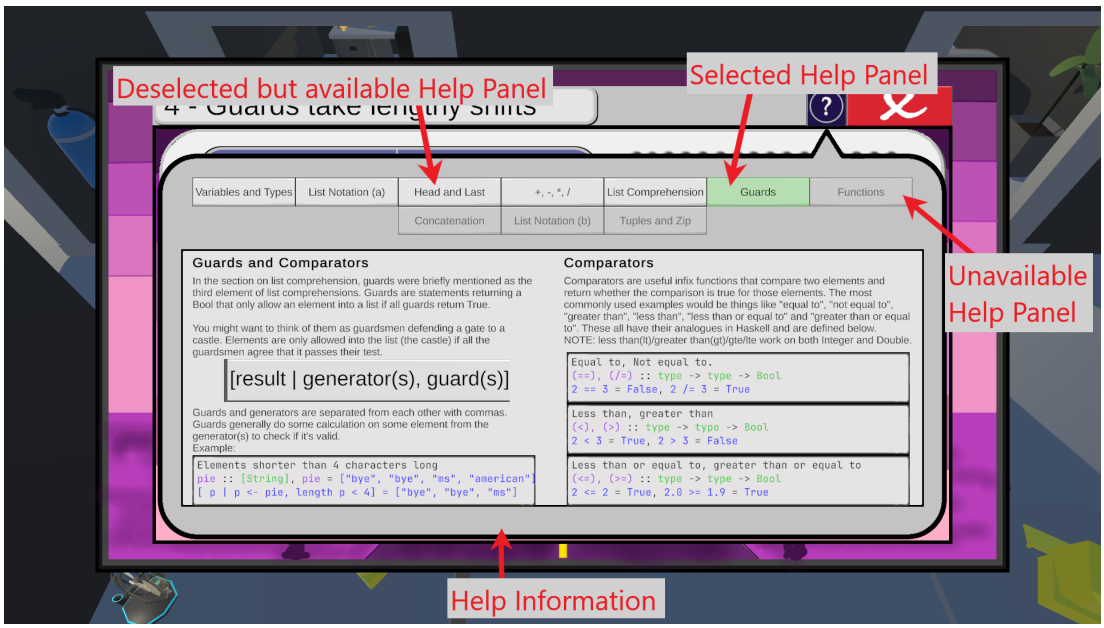

Figure 3.4: The in-game Help Screen

3.5 Evaluating Answers

3.5.1 Options

There were two primary options explored for evaluating learners' answers to the provided problems in this project.

- The first, was implementing an independent parser and interpreter for Haskell List Comprehensions. This had several advantages, it would effectively act as a DSL for the problems represented in the game, it would also allow for customised useful error messages and potential graphical integration with the game interface. This would likely be the option to choose should the game have significantly more development time because of the potential ease of integration.
- The second option was to implement a thread running GHCi that the game could communicate with. This would be a downgrade from the DSL idea - there could be no specific error analysis that might be more useful to the player and any graphical integration (e.g. highlighting errors) would not be possible. It would also mean implementing the threading completely outside of Unity's main game loop - the Unity scripting API is not thread safe and therefore is unable to be called from any thread other than the main one. However, the major advantage of using GHCi was that it was already a generalised Haskell Parser, any correct Haskell could be interpreted and run. This allowed for potentially more flexibility when it came to questions and was *much* faster to implement.

Ultimately, the GHCi approach was taken, while it was significantly more limiting in what could be done to help the user understand any errors they had made, it was the only realistic option for one person to implement in the time given.

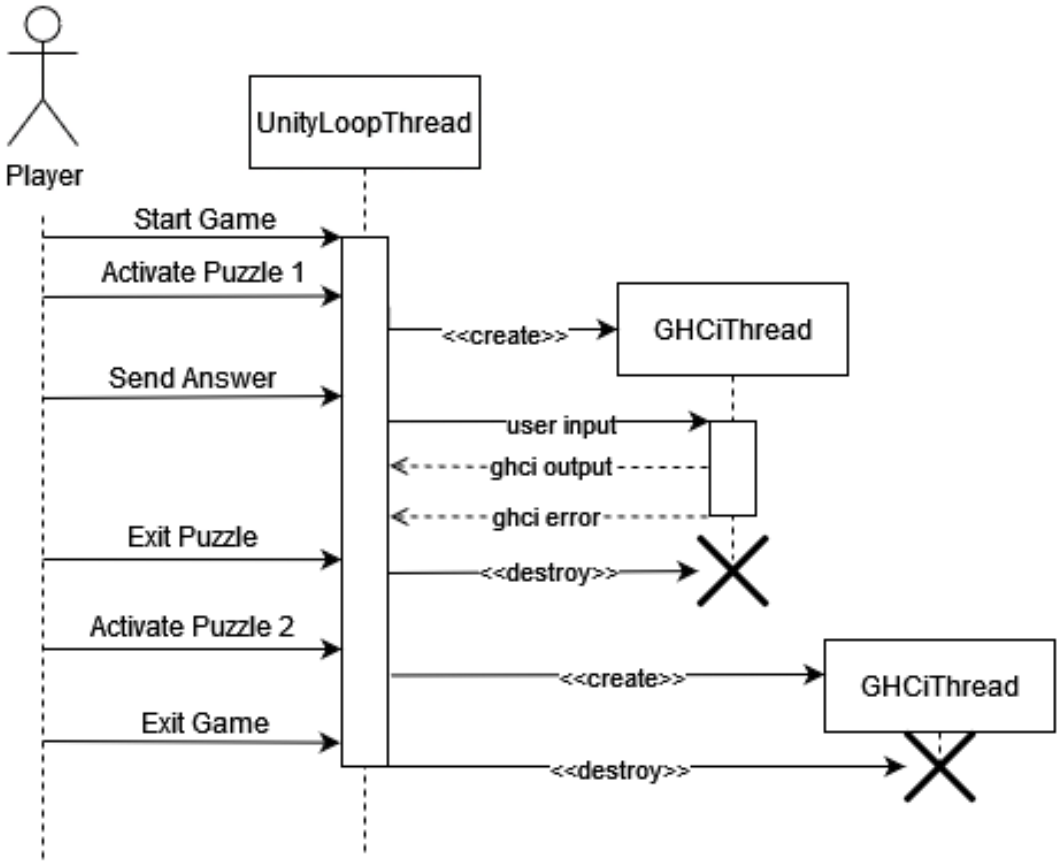

Figure 3.5: Sequence Diagram of Typical GHCi Thread usage. User input is sent to GHCi for parsing and response then processed and sent back to the game engine to be displayed to the player.

3.5.2 Implementation

Implementing a GHCi interface required starting a concurrent thread that could be controlled by the main Unity game-loop. This was implemented using C#'s System.Threading library alongside a custom thread manager that acted as the interface between the Unity game loop and created concurrent threads. A typical use case is found in Figure 3.5.

3.5.2.1 The naive implementation

At first implementing the GHCi system seemed to be fairly simple; when a player asks for input, send the input to GHCi and if it causes an error, pass that on to the player. This ended up being the first implementation of the system, using two concurrent queues, one for input and the other for output and error, the implementation simply acting as a messenger for GHCi. The Unity interface side of things was equally simple, calling the functions for adding input to the input queue and for retrieving data from the output queue.

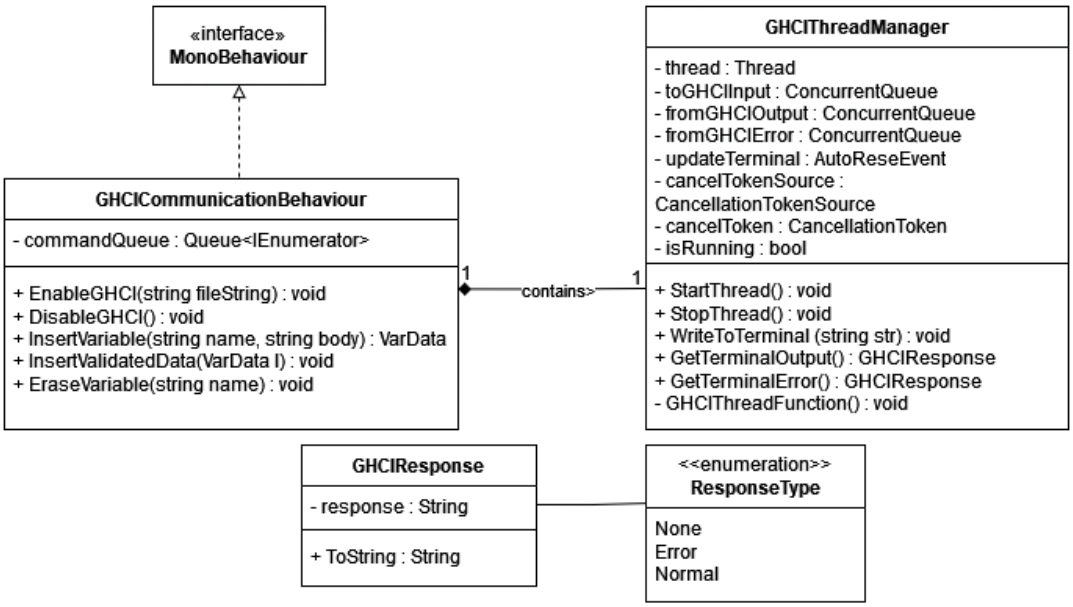

Figure 3.6: Class diagram of the Multithreading control process, MonoBehaviour is part of the Unity API

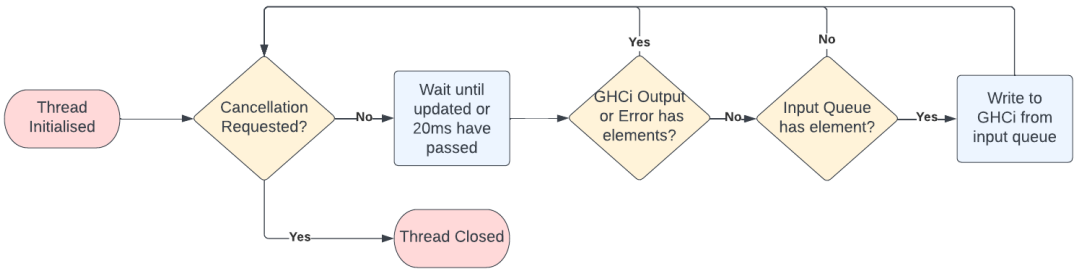

Figure 3.7: Flowchart describing functionality of the GHCi thread

However, this method came with several issues - it required the player to choose a variable name in order to save the inputted value, which while harmless on the surface, quickly runs into unintuitive error messages if they pick a reserved name or the name of a function already defined. It also did not return the result of any list comprehension, merely processing it and storing it inside GHCi as a value under that variable name until the player called on it manually. Additionally, if a particular list comprehension took longer to parse and was followed by something that took noticeably less time, any response could be muddied with multiple responses packaged together. These circumstances meant that this proposed solution was too clunky and unintuitive to act as a good interface for the player.

3.5.2.2 The improved implementation

The second iteration of the system, as described in Figures 3.6, 3.7, was very similar on the threading side of things, although instead of a single output queue, an additional error queue was added and a check to halt further input if elements exist in the output queue that had not been handled yet by the game logic.

On the Unity interface side of things, instead of simply sending raw player input to GHCi, sequences of actions were defined involving that player input. Only five sequences are callable from any game logic in Unity (see Figure 3.8):

- `EnableGHCI`
- `DisableGHCI`
- `InsertVariable`
- `InsertValidatedData`
- `EraseVariable`

These were constructed out of the following actions (see Figure 3.9)

- `SendToGHCI`
- `WriteResultToVar`
- `WriteTypeToVar`
- `WaitOnOutputThenFlush`

Actions were defined using Unity's coroutine functionality [30]. Unity's game-loop allows the addition of functions that are "run" once per loop while active - using C# generators. Coroutines yield until some condition is met, then finish and remove themselves from the gameloop. Along with a simple queue, this allowed only one action to be executed at a time, with any additional actions waiting for previous ones to finish before being called - each also includes a check to clear the queue should the thread be shut down.

This allowed for the throttling of any input to GHCi until one of these predefined actions had run its course. This had the side effect of removing overlapping problems as each action must capture 0 or 1 responses from GHCi, giving clean, easily interpreted output or error. Additionally because of the restrictions on how the player could interact with GHCi, potential undefined behaviours where the game system could break were reduced, for example: activating a multi-line sequence using GHCi's ":set +m".

This system worked well, and seemed very robust. Outside of a few potential issues: if a player decides to use something like `[a | a <- [0..], a < 10000000]`, it will render the current puzzle unplayable while it waits on the solution. Alternatively the possibility exists that a player could use some of GHCi's meta-functions to send some-

thing to the terminal that the system is not designed to handle.

These cases are both extremely unlikely to happen for the target audience of the game - infinite lists are not taught by the game, nor are these kind of lists required in any form for any of the solutions. For the second option, familiarity with GHCi's meta functions would imply that the player is already quite familiar with GHCi and likely deliberately wants to break the game, and since the game poses little security threat to ones' computer - outside of running things on the installed GHCi distribution, which the player could do anyways with how the project is distributed - this issue was not considered necessary to the project.

3.5.3 Checking Correctness

Checking the correctness of answers can be quite difficult given that there are many ways of answering a question correctly - especially in Haskell. Unfortunately the answer checking in this version of HaskellQuest is done with simple String checking of the output from GHCi. This is inflexible as an end result; while many solutions to problems can be evaluated, it means that any answer list must be in the correct order and so allows for less flexible question writing. It had the huge advantage of being extremely easy to change and play around with while implementing puzzle questions with answers.

A different mechanism such as testing the answers in GHCi directly would have been preferable for expanding the project further - the contents of a list could be checked for correctness in case the elements were the same but the order was different.

3.6 Menus and Accessibility

An important, but often overlooked aspect of a game is accessibility and options. HaskellQuest has gone through several iterations trying to make it more accessible to a wider variety of people.

The control scheme was an area where it was deemed important to have customisability. The default controls for moving about in the 3D environment are bound to the standard WASD control scheme for qwerty keyboards [37], however for anybody using a different keyboard, either due to hand strain, or a keyboard for typing in another language that doesn't follow the qwerty pattern, this makes controlling the character extremely tedious.

The default Unity control scheme was used throughout a lot of the project, however when accessibility came up it was found lacking. By default, key presses in Unity are represented as enums, and checking if a key is pressed requires checking that specific value, with no easy way to retrieve an enum from a keystroke. This makes altering control schemes using the default options difficult and as such HaskellQuest is constructed

```
GHCIThreadManager tm
bool threadShutdown

EnableGHCI(string loadfile):
    threadShutdown = false
    tm.StartThread()
    enqueue(SentToGHCI(":load HaskellScripts\" + loadfile))
    enqueue(WaitOnOutputThenFlush)

DisableGHCI():
    threadShutdown = true
    tm.StopThread()

InsertVariable(string name, string body):
    VarData v
    enqueue(SendToGHCI(name + "=" + body))
    enqueue(SendToGHCI(name))
    enqueue(WriteResultToVar(v))
    enqueue(SentToGHCI("typeOf " + name))
    enqueue(WriteTypeToVar(v)
    return l

InsertValidatedData(VarData v):
    enqueue(SendToGHCI(v.name + "=" + v.body))

EraseVariable(VarData v):
    enqueue(SendToGHCI(v.name + "= []"))
```

Figure 3.8: Pseudocode of the five sequences of actions responsible for sending data during gameplay to GHCi. VarData can be considered a container class for a representation of a Haskell variable

```
GHCIThreadManager tm
bool threadShutdown

SendToGHCI(string message):
    wait for (0) seconds
    tm.WriteToTerminal(message)
    StartNextAction()

WriteResultToVar(VarData v):
    wait while (no ghci output)
        and (no ghci error)
        and (not threadShutdown)
    GHCIResponse error = tm.GetTerminalError()
    GHCIResponse output = tm.GetTerminalOutput()
    if error.type != None:
        v.error(error)
    else if output.type != None:
        v.body = output
    StartNextAction()

WriteTypeToVar(VarData v):
    wait while (no ghci output)
        and (no ghci error)
        and (not threadShutdown)
    if error.type != None:
        v.error(error)
    else if output.type != None:
        v.type = ParseType(output)
    StartNextAction()

WaitOnOutputThenFlush():
    wait while (no ghci output)
        and (no ghci error)
    while (tm.GetTerminalError.type != None)
    while (tm.GetTerminalOutput.type != None)
```

Figure 3.9: Pseudocode of the actions composing GHCi interaction and output queue handling

with Unity's more modern non-default control scheme.

Another aspect of accessibility that was of concern was both the intensity of the lighting (relevant in the main menu) and intensity of the sound (relevant throughout the game). For people sensitive to flashes, a photosensitivity option is available in the options menu that reduces the flashing from the background lightning significantly. Additionally the volume of different sound sources can be altered independently to allow for maximum customisability to aid those with sensitive ears.

Information throughout the puzzle screens (Figure 3.2) is laid out in order to help the user understand what is going on. To take the Puzzle Screen as an example, elements are clearly delineated into separate sections by dark outlines with grey space in between them. This is in order to reduce any confusion when becoming familiar with the screen. It is in three primary parts, the Puzzle Description, containing information on how to solve the puzzle, the Data Panel, containing all the information you have available to you in the puzzle, and the Expression Bar, containing any input you want to get within the data panel. Everything that the player needs to solve the puzzle is inside the internal rounded grey rectangle, with any additional parts like the 'help' button or the 'close' button outside of this to provide minimal distraction, the player's attention is kept within the puzzle window as much as possible.

Chapter 4

User Testing

Fifteen participants responded to requests for user testing. They were asked to play HaskellQuest for a minimum of 30 minutes, and then participate in an anonymised survey asking questions about their previous experience with video games, programming and problem solving. In addition questions about what they thought of the game and a short Haskell quiz were included to assess the efficacy of the project. The full survey can be found in Appendix D.

4.1 Investigation Goals

The primary goal of the project was to create an interactive and engaging learning tool to teach Haskell concepts and syntax to individuals who had some experience with imperative language programming but little or no experience or exposure to functional programming languages. For the purposes of analysis user feedback was broken down into two main areas:

- Was HaskellQuest successful at being engaging?
- Was HaskellQuest successful in its role as a learning tool?

Both elements are considered important for this project, although the second likely more so than the first. While both could potentially be improved upon- either making the game more engaging or making it a more successful learning tool- making it a more successful learning tool is the more difficult and complicated prospect. As primarily a learning tool over anything else, HaskellQuest's ability to support learning takes priority - while nonetheless recognising the important part engagement and enjoyment play in supporting successful outcome as outlined in chapter 2.

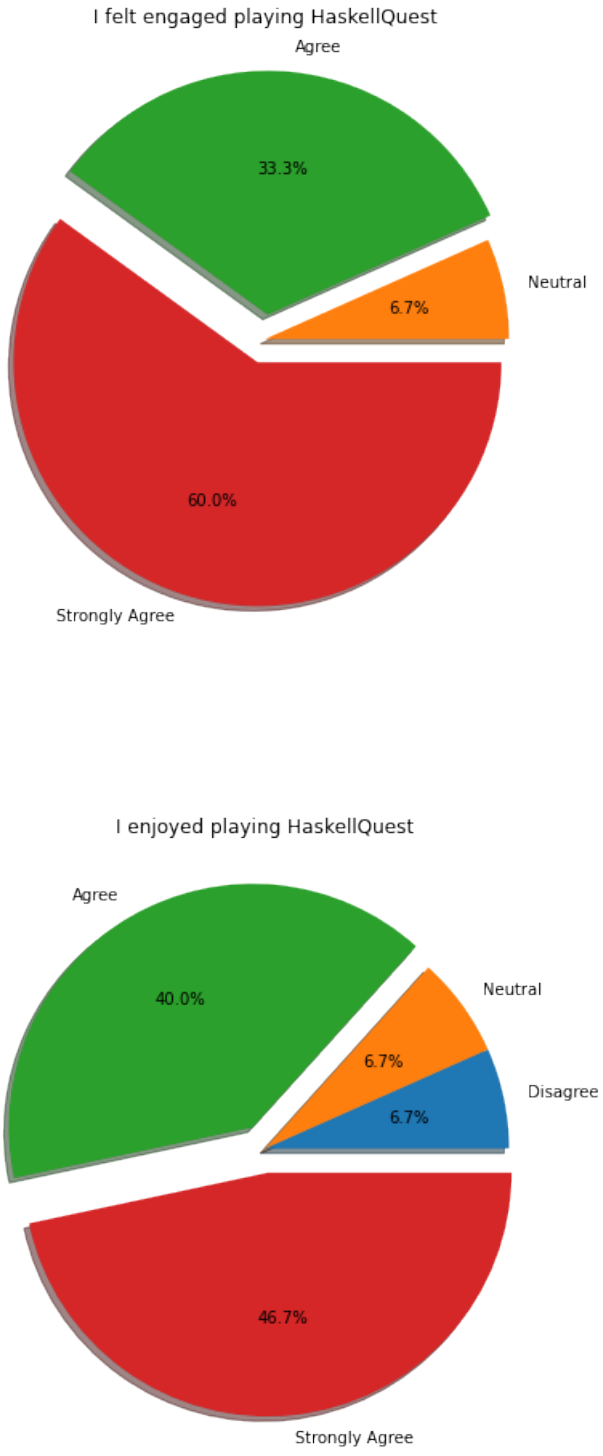

Figure 4.1: Participant Responses to Enjoyment and Engagement Questions

4.2 Gathering Participants

The most successful tool for gathering study participants was word of mouth. After distributing a request for student participants to the Informatics School at the University of Edinburgh and amongst friends there was still a severe dearth of participants. One part of trying to attract more participants involved developing a poster and presenting it during project day, with several people reporting interest. While it is difficult to know how successful this was at gathering participants due to the anonymity of the survey, several participants responded soon thereafter.

Of the fifteen volunteers five were completely new to programming and had neither experience of imperative programming nor functional programming languages at all. The remaining ten had some mix of both imperative and functional experience with only two holding less than a year's imperative programming experience - with the same or greater experience in functional programming. There were no participants that fit the target audience of HaskellQuest.

While all fifteen participants completed the questionnaire, only fourteen out of the fifteen completed some of the questions designed to test Haskell knowledge at the end. The one person who did not answer any Haskell questions had neither imperative or functional language experience and had no experience of playing video games which may have influenced their ability to engage with the subject matter. For this reason they are not included in any score related data.

4.3 Enjoyment and Engagement of Participants

From the results, almost all participants felt engaged playing the game (93%) and enjoyed their experience (86%) (Figure 4.1), this project is very successful in this regard. All participants found the mystery engaging with unanimous feedback that the story aided with their engagement with the game (Figure 4.2). This was further reinforced with several participants' additional comments stating that they enjoyed the intrigue added by the murder mystery. One participant said: "The start of the game with the visual and sound effects with the murder was intriguing and really drew you in to want to find out more".

The included music was similarly successful, again gaining unanimous positive feedback in regards to its effectiveness on engagement (Figure 4.3).

Overall, these two factors directly contributed to participant's engagement with one participant saying that "The game scenario and atmosphere made it much more enjoyable than typical educational methods and provided a little extra motivation in solving the problems."

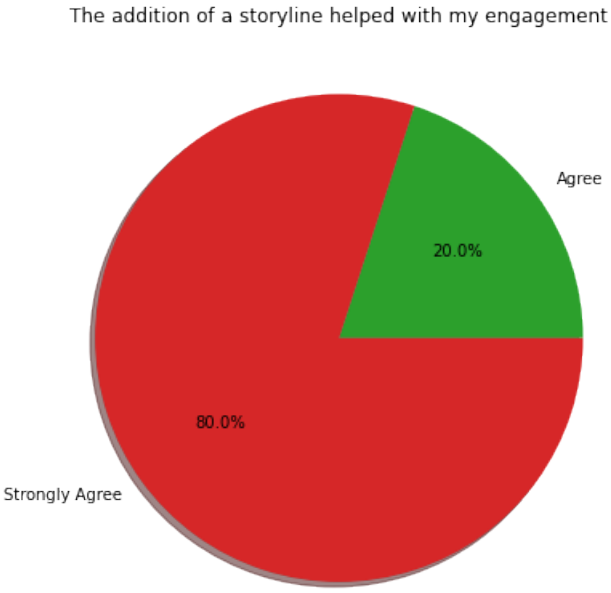

Figure 4.2: Participant Responses to Whether the Story Aided with Engagement

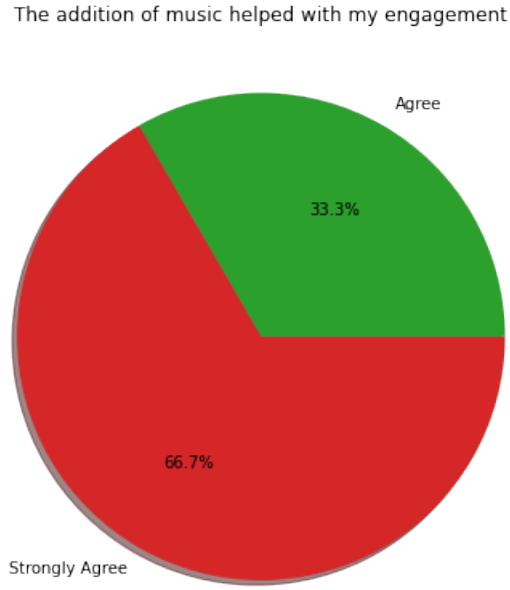

Figure 4.3: Participant Responses to Whether the Music Aided with Engagement

4.4 Participant Learning and Analysis

Participant learning was exclusively measured through the optional Haskell questions at the end of the survey. Out of the fifteen responses, fourteen participants answered at least one question, giving enough data for an observational analysis, although the sample size was determined to be too small to perform any meaningful statistical analysis.

The questions covered the `head` function, pattern matching on lists, the infix addition function, String typing and basic list comprehension Output and Generator syntax. While this does not provide a complete assessment of the materials taught (see Figure 3.1 for complete list), it covers several of the most important concepts and pieces of syntax, as well as assessing skills taught at different stages of the game.

Thirteen out of the fifteen participants agreed that the User Interface for solving the problems felt intuitive to use, with the remaining two participants feeling neutral about it - notably both these participants had limited or no experience with video games in general so this rating may have been due to adapting to an unfamiliar environment rather than a commentary on the UI. Additionally, twelve participants agreed that the help menu was useful, with the remaining three feeling neutrally about it. These results are clear successes in the game's ease of use and provide some supportive evidence that HaskellQuest as a learning tool is not difficult to learn and provides little or no additional barrier to entry beyond what is provided by the educational material itself.

Discounting the participant who did not answer any questions - participants scored on average around 67% in the final Haskell quiz, with one participant achieving a perfect score. Although there was no pretest assessment of Haskell skills, prior knowledge and exposure was assessed by asking participants how much experience they had had with either functional or imperative programming languages before taking part in HaskellQuest. Eleven out of the fifteen participants described themselves as having less than a year's functional programming experience, and five of those had none at all. Making the assumption that someone with experience in Haskell would have easily scored a full mark, the average score results of 67% suggests that many participants had little or no experience with Haskell. The score is indicative that some of the learning aims of HaskellQuest were achieved. This is a positive result.

Interestingly, neither the self-reported-enjoyment and score plot (Figure 4.4), nor the self-reported-engagement and score plot (Figure 4.5) showed particularly strong relations between the pairs of variables, although both are largely positive. This seems to go against the ongoing narrative of heavier enjoyment-result correlation that was explored in the background. However, the performance scores for participants are generally good and the engagement and enjoyment scores are positive also and it may be that the small sample size, alongside the descriptive categories for engagement and enjoyment being too broad are possibly masking possible correlations. It is difficult to know based on such a small number of participants. Additionally, it is important to recognise that the

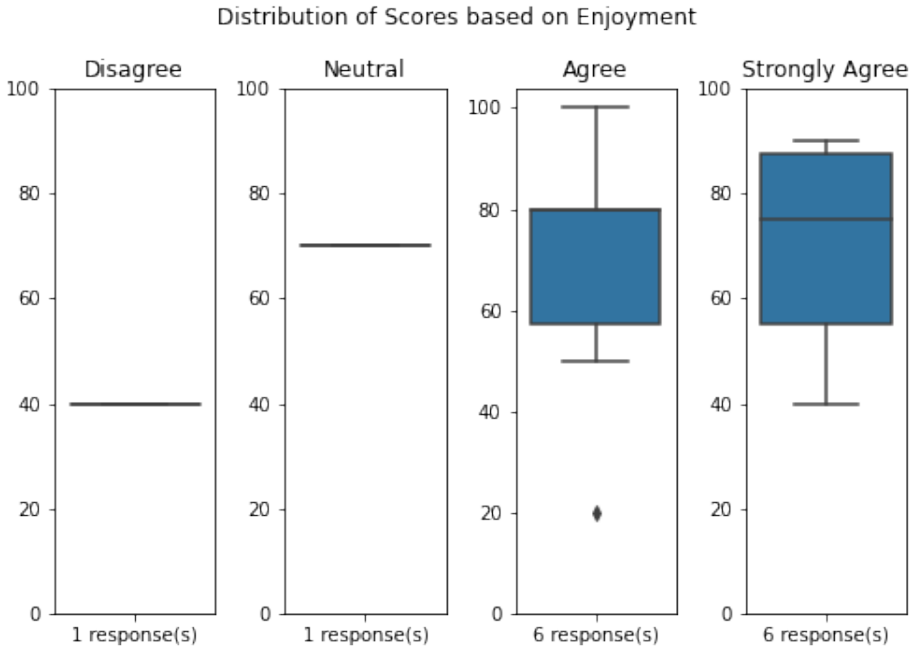

Figure 4.4: Boxplots Showing Enjoyment against Percentage Score. There is little difference between Agree and Strongly Agree.

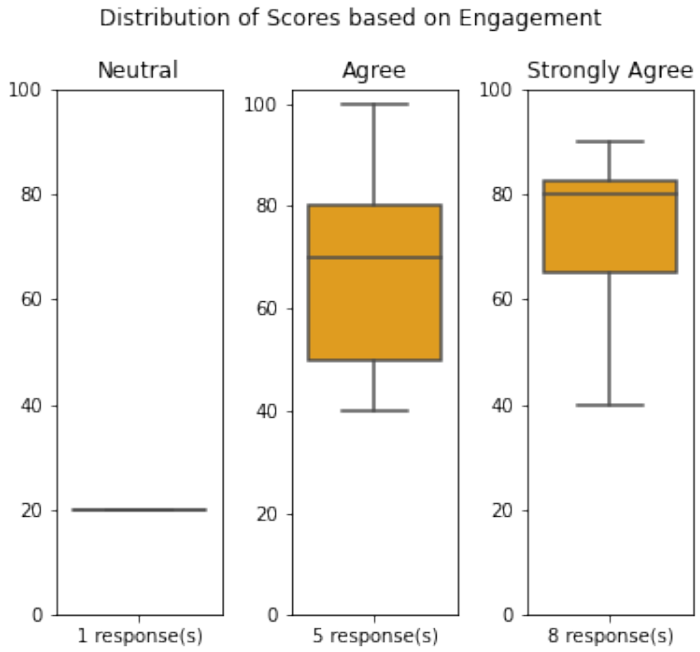

Figure 4.5: Boxplots Showing Engagement against Percentage Score. This shows a little stronger correlation than Enjoyment and Score

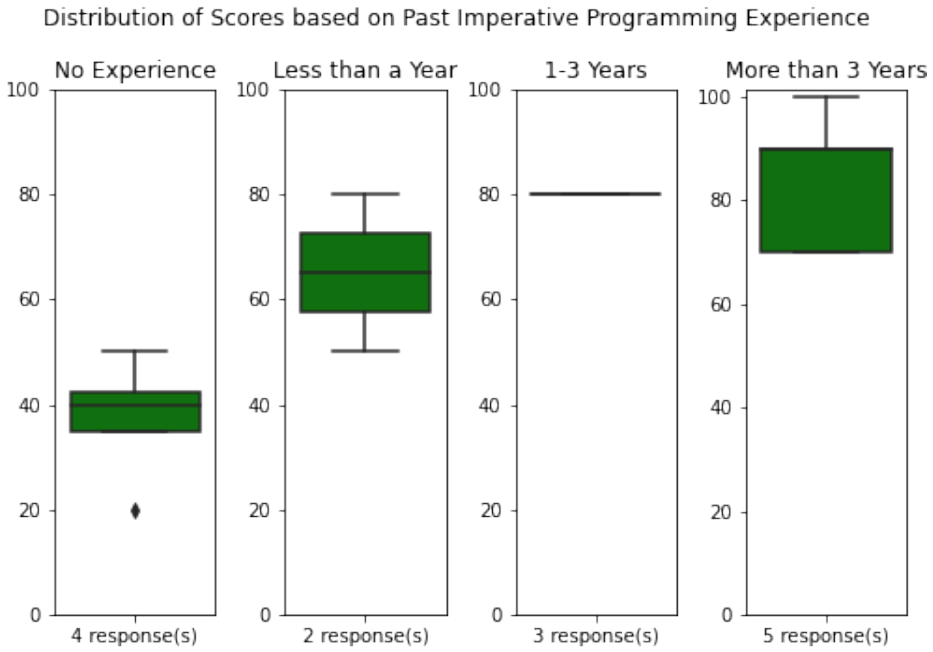

Figure 4.6: Boxplots Showing Imperative Programming Experience against Percentage Score

sample of participants who took part in the user feedback is not random. Participants are not necessarily representative of learners more generally, given that they all volunteered to take part in a study with no clear incentive other than to contribute to the dissertation of an undergraduate student, some of whom they knew. It may be that there may be bias in the self-reporting of game enjoyment and engagement or other complicating factors such as prior participant experience that might muddle what conclusions can be drawn from the data recorded.

As one might expect, looking at previous experience in comparison to score seems to correlate very strongly, both for functional and imperative experience (Figures 4.6 and 4.7). While some participants who had no experience of any kind of programming clearly managed to learn something of the concepts and syntax in Haskell, they did also score amongst the lowest of all participants. This likely reflects the content of the learning material within the game in its current form - it is likely that too much information is presented to the player too quickly for complete beginners. Those with prior experience scored more highly in their final performance scores and this is likely to reflect their different learning needs. The struggle among inexperienced players was reflected strongly in the feedback's additional commentary section, with a number of participants expressing a preference to have access to a greater number of examples/practise questions on the same topic before moving on to new learning material. This point also shows the importance of structuring educational material around the needs or the students, rather than creating a "one size fits all" solution. HaskellQuest was developed with a student group in mind, ones with some imperative programming experience but no or little functional programming experience and as a consequence

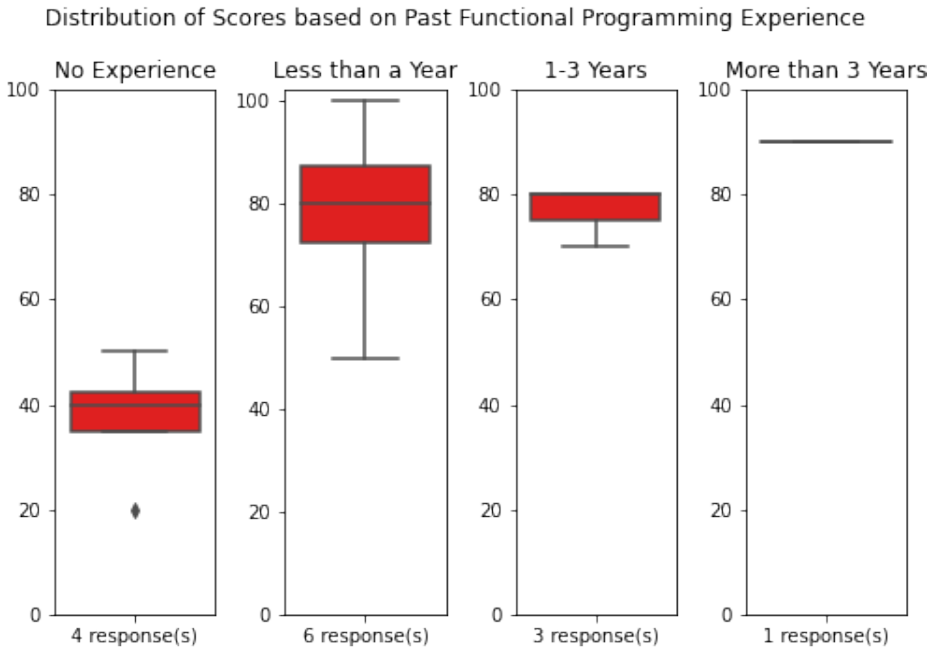

Figure 4.7: Boxplots Showing Functional Programming Experience against Percentage Score

the pacing of the presentation of teaching material is perhaps too fast for beginners. Although it should definitely be mentioned that a slower pace is a good thing for learning, and that if it is too fast for beginners it is likely too fast for near-beginners also.

Interestingly, despite their on-average lower scores, two of the four participants who participated in the quiz and were new to programming still rated the learning curve to be suitable (Figure 4.8).

HaskellQuest's target audience is composed of people with limited programming experience, however only two participants identified themselves as having "less than a year" worth of imperative programming experience. Further muddying the waters, while they both reported similar levels of engagement and enjoyment, their opinions of the learning curve of the game were diametrically opposed. One strongly agreed that the learning curve was appropriate, the other disagreed.

Overall these results make assessing the success of the game as a learning tool - at least in a binary way - difficult to determine. From the evidence gathered from user feedback so far it is clear that HaskellQuest was successful in being able to teach some concepts and syntax of Haskell to users. However, due to the limited sample size of participants involved so far and based on some of the detailed feedback mentioned already, it is clear that there is plenty of room for more development of the tool, more testing and more participant feedback informing its future direction and content.

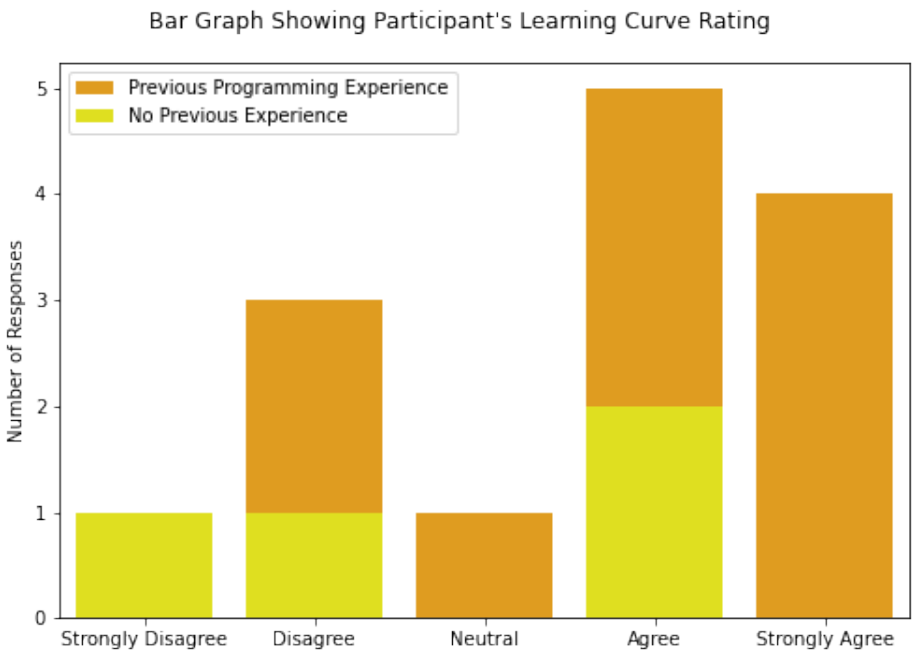

Figure 4.8: A Bar Chart Showing Opinions on the Learning Curve of HaskellQuest

4.5 Participant Ideas for Further Improvement

As part of the user feedback questionnaire participants were provided with space in the survey to give commentary on what they thought would make HaskellQuest a better learning experience for them. The most common comment was that participants wanted more opportunities for exploring the language with more puzzles and opportunities for repetition, to reinforce their learning rather than immediately moving onto a new topic. Awareness of this difficulty was an acute concern in HaskellQuest's development and was the primary motivation behind asking the participants specifically, as part of feedback whether they felt the learning curve of the game was appropriate. Even aside from the direct questions three participants directly commented on the fact that they thought the learning curve was too steep within their commenting space, suggesting that this is an area that participants felt most strongly about.

Participants also felt that further worked examples or a further extended help section would have been a good thing, or that a digital "notepad" to write down their thoughts in different puzzles would be useful. Encouragingly the type of feedback gained from the user feedback had little to do with the core concept and structure of the game itself and more to do with smaller point decisions that would be simple to fix with enough time.

Chapter 5

Conclusion

5.1 Lessons Learned

Ultimately HaskellQuest can be considered a success. The aim was to create an interactive and engaging learning tool facilitating easier learning of Haskell than other methods. It was certainly successful in its first goal of being engaging and interactive, and, although it was not a perfect learning tool and certainly requires more development and iteration to improve upon it, it ultimately succeeded in its pedagogical purpose.

5.2 Further Scope and Improvements

In addition to the feedback taken from analysis of user participation, one of the most important lessons taken from this project is that good quality games take a *long* time to develop and implement. The road from conception to implementation to further refinement and testing is as long and winding as they come, particularly if you are interested in making something not just for fun but also educational. HaskellQuest, both as a game, and a research project as it currently stands is merely a prototype, a rough outline sketch of what it could be. It is a good start, however, and there are a number of specific areas of improvement that could benefit the project going forward.

This project could branch out in several ways:

- Testing whether a more visual style of programming aids learning for students completely new to programming. The idea would be to use something Scratch-like to express list comprehensions and function syntax in Haskell. This could really aid visualising what exactly the language is doing. Once this is implemented, performing an A/B test with a group of beginners using Scratch-like and current (Haskell) versions of the game to see which group ends up with a better understanding of the language. This would work especially well with Haskell's guaranteed lack of side effects, so the blocky expressions of Scratch expressing 1-in-1-out would work perfectly. This was the original scope of this iteration of HaskellQuest, although the idea had to be scrapped due to time constraints.

- Extending levels and puzzles further. Many participants gave feedback that pointed to a desire for more puzzles to practise and reinforcing ideas as they were learned. Along with improving the writing of the mystery and supporting the links between the mystery storyline and the achievement of learning goals each puzzle listed in this project could be extended into a small level of its own, the completion of which, for example, could lead to the gathering of more clues and the progression of the detective story.
- Extending the research aspect of the project through much more user testing, likely using a framework for measuring and cataloguing enjoyment and engagement beyond a simple user survey, such as the one proposed in [19]. Accessing a larger sample of users would be key in order to help drive user feedback in the game design. Additionally, developing greater knowledge of the target audience and the range of their learning needs and preferences would help to inform pacing and scaffolding of learning material. It is important to support more targeted learning in HaskellQuest to meet those learning needs, while understanding that even within the same target audience, learning needs may vary.

Overall this project was extremely interesting to work on, the role of Serious Games have to play in education in the future seems to be an interesting one.

Bibliography

[1] Edeh Samuel Chukwuemeka ACMC. Most difficult programming languages to learn 2022: Top 12 hardest. https://bscholarly.com/most-difficult-programming-languages/. Accessed 8 April 2023.

[2] Stephen Cass. Top programming languages 2022. ieee spectrum. https://spectrum.ieee.org/top-programming-languages-2022. Accessed 8 April 2023.

[3] Carl Cheo. Which programming language should i learn first? https://carlcheo.com/startcoding. Accessed 8 April 2023.

[4] Contributor. Serious game classification : Escape from woomera (2003). http://serious.gameclassification.com/EN/games/1222-Escape-From-Woomera/index.html. Accessed 9 April 2023.

[5] Wikipedia Contributors. Expectancy-value theory. https://en.wikipedia.org/wiki/Expectancy-value_theory, February 2018. Accessed 12 March 2023.

[6] Wikipedia Contributors. Zone of proximal development. https://en.wikipedia.org/wiki/Zone_of_proximal_development, May 2019.

[7] Damien Djaouti, Julian Alvarez, and Jean-Pierre Jessel. Classifying serious games. *Advances in Game-Based Learning*, pages 118–136, 2019. https://www.igi-global.com/chapter/classifying-serious-games/52492.

[8] Jacquelynne S Eccles, Allan Wigfield, and Ulrich Schiefele. *Handbook of child psychology: Social, emotional, and personality development*. John Wiley & Sons, Inc.., 1998. https://psycnet.apa.org/record/2005-03132-015.

[9] Luke V Erickson and Dorothy Sammons-Lohse. Learning through video games: The impacts of competition and cooperation. *E-Learning and Digital Media*, August 2020.

[10] Sakshi Gupta. Top 5 easiest and top 5 hardest programming languages to learn. https://www.springboard.com/blog/software-engineering/top-programming-languages/, July 2020.

[11] Joanna Hernik and Elżbieta Jaworska. The effect of enjoyment on learning. *INTED2018 Proceedings*, March 2018.

[12] Mark H. Histed, Anitha Pasupathy, and Earl K. Miller. Learning substrates in the primate prefrontal cortex and striatum: Sustained activity related to successful actions. *Neuron*, 63:244–253, July 2009.

[13] Benjamin K. Hoover. Relating student perceptions of course interest, enjoyment, value, and ease with academic achievement in university agriculture courses. *NACTA Journal*, 61(4):324–328, 2017.

[14] Shouping Hu and George D. Kuh. Being (dis)engaged in educationally purposeful activities: The influences of student and institutional characteristics. *Research in Higher Education*, 43:555–575, 2002.

[15] Lightbot Inc. Lightbot. `https://lightbot.com/`.

[16] Amy Ko. Programming languages are the least usable, but most powerful human-computer interfaces ever invented — bits and behavior. `https://blogs.uw.edu/ajko/2014/03/25/programming-languages-are-the-least-usable-but-most-powerful-human-computer-interfaces-ever-invented/`, March 2014. Accessed 23 March 2023.

[17] Miran Lipovaca. *Learn you a Haskell for great good! : a beginner's guide*. No Starch Press, 2011.

[18] Elliot Markowitz. Data shows college students struggling to stay motivated. `https://www.fierceeducation.com/best-practices/data-shows-college-students-struggling-to-stay-motivated`, October 2020. Accessed 5 April 2023.

[19] Elizabeth Matthews, Geoffrey Matthews, and Juan E. Gilbert. A framework for the assessment of enjoyment in video games. *Lecture Notes in Computer Science*, pages 460–476, 2018. Proceedings of the International Conferenced on Human-Computer Interaction, HCI 2018.

[20] S. Nair and Jain Mathew. Learning through play: Gamification of learning, a systematic review of studies on gamified learning. `https://www.semanticscholar.org/paper/Learning-through-Play-Nair-Mathew/a9d3035d52631f4e35f7f0417d8aaaf34564e237`. Journal of Information Technology Management, Vol. 14, Issue 1, pages 113-126. 2021.

[21] Rex Page. Two dozen short lessons in haskell. `https://fileadmin.cs.lth.se/cs/Education/EDAN40/TwoDozenLessons/twoDznQ.pdf`, 1997.

[22] Reinhard Pekrun and Lisa Linnenbrink-Garcia. Academic emotions and student engagement. *Handbook of Research on Student Engagement*, pages 259–282, 2012.

[23] Rosa Maria Puca and Heinz-Dieter Schmalt. Task enjoyment: A mediator between achievement motives and performance. *Motivation and Emotion*, 23:15–29, 1999.

[24] Erin Schaffer. What's the best programming language to learn first? `https://www.educative.io/blog/best-first-programming-language`. Accessed 5 March 2023.

[25] Kelvin Seifert. Educational psychology. https://courses.lumenlearning.com/suny-hvcc-educationalpsychology/, 2012.

[26] Robert Spadoni. What is film atmosphere? *Quarterly Review of Film and Video*, pages 1–28, July 2019. https://doi.org/10.1080/10509208.2019.1606558.

[27] Statista. Most used languages among software developers globally 2019. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/, 2022. Accessed 8 April 2023.

[28] Statista. Global gaming penetration by country 2021. https://www.statista.com/statistics/195768/global-gaming-reach-by-country/, January 2023. Accessed 8 April 2023.

[29] Mark Sweney. More than 9m play candy crush for three hours or more a day. https://www.theguardian.com/games/2019/jun/26/more-than-9m-play-candy-crush-for-three-hours-or-more-a-day-addiction, June 2019.

[30] Unity Technologies. Unity - manual: Coroutines. https://docs.unity3d.com/Manual/Coroutines.html. Accessed 8 April 2023.

[31] BBC Bitesize Revision tool. Gcse computer science. https://www.bbc.co.uk/bitesize/subjects/z34k7ty. Accessed 8 April 2023.

[32] store.steampowered.com Valve. Hollow knight steam store page. https://store.steampowered.com/app/367520/Hollow_Knight/?curator_clanid=31285130. Accessed 8 April 2023.

[33] store.steampowered.com Valve. Outer wilds steam store page. https://store.steampowered.com/app/753640/Outer_Wilds/?curator_clanid=31285130. Accessed 8 April 2023.

[34] store.steampowered.com Valve. Overcooked 2 steam store page. https://store.steampowered.com/app/728880/Overcooked_2/?curator_clanid=31285130. Accessed 8 April 2023.

[35] store.steampowered.com Valve. Subnautica steam store page. https://store.steampowered.com/app/264710/Subnautica/?curator_clanid=31285130. Accessed 8 April 2023.

[36] wiki.haskell.org Wiki Contributor. Haskell in industry - haskellwiki. https://wiki.haskell.org/Haskell_in_industry, 2018. Accessed 28 October 2022.

[37] Tyler Wilde. How wasd became the standard pc control scheme. https://www.pcgamer.com/how-wasd-became-the-standard-pc-control-scheme/, June 2016. Accessed 9 April 2023.

[38] Mirac Yallihep and Birgul Kutlu. Mobile serious games: Effects on students' understanding of programming concepts and attitudes towards information technology. *Education and Information Technologies*, October 2019.

[39] Jeffrey Zacharakis. Elder play: Preliminary research results on how older adults learn through motorcycling. https://journals.sagepub.com/doi/10.1177/1045159519851462. Adult Learning Volume 3, Issue 4. May 2019.

[40] Jennifer Zosh, Emily Hopkins, Hanne Jensen, Claire Liu, Dave Neale, Kathy Hirsh-Pasek, S Solis, and David Whitebread. Learning through play: a review of the evidence. https://akcesedukacja.pl/images/dokumenty-pdf/Insight_and_Research/LEGO-Foundation---Learning-through-play---review-of-evidence-2017.pdf, 2017.

Appendix A

Participant Consent Form

Participant Consent Form

Project title:	HaskellQuest
Principal investigator (PI):	Don Sanella
Researcher:	Maxim Despinoy
PI contact details:	Don.Sannella@ed.ac.uk

By participating in the study you agree that:

- I will participate in playing the game HaskellQuest as provided and make a good effort to complete the game. I understand that this should take around 30-60 minutes.
- I will complete an electronic questionnaire after playing the game involving several questions similar to game problems to test my understanding and programming experience.
- 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**

Name of person giving consent	Date dd/mm/yy	Signature

Name of person taking consent	Date	Signature
Maxim Despinoy	13/03/2023	Maxim Despinoy

THE UNIVERSITY of EDINBURGH
informatics

Appendix B

Participant Information Sheet

Participant Information Sheet

Project title:	HaskellQuest
Principal investigator:	Don Sanella
Researcher collecting data:	Maxim Despinoy

This study was certified according to the Informatics Research Ethics Process, RT number 7313. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

Maxim Despinoy – The Researcher

Don Sanella – The Supervisor

What is the purpose of the study?

The study is interested in the effectiveness of gamifying learning environments when it comes to learning programming languages. It hopes to look into whether abstracting away from a syntactically written programming language into something more visual and “gamelike” has a positive effect on learner enjoyment, motivation and learning.

Why have I been asked to take part?

The study is interested in people across demographics, but is particularly interested in those with limited experience with functional programming.

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?

You will be given a link to a repository of the game including instructions on installing and running it for the first time. You are expected to install and run the game and make a good effort to learn and complete the puzzles. Absent of installation time, this should take 30-60 minutes.

After making a good effort to complete the game (whether you are successful or not) you are expected to take a short (20-30 minutes) questionnaire involving questions similar to those found in the game as well as several questions looking into your prior programming experience, age and education level.

If you accede in the consent form, you will be contacted between 6 and 9 days after performing the original tasks with an additional short questionnaire with similar questions that will be used to assess recall.

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

Unfortunately not.

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/research team consisting of Maxim Despinoy and Don Sanella.

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: Maxim Despinoy at s1750671@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.

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.

Appendix C

Puzzle Questions and Answers

C.1 Start of the Tutorial - Puzzle 1

C.1.1 Text in Puzzle Description

Send me a list with two string elements `"hello"` and `"world"`. Need to check you've been listening detective.

C.1.2 Elements in Data Panel

`n/a`

C.1.3 Model Answer

```
["hello", "world"]
```

C.2 Puzzle 2

C.2.1 Text in Puzzle Description

Send me the value of the first and last item of the `ints` list multiplied together. I'll give you a hint, detective it's about half of your assigned car's mileage, heh.

Second Hint: look at the Help menu for more information on the functions you might need.

C.2.2 Elements in Data Panel

```
ints::[Integer]
ints = [24, 100, 5, 319, 72013, 1751]
```

C.2.3 Model Answer

```
head ints * last ints
```

C.3 Puzzle 3

C.3.1 Text in Puzzle Description

Send me a list made from the first element of `firsts` added to each element of the `toAdd` list. This one's pretty essential to get.

Hint: you might want to use head again.

C.3.2 Elements in Data Panel

```
firsts::[Integer]
firsts = [2, 4, 6, 8]


toAdd::[Double]
toAdd = [12.0, 184.4, 17.3, 2.5]
```

C.3.3 Model Answer

```
[head firsts + a | a <- toAdd]
```

C.4 Puzzle 4

C.4.1 Text in Puzzle Description

Alright, here I need you to give me every last element of the list `Phrases`, as long as the string you're taking it from consists of at least 6 characters.

We're getting into tricky territory here, so if you get this one, I'll give you a pass.

C.4.2 Elements in Data Panel

```
phrases::[String]
phrases = ["grating", "do", "not", "no", "try", "geronimo", "oooooo",
"half", "way", "there", "somehow, palpatine returned", "igloo", "stay
safe_", "cheese-j", "with an o", "paint blob"]
```

C.4.3 Model Answer

```
[last a | a <- phrases, length a >= 6]
```

C.5 Puzzle 5

C.5.1 Text in Puzzle Description

I lied, no pass - instead you get more work!

Using the function `isDivisibleBy` which takes two numbers and gives whether the first is divisible by the second, send me a list that takes (in order) every element of list `as` that is divisible by 4, and is concatenated with every element of list `bs` that is divisible by 3.

Hint: Use `isDivisibleBy x 4` as a guard

C.5.2 Elements in Data Panel

```
as::[Integer]
as = [40, 23, 1, 8, 92, 13]
```

```
bs::[Integer]
bs = [36, 28, 2, 76, 15]
```

C.5.3 Model Answer

```
[a | a <- as, isDivisibleBy a 4] ++ [b | b <- bs, isDivisibleBy b 3]
```

C.6 Puzzle 6

C.6.1 Text in Puzzle Description

A little bit of an oddball puzzle this time, this one might show a result you don't expect, so maybe stick around after solving to see what you got.

In a list comprehension, take elements from `ints` and `listsOfInts`, then append elements of `ints` to the start of the lists you got from `listsOfInts`.

Is the result what you expected detective?
Hint: don't overthink this.

C.6.2 Elements in Data Panel

```
ints::[Integer]
ints = [1, 14, 144]


listOfInts::[[Integer]]
listOfInts = [[2, 3, 4],[15, 16, 17],[145, 146, 147]]
```

C.6.3 Model Answer

```
[i:is | i <- ints, is <- listOfInts]
```

C.7 The End of the Tutorial - Puzzle 7

C.7.1 Text in Puzzle Description

Replace the first character of every string in `strings` with the corresponding character in the same position from `characters`.

Good luck detective, this one is hard.

Hint: use `zip` and `a:as` notation and _ to select specific elements from each list and combine them.

C.7.2 Elements in Data Panel

```
strings::[String]
strings = ["one", "smart", "fellow", "he", "felt", "smart"]


characters::String
characters = "ofshsf"
```

C.7.3 Model Answer

```
[b:as | (b, a:as) <- zip characters strings]
```

C.8 The Murder Mystery - Puzzle 8

C.8.1 Text in Puzzle Description

I'm looking for a knife that is 20cm long and manufactured by Lovelace.

As far as I know, this shop only keeps their records of sale by the name of the knife, so I should try to find a knife the name of every knife that shares those properties.

C.8.2 Elements in Data Panel

```
knives::[(String, String, String, String)]
knives = [("Cleaver", "10cm", "Steel", "Jurgin"), ("Meat Knife", "20cm",
"Steel", "Jurgin"), ("Cutter500", "15cm", "Steel", "Lovelace"), ("Butter
Knife", "5cm", "Folded Iron", "Breton"), ("Letter Opener", "20cm", "Steel",
"Bretton"), ("Shank", "20cm", "Steel", "Lovelace"), ("Pointu", "12.5cm",
"Aluminium", "Jurgin"), ("Delimbificator", "20cm", "Steel Alloy", "Lovelace"),
("Cleaver", "10cm", "Steel", "Jurgin"), ("ShortswordJr", "20cm", "Copper",
"Lovelace"), ("Shortsword", "40cm", "Steel", "Lovelace")]
```

C.8.3 Model Answer

```
[a | a <- (a,b,_,d), b == "20cm", d == "Lovelace"]
```

C.9 Puzzle 9

C.9.1 Text in Puzzle Description

I've managed to work out a possible list of murder weapons, now to turn that into a list of suspects.

The weapon looked quite pristine which puts the likely date of purchase into maybe a week before the murder, I should be looking at the days 1-7 of the month...

Hint: Elements are of the form `(String, String, Integer)` representing the knife name, person name and month day also: the elements need to be in the order they appear in the list, try using the knives list as a second generator.

C.9.2 Elements in Data Panel

```
knives::[String]
knives = ["Shank","Delimbificator","ShortswordJr"]
```

```
sales::[(String, String, Integer)]
sales = [("ShortswordJr", "Dorothy Matthers", 1), ("Butter Knife", "Tiffany
Day", 2), ("Letter Opener", "Valefor Thirst", 14), ("Meat Knife", "Alison
Bree", 16), ("Cutter500", "Jemimah Duval", 3), ("Delimbificator", "John
Brown", 28), ("Shank", "Poe Mystiq", 16), ("Cutter500", "Yasmin Brown",
13), ("Butter Knife", "Ceciliah Fox", 11), ("Meat Knife", "John Brown",
21), ("Cleaver", "Edgar Allen", 22), ("Butter Knife", "Jeffrey Boycott",
19), ("ShortswordJr", "Henri Dreyfus", 3), ("Cleaver", "Henry Potter",
```

```
7), ("Shortsword", "Jemimah Duval", 25), ("Meat Knife", "Henri Dreyfus",
13), ("Cutter500", "Ceciliah Fox", 6), ("Butter Knife", "Trevor Potter",
24), ("Letter Opener", "Dominic Punnings", 30), ("Delimbificator", "Edward
Maiorson", 4), ("Delimbificator", "Ceciliah Fox", 8), ("ShortswordJr",
"Jemimah Duval", 4), ("Cleaver", "Finnegan Fawks", 12), ("Shank", "John
Brown", 3), ("Pointu", "Yasmin Brown", 2), ("Pointu", "Jericho Yuletag",
16), ("Cleaver2", "Theirry", 5)]
```

C.9.3 Model Answer

```
[b | (a,b,c) <- sales, d <- knives, a == d, c >= 1, c <= 7]
```

C.10 Puzzle 10

C.10.1 Text in Puzzle Description

This one might need to be done in stages, I need to compare their monetary records with the suspect list that I uncovered earlier.

If I find all of them, then I know nothing untoward is going on, but if one of them is not on there, we'll have found our prime suspect. I need their name as the only element of my answer.

Hint: use `not (elem x y)` as a guard to only take elements that are not in a given list!

C.10.2 Elements in Data Panel

```
suspects::[String]
suspects = ["Dorothy Matthers","Henri Dreyfus","Edward Maiorson","Jemimah
Duval","John Brown"]
```

```
money::[(String, Integer)]
money = [(String, Integer)]<> body=[("Dorothy Matthers", 1050), ("Tiffany
Day", 300), ("Valefor Thirst", 200), ("Alison Bree", 150), ("Jemimah
Duval", 1400), ("John Brown", 2000), ("Poe Mystiq", 350), ("Yasmin Brown",
220), ("Ceciliah Fox", 170), ("Edgar Allen", 50), ("Jeffrey Boycott",
200), ("Henri Dreyfus", 400), ("Henry Potter", 100), ("Trevor Potter",
100), ("Dominic Punnings", 900)]
```

C.10.3 Model Answer

```
[a | a <- suspects, not (elem a [b | (b, _) <- money, c <- suspects, b
== c])]
```

Appendix D

User Feedback Quiz

Note: Due to some issues within Microsoft forms the last page had to be screenshot and so is not formatted as nicely.

HaskellQuest Participation and Consent form

* Required

1. I have read the HaskellQuest Participant Information document and consent to the terms labelled under "Consent" therein. *

   - ◯ Yes
   - ◯ No

2. I have read the HaskellQuest Participant Consent form and agree with the following statements *

   - ☐ I agree to allow my data to be used in future ethically approved research
   - ☐ I agree to take part in this study

Questions about you

Answering questions is optional.

3. How much experience do you have playing video games?

- ◯ No experience
- ◯ Some experience (e.g. perhaps rarely with friends)
- ◯ A reasonable amount of experience (e.g. as occasional entertainment)
- ◯ A large amount of experience (e.g. as regular entertainment)

4. How much imperative programming experience do you have? (e.g. using C/Java /Python)

- ◯ None
- ◯ Less than a year
- ◯ 1-3 years
- ◯ More than 3 years

5. How much functional programming experience do you have? (e.g. Haskell, using function arguments in Python/Java, Python list comprehensions)

- ◯ None
- ◯ Less than a year
- ◯ 1-3 years
- ◯ More than 3 years

6. How would you describe your problem solving skills?

- ◯ Very weak
- ◯ Weak
- ◯ Strong
- ◯ Very strong

7. How would you describe your level of education in fields that require classical problem solving skills? (STEM subjects would be the most obvious example)

- ◯ Pre Highschool level
- ◯ Highschool level
- ◯ University level
- ◯ Postgraduate level

Questions about HaskellQuest

Answering questions is optional.

8. How much do you agree with the following statements, if you answered "neutral or disagree" to any of them, please explain why you feel that way in the space given in question 9.

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
I enjoyed playing HaskellQuest	◯	◯	◯	◯	◯
I felt engaged playing HaskellQuest	◯	◯	◯	◯	◯
The addition of music helped with my engagement	◯	◯	◯	◯	◯
The addition of a storyline helped with my engagement	◯	◯	◯	◯	◯
I was able to solve most in-game questions with the information provided	◯	◯	◯	◯	◯
The UI for helping solve problems was intuitive	◯	◯	◯	◯	◯
The help menu was useful	◯	◯	◯	◯	◯
The learning curve between puzzles felt appropriate	◯	◯	◯	◯	◯

9. Additional thoughts/explanations pertaining to the answers above

10. In what ways do you think HaskellQuest could be improved to provide a better learning experience?

11. Any additional remarks

12. What is the name of the type in Haskell for representing lists of characters, words or sentences?

Enter your answer

13. Select the list given by the list comprehension "[a + 1 | a <- [1, 2, 3]]".

- ◯ [1, 2, 3]
- ◯ [1, 2, 3, 1]
- ◯ [2, 3, 4]
- ◯ [2]

14. What might be the typing for the list: q = [("one", ['a']), ("two", ['2']), [("three", ['n', 'o'])]

- ☐ q :: [(String, [Int])]
- ☐ q :: [(Double, String)]
- ☐ q :: [(String, [Char])]
- ☐ q :: [(String, String)]

15. What character represents the concept of a junk value? You might use it to discard certain elements of tuples for instance.

Enter your answer

16. Which function would give you the first value of a list, and, give it's equivalent pattern using the a:as notation of lists.

Enter your answer

Appendix E

Haskell Quiz Marking Scheme

Question	Example Answer	Further Comments
Question 12	String	Worth 1 mark
Question 13	[2,3,4]	Worth 1 mark
Question 14	[(String, [Char])], [(String, String)]	Both answers are worth 0.5 marks each so an answer of [(String, [Char])] only recieves only 0.5 marks. Any answers including an incorrect solution cannot receive any marks.
Question 15	_ (underscore)	Any variation of "underscore" or "_" is acceptable. Worth 1 mark
Question 16	head, a:_	Each part of the question is worth 0.5 marks similar to question 14. If the participant answers part of the question correctly they receive full marks for that part of the question. For the first part, any answer resembling "head", "the head function" is correct. For the second part, any clear knowledge of patter matching showing the isolation of the first element of a list is correct. Other possible solutions might include "a:as = a" or "x in x:_".

Table E.1: Marking Scheme for Quiz Questions 12-16