

HaskellQuest: Can a Fun Game be Educational?

Daniel Segboer



4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh
2024

Abstract

Haskell and functional programming as a whole are often taught in first year computing science courses to encourage students to think in a different way than they might be used to from previous object oriented languages they might have learned. As such, students and developers alike find it quite difficult to grasp the very basic concepts of this programming paradigm.

HaskellQuest is a 2D puzzle game that was developed to lower the barrier of entry by allowing learners to physically interact with functions without having to actually write code. The game features 4 chapters covering the basics of Haskell and, as two phases of testing have proven, it has succeeded as a fun and educational introduction to functional programming.

Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee.

Ethics application number: 808784 ,715532

Date when approval was obtained: 2023-09-11, 2023-12-20

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.

(Daniel Segboer)

Acknowledgements

Special thanks to all who tested the game and my supervisor Donald Sannella for his continued support throughout the project.

Table of Contents

1	Introduction	1
1.1	Goals	1
1.2	Report Structure	2
2	Background	3
2.1	Haskell and Why it's Important	3
2.1.1	Facebook	3
2.1.2	Cardano	4
2.1.3	Haskell in Education	4
2.2	Learning Styles	5
2.3	Gamification	5
2.3.1	Duolingo	5
2.3.2	Previous Work	8
2.4	Questionnaire	10
3	Approach	11
3.1	Game Design	11
3.1.1	Planning	11
3.1.2	Pre-production	11
3.1.3	Production	12
3.1.4	Testing	12
3.2	Game Engine	12
3.2.1	Engines Considered	13
3.2.2	Chosen Engine	13
4	The Game	15
4.1	Game Idea	15
4.2	Mechanics	15
4.3	Download	17
5	Design	18
5.1	Level Design	18
5.1.1	Puzzles	18
5.1.2	Progression	20
5.2	User Interface	24

6	Production	25
6.1	Values & Functions	25
6.2	The Player	27
6.3	Saving data	27
6.4	Visuals	28
7	Evaluation	31
7.1	Alpha Testing	31
7.2	Before Beta Testing	32
7.2.1	Bug Fixes	32
7.2.2	Responding to Feedback	32
7.2.3	Fun changes	33
7.3	Beta Testing	34
8	Conclusion	35
8.1	Main Achievements	35
8.2	Limitations	36
8.3	Future Features	36
8.3.1	Gamification	36
8.3.2	Education	37
8.3.3	Other	37
	Bibliography	39
A	Questionnaire	45
A.1	Functional Programming Education Questionnaire	45
A.2	Participants' information sheet	48
A.3	Participants' consent form	50
B	Testing	51
B.1	Alpha Testing Feedback Form	51
B.2	Alpha Testing Bug Report Form	55
B.3	Beta Testing Feedback Form	57
B.4	Participants' information sheet	61
B.5	Participants' consent form	64
C	HaskellQuest Game Design Document	65
D	HaskellQuest Rough Timeline	95

Chapter 1

Introduction

Learning Haskell in traditional ways is quite unengaging and comes with a high barrier to entry. With the language being quite commonly introduced to first year Computing Science students that have never encountered functional programming, it's important to make learning the language as accessible and engaging as possible for students. Not only that, but developers wishing to learn the language outside of traditional education should not be discouraged and also offered an easy way to grasp the basics of the language.

HaskellQuest was developed to fulfil this very purpose. It is a puzzle game consisting of 16 levels that offer an even more gamified version of preexisting programming games that often rely on users inputting code [1]. It allows those with absolutely zero previous experience to explore the language. Instead of writing code, players can physically interact with values and input them into functions helping them gain an intuition for various aspects of the language, from very basic arithmetic all the way up to monads. The goal of each level is to transform an initial value into some specific goal value by passing it through various functions in a specific order. The experience is kept engaging thanks to various game-like elements such as enemies, a clean and charming art style, as well as a timer with medals that players can earn, encouraging them to replay levels to improve their time and simultaneously their understanding of Haskell.

1.1 Goals

The main goals for HaskellQuest were:

- Successfully complete a fun yet educational game about Haskell that didn't rely on players writing code.
- Allow players to save their game and come back to it later.
- Prevent the project scope from blowing up to complete HaskellQuest on time and allow for multiple rounds of testing for evaluation.
- Adhere to software and game development industry standards.

1.2 Report Structure

This report offers a comprehensive look at the development of a 2D action puzzle game for teaching Haskell:

- Chapter 2 provides a look into Haskell, functional programming education and gamification alongside related questionnaire results.
- Chapter 3 introduces key aspects of game and software design used in the industry and a look into game engines.
- Chapter 4 outlines the main ideas of the game and explains how it works.
- Chapter 5 offers an insight into the design process behind puzzles and the UI.
- Chapter 6 looks into the interesting implementation details behind HaskellQuest.
- Chapter 7 shows the results of two phases of user testing.
- Chapter 8 summarises the main achievements, limitations and potential future features of HaskellQuest.

Chapter 2

Background

2.1 Haskell and Why it's Important

Haskell is an advanced, purely functional programming language [2] published in 1990 [3], being named after Haskell B. Curry – a mathematician whom the creators wanted to honour for his work that was a foundation for functional languages - with his wife later remarking “You know, Haskell actually never liked the name Haskell”.

Haskell's mathematical precision has lead it to become the 27th most commonly searched language on Google [4] as its use has been creeping into all sorts of different sectors. According to Haskell Wiki [5] the language has found uses in aerospace and defence, tech and web startups, finance, hardware, and even a lawnmower manufacturer. Haskell Cosmos [6] lists over 200 companies from around the world that use the language including big banks, like Barclays, Standard Chartered Bank, Deutsche Bank, as well as big tech companies like Microsoft, Tesla and Facebook.

2.1.1 Facebook

Facebook (now Meta) has been dabbling in Haskell for a while. For example, their 2014 Haxl library for Haskell allowed for implicit concurrency – allowing developers to more easily fetch data from other systems at Facebook[7]. Since 2015, Facebook have officially been using Haskell when they switched their spam filtering solution “Sigma” from using their in-house language FXL which, alongside being quite slow and lacking abstraction, could not handle the ever increasing complexity of Facebook's anti-spam policies[8]. Haskell was chosen for a variety of reasons, the main one being performance. When introduced in 2015, the Haskell-powered version of Sigma was processing more than 1M requests per second, at a speed up to 3x faster than FXL. Other reasons for choosing Haskell include it's functional nature, interactivity and the ability to deploy new policies quickly.

2.1.2 Cardano

Haskell has also found its place in the industry of cryptocurrencies, being the chosen language for Smart Contracts on Cardano – a blockchain launched in 2017 by the co-founder of Ethereum, Charles Hoskinson, with the goal to be scalable and energy efficient [9]. The native token of Cardano, ADA, is one of the biggest cryptocurrencies by market cap. When it comes to Haskell's role, Elliot Hill of the Cardano Foundation stated that “assurance and reliability are essential” when billions of dollars are expected to be held on the blockchain, locked through smart contracts [10]. That's why Cardano's open source Plutus & Marlowe smart contract libraries were written in Haskell, the code offers a high level of assurance as it can be formally verified. Additionally, they could utilise Haskell's already existing resources including documentation, toolkits, community & research instead of attempting to reinvent the wheel – like so many other blockchains have tried. Most opted to use a custom language, which comes with its own complications as they all require completely new libraries, learning resources, toolchains, etc. According to a 2018 paper, 3.4% of all smart contracts were found to be vulnerable using a simple algorithm that just attempted the most common exploits [82]. The consequences from a poorly implemented custom language can be disastrous [11]. For instance, in 2016 the DAO hack saw hackers exploit a vulnerability in the smart contracts of the DAO, a decentralised venture capital fund, resulting in around \$150M being siphoned from the fund over a matter of weeks [12]. The exploit was identified beforehand, but not fixed. After the whole ordeal, the community was left fractured and eventually led the DAO to being shut down.

2.1.3 Haskell in Education

Functional programming is taught in first year programming courses around the world. The Haskell Wiki lists around 50 courses from many universities like the University of Oxford, University of Edinburgh as well international universities like the University of Bacau in Romania, the University of Heidelberg in Germany, among others [13]. The language itself was ranked first by 1800 programmers in the Hammer Principle Programming Languages survey in the category “I would recommend most programmers learn this language, regardless of whether they have a specific need for it” [14].

Functional programming is a lot different to what students are used to. Edsger W. Dijkstra wrote a letter in 2001 to the University of Texas, after hearing rumours that Haskell was going to be replaced by Java [15]. He mentioned that “a very practical reason for preferring functional programming in a freshman course is that most students already have a certain familiarity with imperative programming” and it would help them understand that the world of programming is much more vast than they perhaps would've thought. He believed it helped create an open-minded culture for Computing Science at the university, where students could experience new languages and learn to think in a new way – something that his Colleagues from outside Texas would envy.

2.2 Learning Styles

There is an idea in education that everyone has a preferred style of learning. This theory originated from Neil D. Fleming, a school inspector in New Zealand. He explained, "I was puzzled when I observed excellent teachers who did not reach some learners, and poor teachers who did"[78]. In his and Colleen Mills' seminal 1992 paper they attributed that to students having different preferred styles[79], later launching the VARK model[16] to represent the four different modalities: visual, aural, read/write, kinesthetic. As of 2020, almost 90% [81] of educators from all around the world were found to still believe in this model. Over half of the proponents of this theory, teachers included, believe learning styles are inherited and unchangeable - a fundamental part of a person[80].

However, this is a myth - one that survives because "the appeal of the learning styles myth rests in its fit with the way people like to think about behavior"[17]. The truth is that styles are, at best, a preference. Instead of obsessing over offering specific styles for students, it's found that it's better to offer multi-modal lessons that combine hands-on learning, audio and visual stimulus, reading and writing[18]. Games are a perfect platform to accommodate multi-modal learning as they offer the tools for players to see, hear, read, write and interact with a game world[19].

2.3 Gamification

Intentional gamification is the process of enhancing an activity to be more fun and enjoyable by introducing game-like elements, thus increasing the motivation and retention of users [20]. Serious games are a related concept that use gamification to turn educational resources into more engaging games, being used in a variety of sectors including education, science, health, aerospace and defence[21]. The main strength of these games, alongside their increased appeal, is the immediate and individual feedback that can be generated for each user, allowing them to learn in a more personalized way. While not essential, a story can help immerse a player in the experience and give them a reason to care. A large part of serious games are simulators – such as Microsoft Flight Simulator [22] – that immerse the player into a world full of fabricated scenarios and allow them to practice what to do in the real world safely with no risk of serious damage.

2.3.1 Duolingo

Duolingo is primarily a language learning app (with recent spin-offs allowing children to learn basic math and literacy) for web, IOS and Android [23]. The app currently features 39 languages for English speakers alone including the popular choices like Spanish and Japanese to more niche tongues like Scottish Gaelic and Polish. The app even features constructed languages like the infamous Esperanto or Klingon from Star Trek [24]. While the app focuses mainly on its educational core and it is not exactly a game, the product page claims "Duolingo is fun and effective. Game-like lessons and fun characters help you build solid speaking, reading, listening, and writing skills"

[25]. Indeed, the app does contain a lot of game-like elements such as an item shop, skins, levels, streaks, daily goals, cute characters, community forums, funny sentences [26] among others. An experienced Duolingo user explained that “the true power of Duolingo is getting you started in a language” [27], almost akin to being the gateway drug of language learning.

Beth Chasse of Duolingo said “Games have always been a source of inspiration for us at Duolingo” in her post about gamification[28]. They understand that language learning can be overwhelming, so their goals with the app were primarily to make learning fun, and to help users develop long term study habits. Game-like levels alongside a progress bar (as seen in figure 2.1) were introduced for lessons, boosting engagement. Their next goal was to encourage users to review old material, as spaced repetition was shown to improve knowledge retention. They achieved this by introducing skill degradation, which visualised the concept of spaced repetition by making skills become visibly damaged over time. Users loved having fully maxed out skills and wanted to repair these broken skills – the Duolingo team saw an improvement to all engagement metrics after implementing this feature.

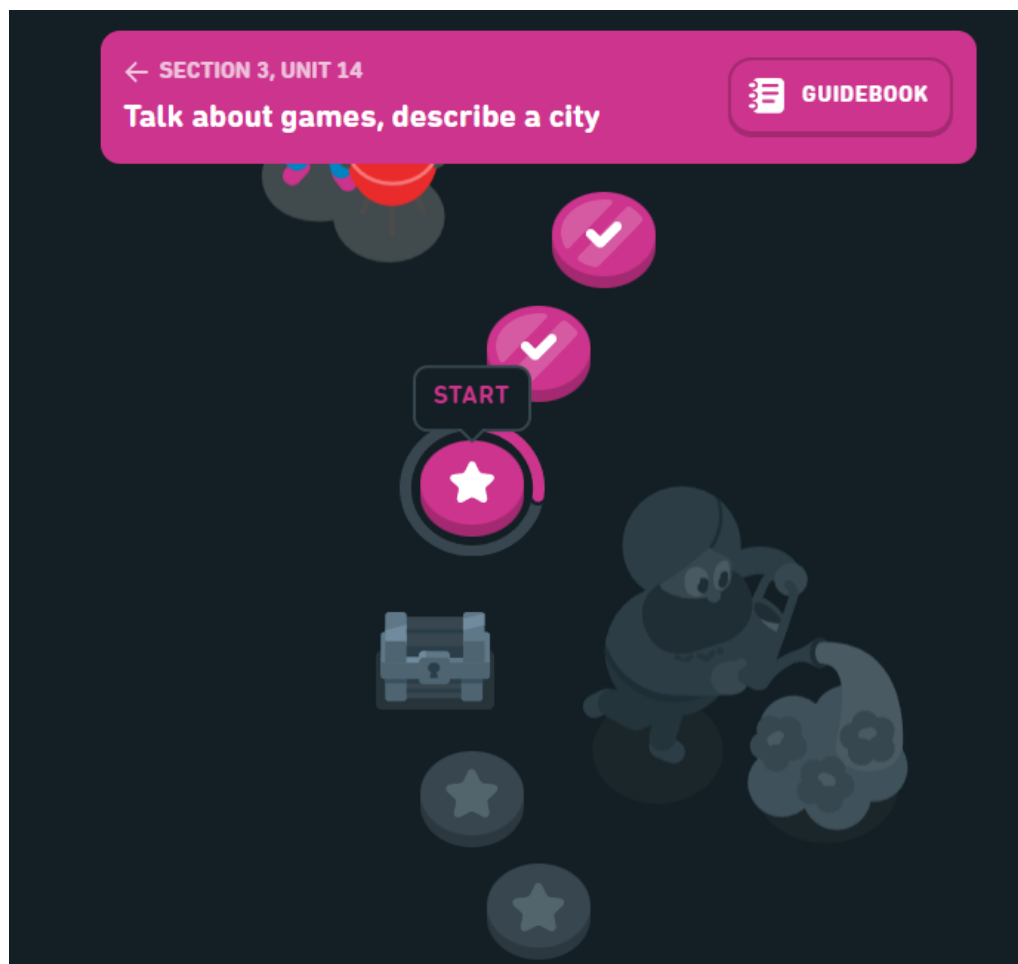


Figure 2.1: Duolingo’s gamified interface featuring lessons split into levels with progress bars.

The next goal of gamification was to encourage consistency. Streaks (figure 2.2) are a huge part of Duolingo, a simple counter that holds you accountable to practice daily by increasing upon lesson completion (alongside an exciting animation) and resetting when a lesson is missed. As of 2022, 6M learners have a streak of more than 7 days [29] with those users having a 3.6x higher chance to finish a course when compared to users who didn't reach a week long streak. These were introduced to make the act of completing a daily lesson almost automatic, encouraging consistency. For newer users, the streaks make them feel like they're making big progress with their studies, as a jump from 2 to 3 days is 50%. For long term users, Duolingo taps into loss aversion. This is a bias in our brain that discourages you from losing something, it encourages users to sneak in a quick 5 minute lesson even on their laziest day. Freeze streaks were introduced later on, further boosting the daily learners. Players could buy an item from the shop to prevent the loss of a streak on days where they weren't able to learn. This was done as losing a streak was very demotivating, and perhaps their easy-to-lose nature discouraged some from starting one in the first place. Offering people slack as they learn has been shown to be more motivating than having to stick to rigid rules all the time [30].

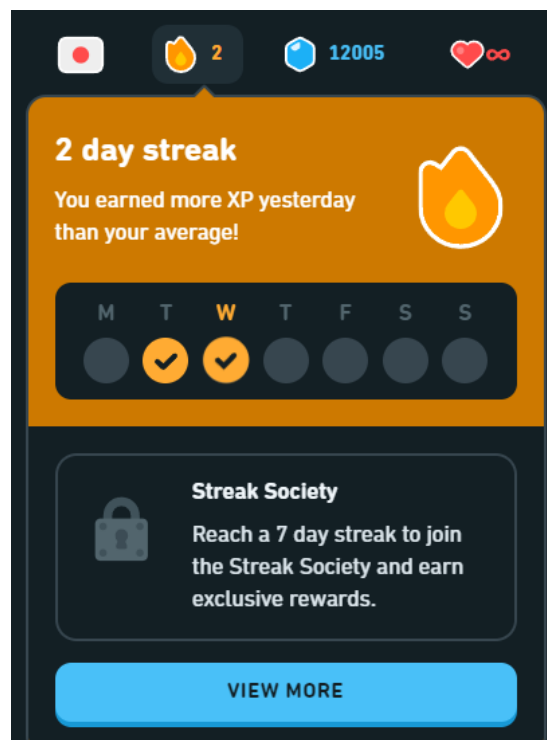


Figure 2.2: Duolingo's streak, represented by a flame that users must keep alight by completing daily lessons.

The Duolingo team understands the importance of exciting art direction to keep players engaged. Flashy animations were introduced for extending a streak, as well as special variants for milestones (week, year, etc) which further boosted engagement – showing that exciting visuals can make a difference even in educational applications. Their adorable mascot has become beloved by the internet, often being paired with threatening captions mocking Duolingo's aggressive notifications [31].

2.3.2 Previous Work

There are countless projects that have tackled the issue of turning the act of learning Haskell into something that's a bit more more engaging and accessible. Each have their different approaches when it comes to themes and what aspects of Haskell they cover, but most share the key idea of requiring users to input code in order to progress through the game.

Maxim Despinoy's 2023 HaskellQuest[32] is a story based puzzle game made in Unity that features the player take on the role of a detective who uses Haskell to write queries and find clues. Right from the beginning the player is immersed into the game world through a striking introduction sequence complete with music, voice acting and stylised visuals. Despite offering no educational value, this gives the player a reason to care about the world they're about to be put in. In the game itself, the player takes control of a detective in a 3D world who must find clues to solve the murder mystery on yellow PCs scattered around the level. It slowly introduces new Haskell concepts and extensively explains them to the player through lengthy dialogue and multiple examples, which the player can come back to if they need to. The puzzles mainly focus on list comprehensions and require users to write Haskell code that is quite similar to what they might actually write in the real world. The player is told what they need to do e.g. find "a knife that is 20cm long and manufactured by Lovelace" from a list of knives as seen in figure 2.3. The output of the Haskell code written by the user is shown, allowing them to get instant feedback as to where they went wrong and amend their query appropriately. If the code fails to compile, an appropriate error is shown to the player. Once the correct output is achieved, the puzzle is complete and marked as "solved".



Figure 2.3: PC interface from Maxim Despinoy's HaskellQuest

Eve Bogomil’s HaskellQuest[33] project from 2023 is a turn-based based RPG (role-playing game) also made in Unity with a heavy emphasis on story that follows the main character uncovering lost Haskell knowledge in a fantasy world. It starts off with a simple introduction sequence that sets the scene and allows the player to choose their name. The player is then dropped into a large 2D world with a charming pixelated artstyle and encouraged to explore, talk to NPCs (non-player characters), find chests and notes, and immerse themselves into the world. This not only keeps a player engaged but also helps them progress through the game and uncover Haskell knowledge which is stored in their grimoire - an ancient spell book. Eventually the player is lead to a cave where they must write Haskell code to defeat enemies, as seen in figure 2.4. This Haskell code, compiled externally on an online compiler, is quite abstract and features actions like attacking and healing which help novices gain an intuition for functional programming. The game is quite limited with only 3 levels within the cave which are very similar to each other. It also does not currently give players the ability to save their progress which makes it difficult to come back to at a later point.



Figure 2.4: Battle screen from Eve Bogomil’s HaskellQuest

These two games, despite their varied themes, both share an underlying problem: relying on users to write code and ultimately ending up just being glorified tutorial exercises with pretty pictures. This is somewhat akin to ScalaQuest[34], a similar game for learning Scala that also relies on players solving puzzles by inputting code. Another issue is the amount of text in both games being very extensive - they rely on it to explain not only Haskell but also key game mechanics before allowing the player to actually interact with the language. For beginners especially, it’s easy to get overwhelmed and lose interest before even writing any code.

2.4 Questionnaire

Before designing HaskellQuest, a study was conducted to understand the experience of students of the INF1A course that teaches Haskell in the University of Edinburgh. Its purpose was to see how helpful a game would be in helping people understand Haskell and functional programming in general, and what the game should actually include. All 17 responses to the functional programming education questionnaire can be seen in appendix A.

82% of participants stated that they had no previous experience with functional programming. Most participants (82%) agreed that Haskell helped them with their understanding of programming in general, with one person disagreeing (6%) and two people not being sure (12%). 82% again agreed that they were happy to have learned a functional programming language, as opposed to only learning object oriented languages such as Python or Java. 17% of people were unsure. This data shows that there is value in learning a language that forces people to think in a different way to the more commonly taught object oriented languages, and as such justifies the existence of HaskellQuest.

Participants were asked how hard learning Haskell was on a scale of 1 to 10, with 1 being easy and 10 being very difficult. The average score was 4.35, with a large spread between 1 and 8. The two most voted on responses were 3 and 7, showcasing just how varied people's experience with Haskell was. The question about the most difficult part of Haskell was open ended and gave participants the opportunity to say what they found hard while learning the language. The most mentioned word was "monads". Some people mentioned that it was difficult to initially get the hang of thinking in a different way, with one participant saying that "getting started especially week 1-4" was hard. Another participant said it was hard "thinking without oop loops". Students did not find learning this language easy, not only with more advanced topics like monads but also at the beginning, and as such HaskellQuest should cover all of these.

Participants were asked about what types of games they enjoyed. The results were fairly evenly spread between "Story/Characters", "Puzzles", "Action", "Art Direction" and "Exploration", with each one receiving around 52% of the votes. From this data it was clear that computing science students enjoy a wide variety of games and thus for HaskellQuest to succeed, it should incorporate as many elements as possible if time allows. People were evenly split between "Complexity" and "Simplicity", with the former getting 5 votes and the latter receiving 4.

Chapter 3

Approach

3.1 Game Design

The standard game design pipeline contains 7 stages[35] which include: planning, pre-production, production, testing, pre-launch, launch, post-production. For HaskellQuest, this was shortened to 4 stages, due to time constraints.

3.1.1 Planning

The planning for HaskellQuest began before summer of 2023, and lasted until week two of semester one. It answered some basic questions about what the game will actually look like, including whether it will be 2D or 3D, what key features it will contain, who the target audience will be. A rough timeline was produced (appendix D), which utilizes the agile software development cycle to split the whole project into short 2 week sprints. At the end of the 2 weeks, there would be time to reflect on progress. There was also a bit of leeway at the end of the production stage as game development can often run into hurdles that can delay the project from being finished. In 2023 alone, there have been over 100 releases that were delayed [36] . The planning stage also included revisiting and relearning the Haskell language, and understanding its capabilities.

3.1.2 Pre-production

Pre-production involved researching past educational games and producing a game design document. A GDD is a “living document”[37] that evolves throughout the production lifecycle. It usually does not have any specific format, but it should include everything essential for the developers to later create the game. The HaskellQuest GDD included a wireframe for the User Interface of the game, an inspiration board for art direction, enemy designs, as well as a detailed plan for the initial 13 levels of the game. This document can be seen in appendix C. The level plans were initially drawn up roughly, but then remade more professionally using Smartdraw[38] . Due to time constraints, the initial build of the game had only 13 levels, covering the very basics of Haskell. This would be the minimum viable product for the game[39] , only including the essential features needed to get it into the hands of players. Later on it was extended

to 16 levels for the beta test alongside other extra features, and still has space to expand in the future with new levels.

3.1.3 Production

The production stage was the most time consuming part of the project, starting at week 6 of the first semester and lasting well into the second semester. It involved turning the design document into an actual playable game. This stage took notes from the Agile framework and split the game into 2 week sprints, each one of which would focus on specific aspects of the game and build on previous progress. This plan was not rigid, as often features had to be pushed back to later sprints, however there was enough leeway in the plan to account for this. By the end of the first sprint a Vertical Slice of the game was to be made, which is a small demo of a game that acts as a proof of concept for the final product[40]. This version of the game was very limited in scope and used to verify that the core game mechanics were fun and that the game was on the right track. The minimum viable product of the game was ready by the end of the first semester, and could be sent out to testers at the beginning of the second semester. There was then time to improve the game based on feedback, and conduct another test.

3.1.4 Testing

The testing for HaskellQuest was split into two main phases, alpha and beta. In game development, the main focus of the alpha phase is to identify problems during development, meaning the game is still unfinished. This includes finding any bugs and making sure the software is fulfilling its original purpose[41]. The alpha testing for HaskellQuest involved sending out the game alongside a questionnaire and bug report form to University of Edinburgh students that have studied the INF1A course that teaches Haskell. The questionnaire was used to evaluate how successful the game was of fulfilling its goal of being fun and educational, while also providing testers with the opportunity to share their suggestions for the future of the game. The beta testing phase of game development focuses more on polishing up an almost finished game before release. In terms of HaskellQuest, the second phase of testing involved giving testers a questionnaire that included questions to evaluate changes made since the alpha test. Alongside beta and alpha testing, there were frequent points throughout the project where informal feedback from roommates and colleagues was gathered.

3.2 Game Engine

Game engines simplify the creation of games for developers. They are a framework that includes basic video game elements so that the programmer doesn't need to make them from scratch every time [42]. They come prebuilt with ways to manage input, process sound, render graphics, calculate physics and interaction between objects, among many other things [43]. One of the options for this project was to write the game from scratch in Haskell, but that was deemed unnecessary. The Haskell code featured in this game is not written by the user and does not need to be compiled while playing making it possible to use a game engine that, despite using a different language, allowed the

utilization of the aforementioned advanced features to simplify the game development process. The Haskell functions were simple and could be easily rewritten in whatever language the engine used.

3.2.1 Engines Considered

Nowadays there are plenty of free game engines for indie developers to choose from such as Unity[44], Unreal Engine 4 & 5[45], Godot[46], RPG Maker[47], GameMaker Studio[48] and more. RPG Maker was out of the question since the beginning, as HaskellQuest was not a Roleplaying Game (RPG) that mainly focuses on a story where players level up their character, interact with characters and collect items[49]. Unreal Engine was a strong contender with Unreal Engine 5 recently being released in April 2022, featuring an open source repository on GitHub, life-like graphics, and a strong multiplayer framework. The engine holds around a 13% share in the game development market [50] and is used to power many popular games such as the infamous Fortnite[51]), Borderlands 3 [52], Sea of Thieves [53], and Ark: Survival Evolved[54] However, the engine was ultimately not chosen due to its focus on 3D games, rather than 2D ones. The engine, while featuring a simplified visual coding solution that they call Blueprints[55], still relies on the C++ which would be another thing to learn. This left me with a choice between Unity, Gamemaker Studio and Godot.

3.2.2 Chosen Engine

This project uses Unity (as seen in figure 3.1). The engine uses GameObjects, to which it's possible to attach scripts, collision, rigidbody physics, audio, and many more prebuilt features. The main reason for the choice of this engine is extensive past experience in not only the engine but also C#, the programming language it uses. This experience includes some simple game projects made for fun and the creation of custom levels and assets for the free to play PC game Unturned[56], with several maps being included officially in the game itself[57][58]. An important consideration when choosing the engine was making sure everyone who wanted to program in Haskell would be able to play it. Unity allows for this as its possible to export games for a variety of platforms including not only the major desktop operating systems like Windows, Mac and Linux but also mobile platforms. Its flexible input system allows HaskellQuest to accept a variety of input devices. Another big reason for using the engine is its active community [59]. The engine has garnered a strong and passionate community that can help directly on the Unity Forums [60] or via the limitless free tutorials for the engine on websites like YouTube, with channels such as Brackeys[61] having over 400 videos mainly dedicated to game development in the engine. Unity themselves also have their own learning resource called Unity Learn[62], featuring over 500 hours of learning resources for the engine. The Unity Asset Store[63] features countless free assets ranging from UI, characters, fonts, textures, and many more. These allow developers to focus on getting the game playable while still making sure it looks great without having to spend an excessive amount of time on art. Unity currently has the highest market share of any engines[64], with 48% of developers in indie studios using Unity as their primary engine [65]. Many top games are made with the engine, including Genshin Impact[66]

with 60M monthly average players[67], Among Us with 400M total players as of 2023 [68], Escape from Tarkov [69] with around 2M monthly average players.

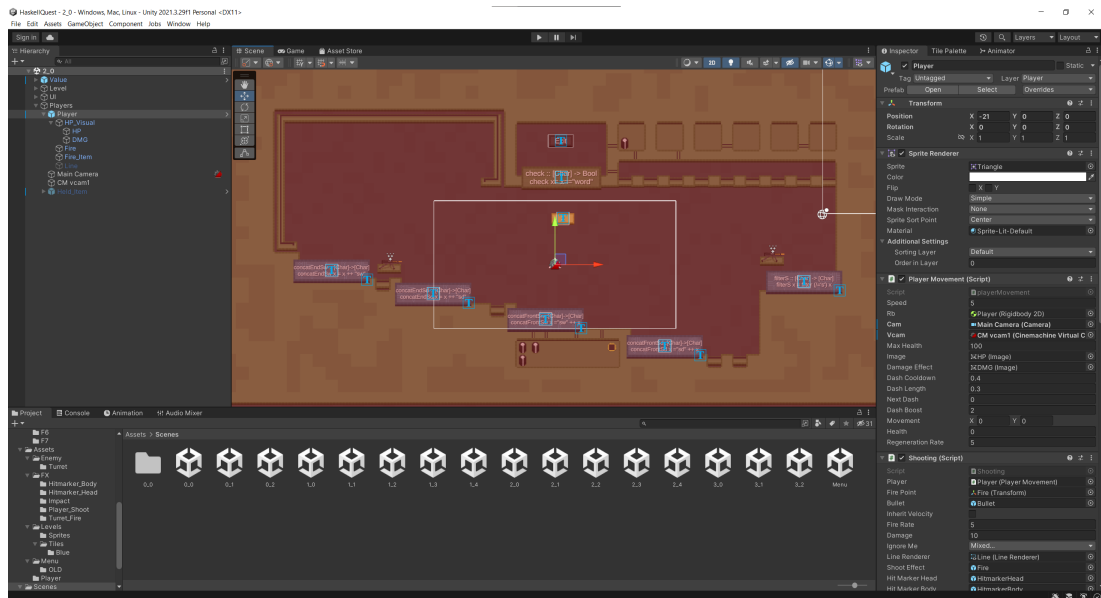


Figure 3.1: HaskellQuest in the Unity Editor.

Chapter 4

The Game

4.1 Game Idea

HaskellQuest is an action packed 2D puzzle game. Players, as they progress through all 4 chapters of the game each covering a specific aspect of Haskell, physically interact with values and functions, allowing them to get the immediate and individual feedback that serious games excel in. The key idea being that players need not know how to write Haskell code, yet they can still explore the language by playing around with values and functions to see what they do. Each level is more difficult than the last and either introduces a new topic or builds on previous knowledge. The challenges are designed to feature functions that students might encounter in their functional programming courses, such as square or the Fibonacci sequence, rather than some abstract “attacks”, etc. that other games rely on.

4.2 Mechanics

Values are represented by small yellow rectangles, with the value itself being shown on the rectangle. When a player goes over a value they pick it up and hold it until it is thrown. These values can be thrown into functions, represented by larger rectangles, which will modify the value in some way and then spit it out for the player to pick up again. The player must use this interaction between values and functions to get some sort of desired final value that can then be passed into a Boolean function, represented by a large dark green rectangle, that will disappear when the correct value is passed in. This allows the player to pass through and complete the level by touching a black “exit” block. If the value is incorrect or of the wrong type, it will simply bounce off. These physical interactions allow players to connect more with the language and be able to explore it in a way that otherwise would have been impossible without actually writing code. All of these elements can be seen in figure 4.1.

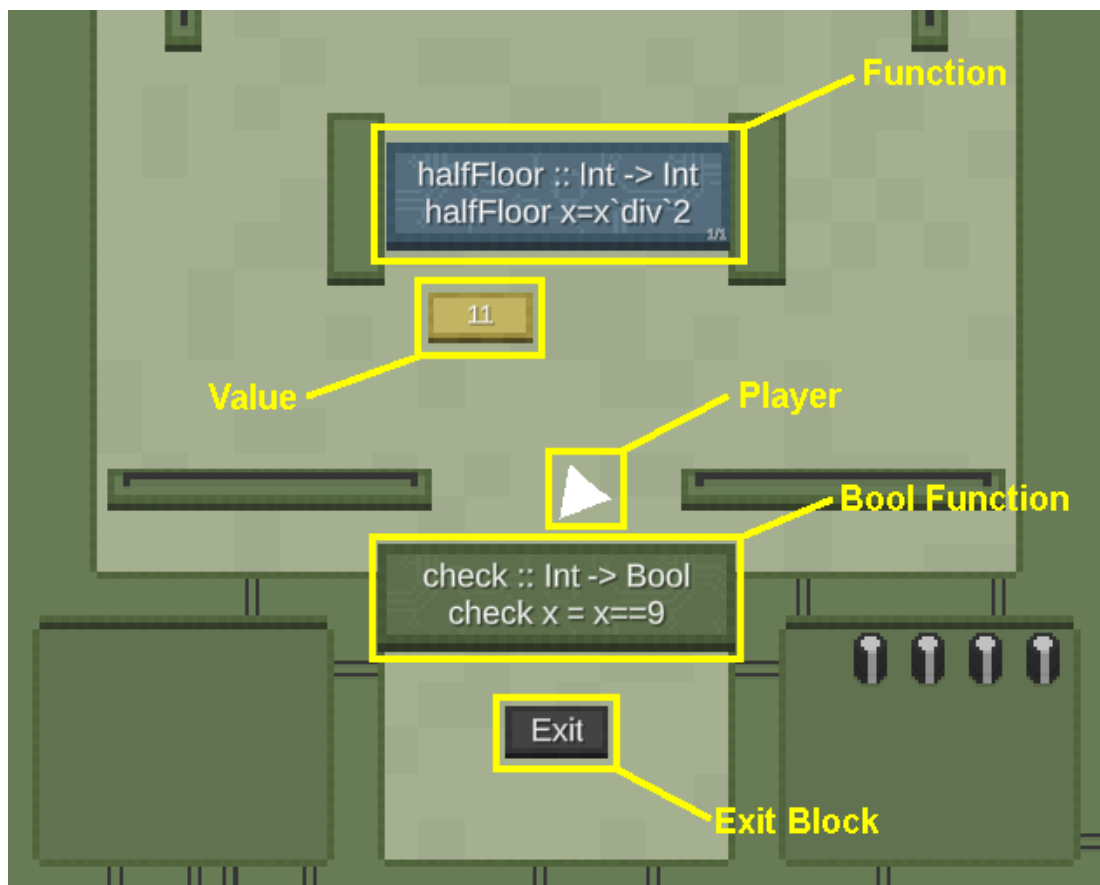


Figure 4.1: Various Elements of HaskellQuest puzzles.

Certain game levels have turrets sprinkled throughout them. These enemies fire upon the player when they get within a certain range, and they can be defeated by the player by shooting at them. They have a weak point in the form of a flashing red light, which will receive double the damage when hit. Another way to counter these turrets is to block line of sight between them and the player by throwing a yellow value rectangle in their direction. This was also be used in the opposite way by placing values next to enemies that players must pick up to complete the level. Upon picking up these values, enemies would start attacking the player as they would gain line of sight. Enemies, while providing no real educational value, play a key role in gamifying the experience of learning Haskell as they offer an exciting challenge to players - failing to take one down could result in losing a whole level's worth of progress. Their damage and health values are very generous, meaning they won't be frustrating to fight and lose to. Alongside this the player has quite swift health regeneration, meaning that they are not punished too hard for making mistakes while fighting them.

There is a timer that keeps track of how long a player takes to complete a level. It is saved and shown next to the level icon in the main menu. If a player beats their best time, the new faster time will be displayed instead. On top of this players are rewarded with medals that show them how well they did, as seen in 4.2. These two features work together to reward players for repeating levels, just like in the Duolingo case study. Players, while improving their time, simultaneously work towards reinforcing their

understanding of programming in Haskell. Players can choose to reset all their times in the options.



Figure 4.2: Level Selection with Personal Best Time and Medals

4.3 Download

The version of the game that was used for beta testing is currently available for download from Google Drive. It includes versions for Mac, Windows and Linux alongside a relevant README file for each one. You are welcome to try it out: <https://drive.google.com/drive/folders/1nsSHiBCI3c8oJre65YOmsC1yR3yVg1N5>

Chapter 5

Design

5.1 Level Design

From the very beginning it was clear that Level Design would play a key part in helping the game to fulfil its purpose. The game relies heavily on physical interactions between values and functions so each individual level would need to facilitate those interactions and guide players. Moreover, levels as a whole need to slowly ramp up in difficulty as they introduce ideas - not only new Haskell concepts but also fresh gameplay elements. Levels should also be visually interesting in order to fulfil the goal of gamification.

5.1.1 Puzzles

Individual levels are planned out using flowcharts produced in Smartdraw[38]. First of all, a general goal is decided upon for a level. This could either be introducing a new concept or testing the player's knowledge of a previously established ideas. Following that, there is a period of brainstorming and messing around with Haskell code to think up functions that will fulfil the level's goal. Finally, these Haskell functions are arranged in a flowchart alongside an initial value that'd be given to the player, both represented by rectangles. The final boolean function that tests the player's value is represented by a diamond, and the end of the level is represented by an ellipse.

An example of a flowchart for a level can be seen in figure 5.1. The value of 15 is passed into the three subsequent functions in the shown order. This will produce a value of 6 which is even and less than 8, as required by the boolean function. Note that this is not the only way to complete this particular level as `subEight` and `halfFloor` can be done in the opposite order, which will yield a value of 2 after adding 3. This diagram also shows the amount of times a function needs to be used for the solution alongside the total amount of times it can be used, in this example all the functions have 1(use needed)/1(total uses).

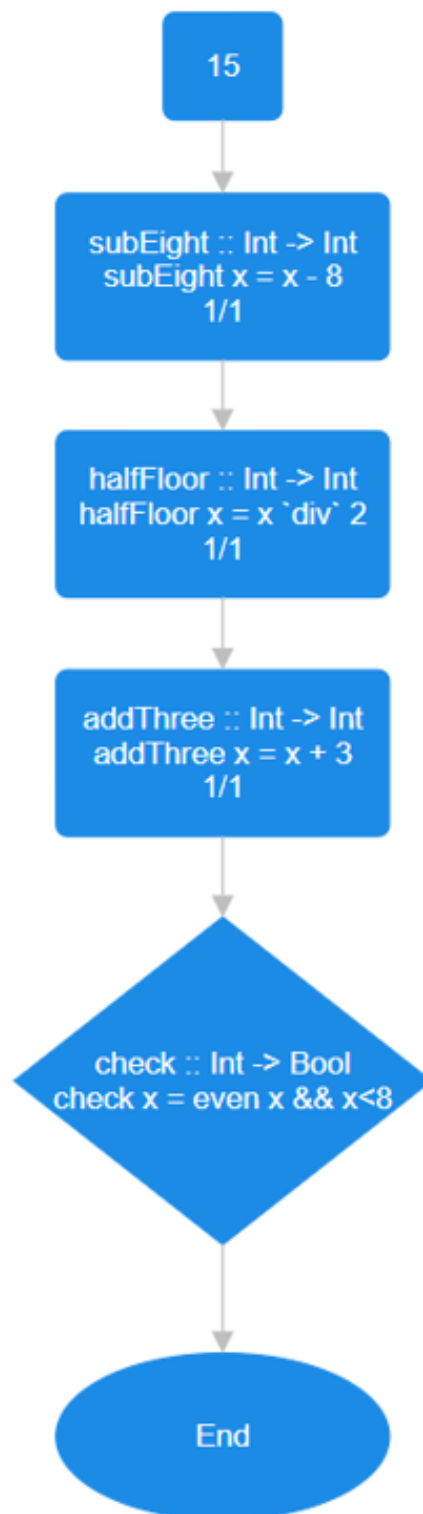


Figure 5.1: Level 1 Chapter 2 Flowchart.

5.1.2 Progression

The game is comprised of 16 progressively more difficult levels divided into 4 chapters, each covering a different aspect of Haskell. All 16 levels are unlocked and playable from the beginning, allowing players to skip anything they might already know directly to something new they want to learn.

Chapter 1 covers the very basics of Haskell. It allows the player to see very simple arithmetic operations done in Haskell, while introducing them to basic gameplay elements. The first level (seen in figure 5.2) acts as an easy introduction to the game as a whole, featuring only one function, one value and one enemy. With an almost trivial solution, the player only needs to pass the value of 2 into the square function once to get the desired value of 4. This offers a lightweight introduction to the basic gameplay of HaskellQuest - allowing players to get accustomed to enemies as well as the interactions between functions and values.



Figure 5.2: Level 1 Chapter 1 - Featuring the basic elements of the game including an enemy, a value, a basic square function and boolean function.

The second level introduces functions with limited uses, having 3 functions that can only be used once. These functions were necessary to prevent unintended solutions to levels. For example a hypothetical level featuring an `addOne` function as part of its intended solution and a goal value greater than the initial value would be trivial to do as the `addOne` function could be repeatedly used to eventually get to the goal value. Limited functions were also key in guiding the player to the correct solution as they give a clue as to how many times they must be used. In their absence, a player would have to consider infinite possibilities and combinations of functions to get the desired value. The third level shows the player an interesting bit of Haskell syntax, that infix operators such as addition can be written as a prefix function. The solution forces the

player to use 2 functions that both add 2 but are written in different ways, allowing them to experience and see that they are equivalent.

Chapter 2 serves as an introduction to Haskell's lists. The first level consists of a `[1,1]` starting value and a function that takes in that list and computes the subsequent elements in the Fibonacci sequence - a sequence that many would already be familiar with, allowing them to connect their past knowledge and see it from the perspective of a Haskell function. The fourth level was the first maze-like level in the game. These are designed to combine the puzzle element of the game with exploration - one of the most desired gameplay aspects chosen by respondents to the Functional Programming Education questionnaire. Maze levels are full of dead ends and ambushes which keeps the process of finding hidden functions and values exciting and engaging. Level 5 introduces the idea of red herring functions which are present in the level, but not part of the intended solution. In this particular level the unneeded functions are visually distinct as to give a hint as to which ones are required for the solution, being slightly skewed and lower down than the rest (figure 5.3). These functions introduced another layer of complexity to the game, forcing the player to consider which functions they actually need.

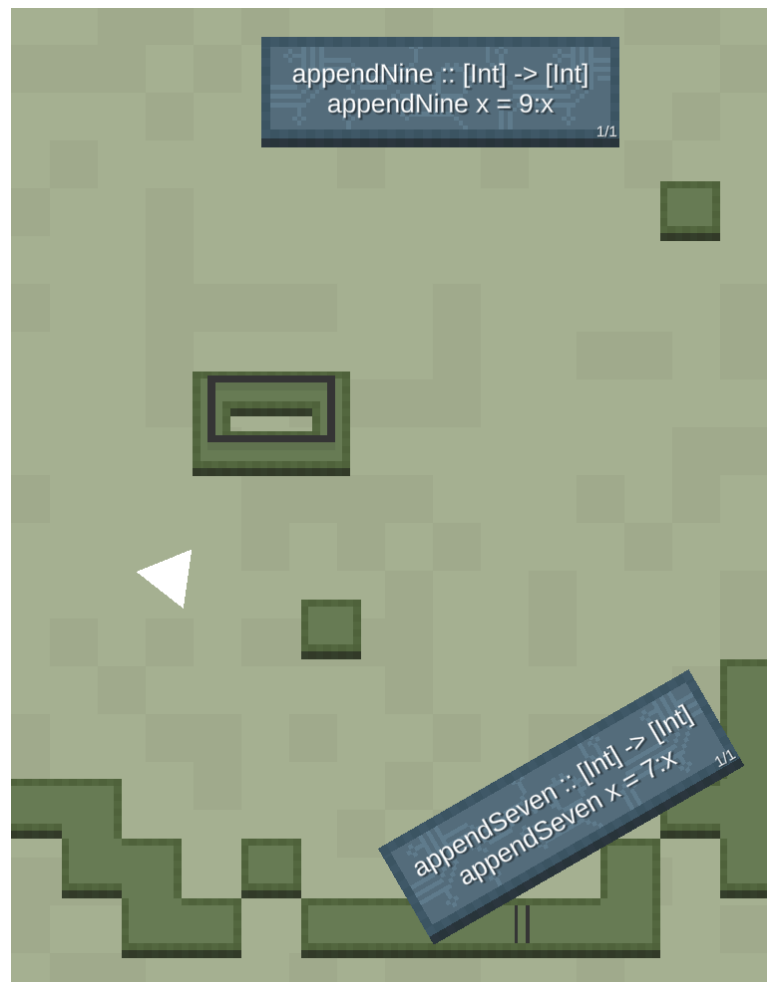


Figure 5.3: Level 5 Chapter 2 - The unneeded function is skewed and "laying" on the ground.

Chapter 3 focuses on map and filter. The first level introduces filter via a simple function that removes a specific character from a string, allowing the player to visually get an intuition for what filter does by seeing what letters get removed from a word. Level 2 features the first instances of functions using map. Subsequent levels combined the two with increasingly difficult puzzles, culminating in Level 5.

Chapter 4, added after the first alpha test after popular request, features monads. The first level shows the first IO function that produces an IO value, which cannot be picked up by players and can only be operated on by other IO functions. This level is also the first to feature multiple values, the correct one having to be chosen and passed into the single IO function to complete the level. Level 2 features a physical in-level depiction of a keyboard that represents users typing in an IO input by producing a value every couple seconds. In this level the keyboard is visible right from the start of the level as seen in figure 5.4, with the value flying out of it and straight into an IO function that modifies it. This allows the player to gain an understanding of what the keyboard is doing right from the start.

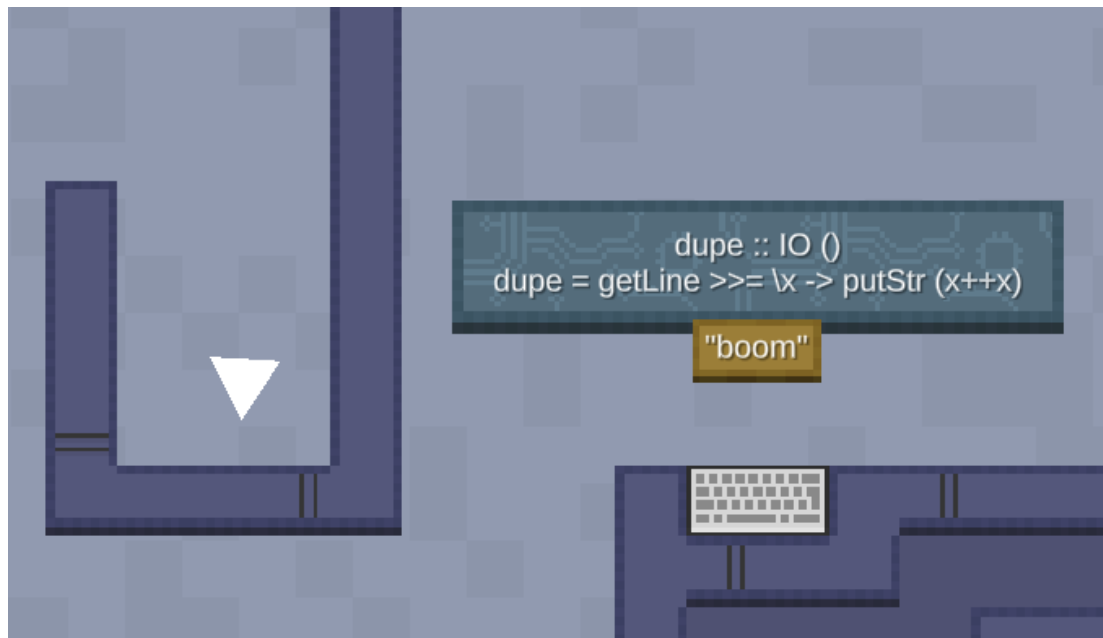


Figure 5.4: Level 2 Chapter 4 - Player, at the start position, being able to see the keyboard producing a value.

As the player passes through the level, they are eventually lead to a yellow IO function - one that can for the first time be picked up. This function needs to be placed in such a way that it captures the IO value from the previous function to produce the correct output. The third level builds on this idea of moving functions to modify an IO value, requiring the player to navigate a maze to find and chain together 5 of 8 functions, letter by letter, to assemble a string that says "monads". The solution to this level can be seen in figure 5.5.



Figure 5.5: Level 3 Chapter 4 Solution - Keyboard at the bottom periodically outputs an IO value that eventually becomes "monads".

5.2 User Interface

The goal with the game’s UI is to keep it minimalistic, as a reflection of the game’s conceptual simplicity. The interactive prototype was made in PowerPoint for each of the game’s menus (figure 5.6) as part of the game’s GDD, as seen in appendix C.

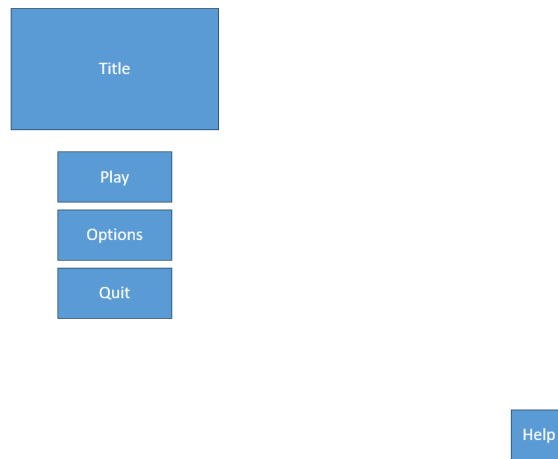


Figure 5.6: HaskellQuest main menu prototype.

The main menu features the game’s name alongside buttons to play, go to options, quit and go to the help screen. Selecting “Play” opens up the chapter selection screen (figure 5.7), and clicking a chapter allows for the selection of individual levels. Within a level there is no UI at all, allowing the player to focus on puzzles. Only after pressing ESC is the player given an option to quit the level.



Figure 5.7: HaskellQuest chapter selection screen.

Chapter 6

Production

Thanks to the game not relying on players writing code, the implementation does not require any in-game compilation of Haskell code. This greatly simplifies the production stage as, unlike previous related work [33][32], it is not necessary to utilize external compilers to run Haskell code.

6.1 Values & Functions

The value class holds one of six types of values: an integer, a bool, a list of either of the previous two, a string or a function. The value of the class is displayed on a text object in different ways depending on its type. For example string values are encapsulated in quotation marks, lists are separated by commas and surrounded by square brackets, etc. This can be seen in figure 6.1.

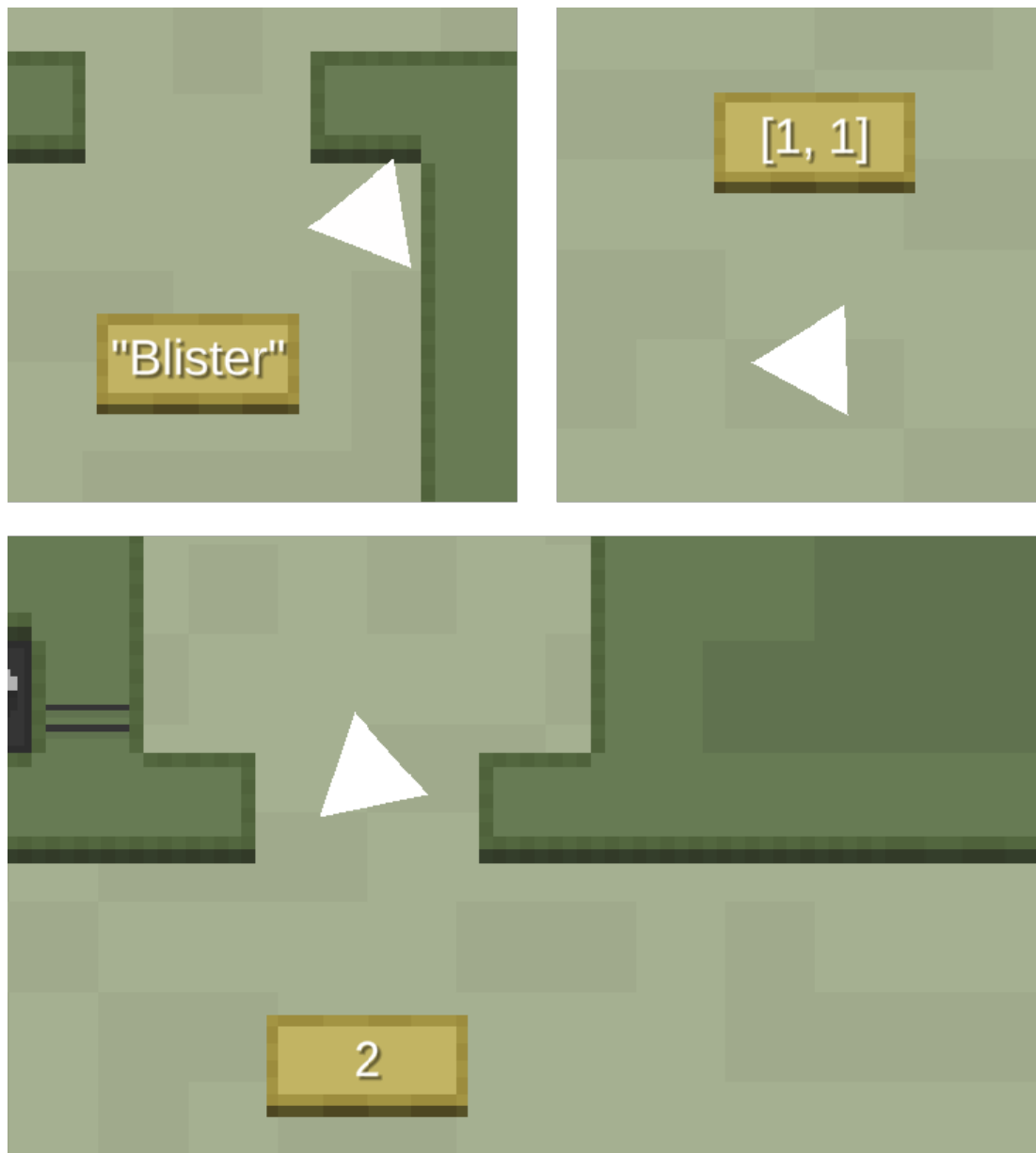


Figure 6.1: Different formatting of a value depending on its type.

All the functionality for the interactions facilitated by functions is implemented in one main class called `func`. This class checks for collision with a value in the `CollisionCheck` method. When such a collision takes place it checks for the type of the value and then calls an appropriate method to modify the value. These methods are virtual and are overwritten by other specific function scripts. These function scripts are C# translations of Haskell functions. For example `func_square` overrides `functionFloat` and modifies the `intVal` variable that holds the integer value to be equal to `intVal*intVal`. Once the value is modified, the `Shoot` method is called which instantiates a new value object with the updated value and spits it out the top of the function with a specified impulse and direction. This value has its collision disabled for a short amount of time to prevent it from being picked up again by the same function. The `func` class also contains a

modifiable health value that can be edited in the unity editor. By default it is -1 meaning that the function has infinite uses. When set to any positive integer, the health value is decremented with each use of the function. If the health value is equal to 0, the collision check is no longer valid and the value simply passes through the function.

6.2 The Player

The inventory class allows the player to pick up values and hold them. When colliding with a value, a check is performed to determine whether the player is already holding a value. If not, the value is destroyed shortly after its properties are copied over to the inventory class. The value is displayed above the character to show the player what they are currently holding. When the right mouse button is pressed, assuming the player is holding something, a value is instantiated in the world and populated with all related fields from the inventory. It is thrown with a specified impulse in the direction the player is looking. Just like with the function class, the value is put into a temporary state where it cannot collide with the player. This prevents it from being instantly picked up again when thrown. The inventory is then set to a state where it is holding nothing, ready for the player to pick up another value.

The Shooting class lets the player shoot bullets back at enemies by casting a ray when the left mouse button is pressed, alongside a few effects. If the ray intersects with an enemy, the enemy receives an appropriate amount of damage. The `playerMovement` class handles player movement alongside a few player attributes. This includes horizontal and vertical movement, dashing, pointing towards the mouse, player HP and zooming. When the player's HP drops it starts regenerating until it's back to full. If it drops below 0, the player loses and the level resets. The project uses the Cinemachine package[70] for Unity to smooth out the movement of the camera, that is attached to the player.

The enemy class implements the functionality for the turret. It constantly checks for line of sight with the player using a raycast. If the player is visible and within a certain range, the enemy shoots by instantiating bullet objects with a specific impulse in the direction of the player. When the enemy's health drops below 0, it is deleted.

6.3 Saving data

Saving data was one of the key functional requirements of the game since the start. It allows players to quit the game partway through and come back after a break, or perhaps when they've learned enough Haskell to help them in later levels.

The saving of data was not as difficult to implement as originally expected. Unity has a convenient way to save simple values (float, int & string) locally to the device via `PlayerPrefs`. This allows users to keep progress even after closing the game. In terms of `HaskellQuest`, `PlayerPrefs` are used to save which levels have been completed as an int value of either 0 for incomplete or 1 for complete. The player's best time is also stored as a float value. These can then be loaded whenever needed to display players' times and medals in the main menu. In the options there is a button to wipe all this

data in case a player wants to reset their progress for whatever reason. The same saving functionality is also used for the volume slider which is stored as a float value and used to determine the volume of the master channel.

Persistence between scenes is not really a concern for HaskellQuest. Each level is a standalone experience contained within a single scene, being free from the influenced of other puzzles. The only exception is the main menu which changes depending on how many levels the player has completed, but that is handled by the aforementioned PlayerPrefs.

6.4 Visuals

While not essential to teaching Haskell, a key part of gamifying the experience is making it look and feel like a game in order to keep people engaged. A little bit of time was spend creating the game's art from scratch instead of using any free packs from the Unity Asset Store. This is done not only so that the art is more consistent and thus nicer looking, but also allows for subconscious molding of players' expectations.

The game's art style is inspired by computer architecture and circuitry, linking to the theme of programming that it teaches. A lot of influence is taken from Synthetik 2[71], a roguelike[72] game that features a lot of enlarged electronics and computer parts in its levels. All sprites were created as pixelart [73] - retro low resolution textures that are quick to make and allow for more time to be spent on more important parts of development, while still keeping the visuals consistent and close to the original vision. The game uses Unity's Tilemap[74] system which streamlines the process of creating 2D levels. It allows for custom made pixelart tiles to be assigned to a palette and easily painted on a 2D grid. An example of one of HaskellQuest's palettes can be seen in figure 6.2.



Figure 6.2: One of two tile palettes used in HaskellQuest.

Visuals play a key role in forming the player's expectations. The key to this is color coding and consistency. All functions that perform operations are always blue, all boolean functions that perform checks are always a dark green, all values are always

yellow, etc. Popular game studio Naughty Dog famously always uses yellow to unconsciously guide players through their levels [75], consistently using it throughout their games to draw the player's attention to important parts of a complex 3D game world. Countless other developers use the color yellow in a similar manner. One such example is Digital Extremes' Warframe[76] where it's used as a marker for objectives that players must complete to finish the mission, as seen in figure 6.3. In Maxim Despinoy's HaskellQuest[32] yellow is used as the color for PC terminals that players need to access in order to complete Haskell puzzles, as seen in figure 6.4.



Figure 6.3: Warframe's yellow objective marker.



Figure 6.4: Yellow PC in Maxim Despinoy's HaskellQuest

In the context of HaskellQuest, the color yellow is used to signify to the player that something can be picked up. Throughout the game this expectation is formed and eventually comes into play in the final monads chapter in two ways. Up to this point the only pickupable values were simple values like integers or lists, but now the player is expected to pick up their first function. This function, being yellow just like all previous pickupable values, signals to players that it can be picked up as opposed to the previous blue functions that were static, as seen in figure 6.5. Another place color coding comes into play in the monads chapter is with IO values, which are output by in-level keyboards or IO functions. These unpickupable IO values are off-color, being a darker shade to signify that they cannot be interacted with.

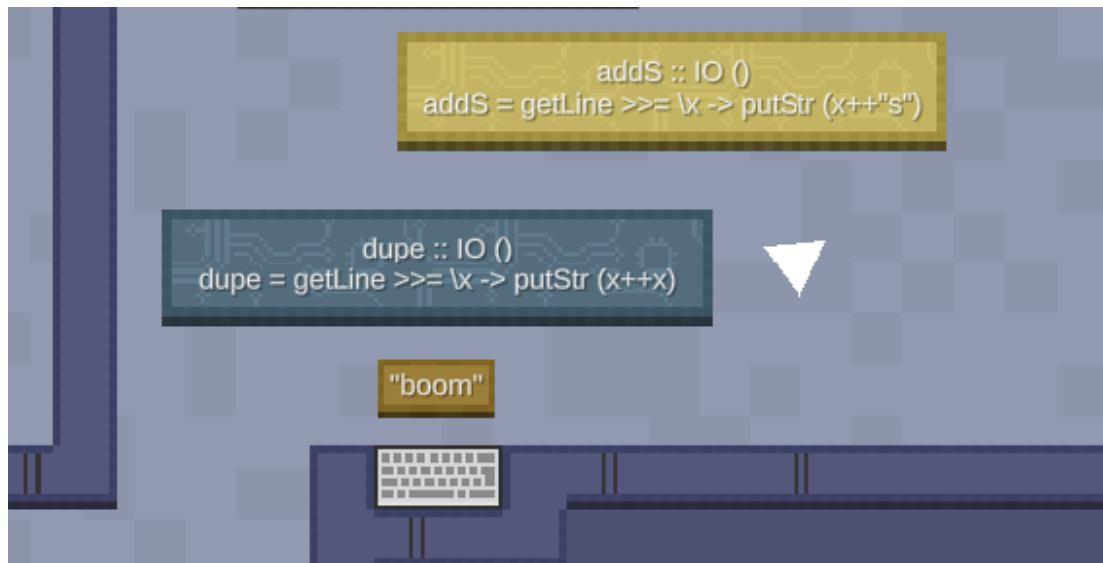


Figure 6.5: Yellow pickupable function next to a blue static function and off-color unpickupable string value

Chapter 7

Evaluation

The game was constantly being evaluated throughout development to ensure that it was meeting its central goal of being fun yet educational. Whenever a new feature is implemented, it is self tested to make sure it is up to standard. Informal feedback was received from colleagues, markers, and friends. Notably, project supervisor Don Sannella provided valuable feedback at various points throughout the project including finding issues with the level design in the design stage and finding bugs alongside providing feedback on the game itself before the alpha test. Watching roommates playing the game, who are unfamiliar with Haskell and programming in general, is a great insight into seeing how friendly the game is being for complete beginners.

The majority of the evaluation phase is focused on gathering user feedback from students that just took the INF1A course at the University of Edinburgh. User testing is split into two phases. Alpha testing focuses on testing an early version of the game to test the waters. It allows users to voice their opinions on what's already there, report bugs and make suggestions for future improvements in a survey. The game is then iterated upon in accordance with this user feedback to produce the final version of HaskellQuest. Beta testing, also using a survey for feedback, focuses on evaluating this final version to see if the changes made resulted in an improved game when compared to the previous iteration. Full responses to surveys can be seen in appendix B.

7.1 Alpha Testing

The alpha of the game includes all features required for the minimum viable product of the game, including 13 levels split into 3 chapters, one enemy type and a basic timer. There is no zoom-out feature, medals, or chapter about monads and all related features, such as the ability to hold multiple values or hold functions as values. Prior to the alpha phase, there was some uncertainty regarding running the game on Mac and Linux as the game was made and, until that point, ran exclusively on a Windows machine. This required consulting others and ensuring the game works on all systems, as well as preparing instructions for testers for running the game. Users are provided with two forms to fill out, a questionnaire for feedback and a bug report form to report any issues. All questions with a rating scale are from 1 to 6 to give less choice than a 1 to 10 scale

while also ensuring respondents have to choose between 3 (slightly negative) and 4 (slightly positive) instead of a scale like 1 to 5 which allows a neutral choice.

The questionnaire received 10 responses. On a scale of 1 to 6 users rate the game an average score of 4.7 in terms of how fun the game is, with only one vote being 3 or lower. On a scale of 1 to 6 the game receives an average of 4.8 in terms of its educational power, again with only one vote being 3 or lower. These results suggest that the game is on the right track to achieve its goal of being fun yet educational. A few questions are about the future of the game, with 40% of respondents saying the game doesn't cover enough aspects of Haskell. Participants suggest that the game should contain more levels and cover topics such as monads and currying. The participants are asked what they liked most about HaskellQuest and the answers vary a lot. Some compliment the "simple and fun" parts of the game like "the art style" and "moving around". Others like the approachable educational aspects with one participant saying "no coding required, very beginner friendly", and another enjoying the "interactions with functions". When asked about any additions they would like to see to the game, participants say that among other levels they would like "Some way of seeing more of each level such as a mini-map or the ability to zoom", and "more enemies".

The bug report form received only 2 responses, despite 3 people claiming to have encountered a bug in the survey, which was a big relief. One user mentions a bug with turret bullets being blocked by the block they sit on. The other describes a specific bug in the fifth level of the map chapter where inputting an empty list into a specific function would not produce an output.

7.2 Before Beta Testing

After the alpha test, the gathered feedback is used to improve the game. These improvements are split into three main parts.

7.2.1 Bug Fixes

The primary focus is fixing the issues encountered by testers. Turrets had their collision size decreased, it was needlessly large and they would hit themselves with their own projectiles. The aforementioned misbehaving function in the fifth level of the third chapter now functions properly. The issue was that the function was supposed to append four and then take the tail of the list, but it was doing those actions in the opposite order. The C# code was updated to more accurately reflect the Haskell function it was representing, by reversing the operations.

7.2.2 Responding to Feedback

The secondary priority was acting upon the feedback provided by testers. Many found it difficult to navigate some of the larger and more complex levels, especially with how little the player could see at one time. A simple "zoom out" functionality was implemented that zooms out the camera, allowing players to see more of their surroundings at one time. This approach was chosen over making something like a "minimap" – a small

map of the level implemented into the user interface, usually in the corner of the screen – to preserve the game’s minimal and simple feel. Some minimal changes to levels were also made, such as limiting the size of the third level in the third chapter. This level was needlessly large and deceived players who, up to this point were encouraged to explore, into thinking there would be something far from the centre. The “Escape” button was also added to the controls menu. For experienced gamers this button is synonymous with pausing a game but for those with less experience, it’s not exactly obvious.

The biggest change to the game was the addition of three levels (alongside related functionality) in the new fourth chapter about monads, which were requested since the first questionnaire about functional programming education. The idea for the monad levels is to abstract away keyboard input via a physical in-level “keyboard” that produces a specific IO value every 5 seconds. This value cannot be picked up by players, it can only be fed into IO functions that then produce another IO output. These functions have to be found in the level and then moved by the player so that they capture the keyboard input and produce some sort of desired output. For instance the final maze-like monad level requires the player to explore, find and chain together functions that each capture the input via `getLine` and use `putStr` to add another letter. From the very beginning of the level, the player can see the initial keyboard input of “” being passed into the `addM` IO function that adds ‘m’ to the input to show the player the basic idea of the level. On the way to this function the player picks up the “addO” function that adds ‘o’, which they must place after the M function they have been shown. They are then required to find the rest of the functions so that the final output of the last IO function is “monads”. There are a few functions in this level that are not needed in the final solution, so that players need to think about what they actually need.

This chapter required an extension to the value system that allowed them to hold a reference to a function `GameObject` that is instantiated alongside the base value `GameObject`. This allows players to manipulate functions just like they do with any other value. A minimal change was needed to the “inventory system” to support multiple values existing in a level, to prevent the player from picking up subsequent values if they were already holding one. The main menu UI was updated to feature the fourth monads chapter, which required adding a horizontal scroll bar as it wouldn’t fit otherwise. This chapter is accompanied with “New!” text in the corner to entice past testers to try it out in beta testing.

7.2.3 Fun changes

Aside from the major changes, there were also a few minor “fun” changes implemented to make the experience more gamified. A blue tile palette was added to be used in the monad levels to make them more visually interesting and distinct from previous challenges, emphasising how different these puzzles were compared to anything the player has seen before. Medals were added as a way for players to evaluate how well they did on a level. They are given depending on the player’s time and are displayed alongside the level in the level select screen. The times for these are chosen individually per level. The bronze medal time is always very generous and acts almost like a participation award encouraging those with even very slow times to give the level

another go to get silver or even gold. The gold medal time is slightly slower than a near perfect playthrough of the level. This is difficult to attain the first time around so it encourages players do several attempts of each level to get all of the gold medals. This is akin to maxing out lessons on Duolingo, reinforcing players' understanding while serving as a fun challenge.

Unfortunately, there was not enough time to act upon all feedback. Testers suggested more enemies and other aspects of Haskell such as currying but these could not be implemented due to strict time constraints.

7.3 Beta Testing

HaskellQuest was ready for beta testing after all the feedback from alpha testing was enacted. This phase mainly focuses on evaluating features added since the alpha test and ensuring the game overall is in a better state. Just as with alpha testing, testers are provided with a questionnaire to fill out for feedback. There is no bug report form this time and instead users are asked to comment about bugs in the final open ended question.

The questionnaire has 11 responses, 1 more than the alpha test. Since the feedback from both tests is anonymous, it's not possible to determine how many of these testers were also a part of the alpha test. To evaluate whether the updated game is better than the version used for alpha testing, some of the same questions are posed as on the alpha testing questionnaire. On a scale of 1 to 6 players give the game a 5.45 average fun rating, compared to 4.7 in the alpha test. The lowest rating is a singular vote for 4. When asked about how helpful the game would be to helping people understand the basics of Haskell on a scale of 1 to 6, respondents give an average score of 5.45 which is also an improvement over the 4.8 received in the alpha test. The majority of respondents voted 6. 81% of respondents say the game covered enough aspects of Haskell, when compared to only 60% in the alpha test.

There are a few questions that evaluate the new features added for the beta test. 72% of respondents say that medals did encourage them to give levels another go. 90% of testers say that the zoom functionality helps in navigating levels.

The questionnaire has a question about the visual appeal of the game, which is something that isn't evaluated in the first test. On a scale of 1 to 6, the average response is 5.0, with the lowest response being 3 and most responses being at 5 to 6. The survey concludes with an open ended question about the HaskellQuest experience. It allows testers to share anything from bug reports to future suggestions. This question has 6 responses and ranges from compliments like "cool game" and "medals were fun to get, i got all gold in the first 2 chapters" to feedback such as "i didnt know there was an option to zoom out when playing. it was hard to navigate some levels" and "HaskellQuest does not go in depth enough with Haskell but it'd be very good for absolute beginners".

Chapter 8

Conclusion

8.1 Main Achievements

Creating a successful programming game without requiring the players to write code was a huge success. At the beginning of the project I was highly doubtful as to whether my vision of a game that relied on physically interacting with values and functions would even be feasible let alone a good enough resource for learning a programming language. I could not find anything that was anywhere near the idea I had in my head as most programming games often opt to require users to write code. However from two rounds of user testing it's obvious that the game not only succeeded in being fun but also holds educational value, which I am very happy about.

Allowing players to save their progress was implemented successfully, which was a key requirement from the start of the project. Players can play the game at their own pace, take a break and come back to it whenever they want to.

This project was successfully finished on time without the project scope blowing up to be unreasonably large. There was a talk by a Rockstar employee at the University of Edinburgh in semester 2 that I attended about student game projects. One of the main points was that projects like this often blow up with endless ideas that can never realistically be done. This is something that HaskellQuest managed to successfully avoid – the original idea was simple enough for all of its main goals to be successfully achieved within the time given yet it was flexible enough to allow for further additions. Moreover there was more than enough time to allow for two phases of testing allowing the game to be refined.

Another achievement is gaining experience when working with game development. I gained skills by learning about and adhering to industry standards. Even mid-development I was already getting endless ideas on what I could add or do to improve on subsequent projects both from a technical and gameplay aspect. Learning about other projects that focused on “gamification” like Duolingo, which I personally have been using, was fascinating and gave me a real insight into how these sorts of projects work behind the scenes. With this game being based around Haskell, I had a chance to revisit the language and deepen my understanding of it. 3 years after doing INF1A I

had completely forgotten everything about monads.

8.2 Limitations

With values and functions occupying physical space in levels, the game unfortunately gets very ugly with larger functions. Even with the later monad levels it feels quite clunky to throw around and navigate between such large pick-up-able functions. With even longer functions the game would probably be very frustrating and confusing to play, which really limits the game's scope to covering small parts of Haskell.

More advanced Haskell features like passing functions into other functions would be very difficult to replicate in C#. The approach of rewriting functions is acceptable for the simple functions seen in the game that often require very basic operations, but does not scale very well. This could be remedied by compiling Haskell code directly as previous projects have done.

The system with timers and medals, while good for encouraging players to replay a level, is not the best way to quantify a player's understanding of Haskell. After a few playthroughs of a level a player will simply memorize what actions they must do to complete it. Unfortunately with the way the game is it's not possible to provide much useful feedback to the player like there would be with a game focused around writing code.

Many "behind the scenes" aspects of the project could have been handled better. For example in regards to the code, each function was usually associated with its own class that implemented its functionality. The issue was that a separate class would be made for e.g. "addThree" and "addTwo". These two could be merged into a singular, more modular class "add" that could be supplied a value to add from the Unity editor. Another example is poor utilisation of prefabs. Unity supports prefabs [77] allowing creation and storage of GameObjects which can then be reused throughout the game. HaskellQuest poorly utilizes this feature. For example the player camera is outside the main player prefab, which makes it difficult to make changes to the camera without having to manually apply the change to all levels. Furthermore the organisation of assets within the Unity Project is not done well. Most scripts are in one folder and sprites have inconsistent naming. These three factors might make it more difficult to build on top of the existing game and expand it in the future.

8.3 Future Features

Due to the limited time for the completion of this project there are many features that were either planned since the beginning or thought up throughout the development cycle that simply could not be implemented due to time constraints. They are listed below as an opportunity for future work to expand upon HaskellQuest.

8.3.1 Gamification

These are changes intended to make the game more fun and engaging:

- Enemy that constantly travels towards the player and explodes on contact.
- Variant of turret enemy that can not only shoot the player but also move around to gain line of sight.
- Enemy type that can itself hold a value, and drops it upon death.
- High risk high reward melee attack option for the player.
- Adding a story, one of the requested features from the functional programming questionnaire, would allow players to get further engaged and immersed in the game.
- More fun movement options, such as explosive barrels that could knock players and enemies around.
- Boss fight, where damage would be dealt by obtaining and then passing incorrect values into functions that'd cause runtime errors.
- More refined art direction including music and animations along with sounds - perhaps a fancy effect for the already existing medals that are rewarded on level completion.
- Currency as a reward for defeating enemies and in-game shop for buying character customizations.
- Rewards for completing chapters or getting a certain amount of medals such as the aforementioned character customization options.
- A roguelike version of the game featuring randomly generated levels that get progressively more difficult, bosses, currency and items.
- Multiplayer mode with new levels that'd require player cooperation to complete.

8.3.2 Education

Features focused on improving the game's educational value:

- More levels and chapters covering further aspects of Haskell, such as currying.
- Function history that would allow players to see, understand and then roll back their actions if needed without resetting the level completely.
- An option to toggle enemies, for those who want to take the game more slowly and focus on learning Haskell.
- Daily randomly generated levels featuring aspects of Haskell that the player has already covered in the game, to encourage them to come back and refresh their knowledge.

8.3.3 Other

Miscellaneous ideas that play no part in making the project more gamified or educational:

- Web and mobile versions of the game.
- Function health turns red when at zero, to make it more obvious that they're now unusable.
- Nicer user interface and allow for scaling based on monitor size.
- Allow the player to hold more than one value at a time.
- Extra step to confirm the resetting of a level upon pressing the R key, currently it is instant and can lead to issues if the player accidentally presses the key.
- Add a way to "freeze" thrown values so that they don't get accidentally picked up.
- Make thrown functions snap to a grid to make them easier to position - in the final monads level it was difficult to align them all at the correct distances from each other.

Bibliography

- [1] CodinGames <https://www.codingame.com/>, accessed: 20 Mar 2024.
- [2] Haskell <https://www.haskell.org/>, accessed: 12 Sept 2023.
- [3] Paul Hudak, John Hughes, Simon Peyton Jones, Philip Wadler, A History of Haskell: Being Lazy With Class, 16 Apr 2007, <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/history.pdf>, accessed: 13 Sept 2023.
- [4] PYPL PopularitY of Programming Language, <https://pypl.github.io/PYPL.html>, accessed: 13 Sept 2023.
- [5] Haskell Wiki, Haskell in Industry, https://wiki.haskell.org/Haskell_in_industry, accessed: 13 Sept 2023.
- [6] Haskell Cosmos, <https://haskellcosm.com/>, accessed: 15 Sept 2023.
- [7] Simon Marlow, Jon Purdy, Open-sourcing Haxl, a free library for Haskell, <https://engineering.fb.com/2014/06/10/web/open-sourcing-haxl-a-library-for-haskell/>, accessed: 16 Sept 2023.
- [8] Simon Marlow, Fighting spam with Haskell, <https://engineering.fb.com/2015/06/26/security/fighting-spam-with-haskell/>, accessed: 16 Sept 2023.
- [9] Coinbase, CardanoADA, <https://www.coinbase.com/en-gb/price/cardano>, accessed: 20 Sept 2023.
- [10] Elliot Hill, Medium, Why Cardano chose Haskell — and why you should care, <https://medium.com/@cardano.foundation/why-cardano-chose-haskell-and-why-you-should-care-why-cardano-chose-haskell>, accessed: 20 Sept 2023.
- [11] Moritz Andresen, Medium, The biggest smart contract hacks in history or how to endanger up to US \$2.2 billion, <https://medium.com/solidified/the-biggest-smart-contract-hacks-in-history-or-how-to-endanger-up-to-us-2-billion>, accessed: 21 Sept 2023.
- [12] David Z. Morris, Coindesk, CoinDesk Turns 10: 2016 - How The DAO Hack

- Changed Ethereum and Crypto, 9 May 2023 <https://www.gemini.com/uk#section-the-response-to-the-dao-hack>, accessed: 1 Sept 2023.
- [13] Haskell Wiki, Haskell in Education https://wiki.haskell.org/Haskell_in_education, accessed: 2 Sept 2023.
- [14] Aaron Contorer, Huffpost, the Language Most Likely to Change the Way you Think About Programming https://www.huffpost.com/entry/haskell-the-language-most_b_4242119 , accessed: 2 Sept 2023.
- [15] Edsger W. Dijkstra, To the members of the Budget Council, <https://chrisdone.com/posts/dijkstra-haskell-java/>, accessed: 3 Sept 2023.
- [16] VARK, VARK Modalities: What do Visual, Aural, Read/write Kinesthetic really mean? <https://vark-learn.com/introduction-to-vark/the-vark-modalities/> , accessed: 3 April 2024.
- [17] American Psychological Association, Belief in Learning Styles Myth May Be Detrimental, 30 May 2019, <https://www.apa.org/news/press/releases/2019/05/learning-styles-myth> , accessed: 3 April 2024.
- [18] Kevin Dickinson, Busting the learning styles myth: Why learning generalists perform best, 10 Nov 2021, <https://bigthink.com/the-learning-curve/learning-styles-myth/> , accessed: 3 April 2024.
- [19] L Rachel Paraiso, Busting the learning styles myth: Why learning generalists perform best, 17 Oct 2020, <https://elearningindustry.com/multimodal-meaning-connecting-multiple-intelligences-to-learning-games> , accessed: 3 April 2024.
- [20] Juho Hamari, Gamification, <https://onlinelibrary.wiley.com/doi/10.1002/9781405165518.wbeos1321> , accessed: 29 Aug 2023.
- [21] Tim Laning, GrendelGames, What are serious games, <https://grendelgames.com/what-are-serious-games/> , accessed: 27 Aug 2023.
- [22] Microsoft Flight Simulator, <https://www.flightsimulator.com/> , accessed: 28 Aug 2023.
- [23] Duolingo, <https://www.duolingo.com> , accessed: 15 Aug 2023.
- [24] Duolingo Courses, <https://www.duolingo.com/courses> , accessed: 15 Aug 2023.
- [25] Duolingo, Google Play Store, <https://play.google.com/store/apps/details?id=com.duolingo> , accessed: 16 Aug 2023.
- [26] Duoplanet, Matt, 100 of the best (and weirdest) duolingo sentences ever, <https://duoplanet.com/100-of-the-best-and-weirdest-duolingo-sentences-ever/> , accessed: 16 Aug 2023.

- [27] Youtube, Livakivi, Why I'm Quitting the Japanese Duolingo Course (An Honest Review), <https://www.youtube.com/watch?v=jf-SbSfiXn4&t=180s> , accessed: 16 Aug 2023.
- [28] Duolingo Blog, Beth Chasse, Taking a crack at gamification, 27 Jul 2021, <https://blog.duolingo.com/gamification-design/> , accessed: 17 Aug 2023.
- [29] Duolingo Blog, Osman Mansur, The habit-building research behind your Duolingo streak, 31 Jan 2022, <https://blog.duolingo.com/how-duolingo-streak-builds-habit/> , accessed: 17 Aug 2023.
- [30] ScienceDirect, Marissa A. Sharif, Suzanne B. Shu, Nudging persistence after failure through emergency reserves, <https://www.sciencedirect.com/science/article/pii/S0749597818304187> , accessed: 17 Aug 2023.
- [31] KnowYourMeme, Evil Duolingo Owl, <https://knowyourmeme.com/memes/evil-duolingo-owl> , accessed: 17 Aug 2023.
- [32] Maxim Despinoy. Having fun learning — A hidden component to success. 2023.
- [33] Eve Bogomil. HaskellQuest: a game for teaching functional programming in Haskell. 2023.
- [34] ScalaQuest: The Scala Adventure by Alejandro Lujan, YouTube <https://www.youtube.com/watch?v=jCyp9-GbXvE> , accessed: 3 April 2024.
- [35] G2, Devin Pickell, The 7 Stages of Game Development, <https://www.g2.com/articles/stages-of-game-development> , accessed: 18 Sept 2023.
- [36] digitaltrends, Tomas Franzese, Every video game delay that happened in 2023, <https://www.digitaltrends.com/gaming/all-2023-video-game-delays/> , accessed: 18 Sept 2023.
- [37] Medium, Nilesh Parashar, What is Game Design Document?, <https://medium.com/@niitwork0921/what-is-game-design-document-f00b986781c2> , accessed: 18 Sept 2023.
- [38] SmartDraw, <https://www.smartdraw.com/> , accessed: 18 Sept 2023.
- [39] TinyHydra, Arun Chapman, Vertical Slice vs MVP: What's the Difference?, <https://tinyhydra.com/vertical-slice-vs-mvp/#what-is-an-mvp> , accessed: 20 Sept 2023.
- [40] Monday Blog, What vertical slicing is and how it's used in project management, <https://monday.com/blog/project-management/vertical-slice/> , accessed: 20 Sept 2023.
- [41] Kevuru Games, Alena Porokh, Main Stages of Video Game Testing from Concept to Perfection, <https://kevurugames.com/blog/main-stages-of-video-game-testing-from-concept-to-perfection/> , accessed: 20 Sept 2023.
- [42] Perforce, The Complete Game Engine Overview, <https://www.perforce.com/resources/vcs/game-engine-overview> , accessed: 1 Oct 2023.

- [43] StudyTonight, Game Engine and History of Game Development, <https://www.studytonight.com/3d-game-engineering-with-unity/game-engine>, accessed: 2 Oct 2023.
- [44] Unity, <https://unity.com/>, accessed: 2 Oct 2023.
- [45] Unreal Engine, <https://www.unrealengine.com/en-US/>, accessed: 2 Oct 2023.
- [46] Godot Engine, <https://godotengine.org/>, accessed: 2 Oct 2023.
- [47] RPG Maker, <https://www.rpgmakerweb.com/>, accessed: 2 Oct 2023.
- [48] Game Maker, <https://gamemaker.io/en>, accessed: 2 Oct 2023.
- [49] Tech Target, Rahul Awati, Role-playing game (RPG), <https://www.techtarget.com/whatis/definition/role-playing-game-RPG>, accessed: 6 Oct 2023.
- [50] Program Ace, Unity vs. Unreal: What to Choose for Your Project?, <https://program-ace.com/blog/unity-vs-unreal/>, accessed: 6 Oct 2023.
- [51] Fortnite, <https://www.fortnite.com/>, accessed: 6 Oct 2023.
- [52] Borderlands, <https://borderlands.2k.com/en-GB/borderlands-3/>, accessed: 6 Oct 2023.
- [53] Sea of Thieves, <https://www.seaofthieves.com/>, accessed: 6 Oct 2023.
- [54] ARK, <https://playark.com/>, accessed: 6 Oct 2023.
- [55] Introduction to Blueprints, Epic Games, <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/>, accessed: 7 Oct 2023.
- [56] Unturned, Steam, 3.22.21.0 Update Notes, <https://store.steampowered.com/news/app/304930/view/5760697935049329373>, accessed: 11 Oct 2023.
- [57] Steam, Unturned 3.20.7.0 Update Notes, 17 Jul 2020, <https://store.steampowered.com/news/app/304930/view/2749961586453033838> , accessed: 26 Feb 2024.
- [58] Steam, Unturned 3.22.21.0 Update Notes, 16 Dec 2022, <https://store.steampowered.com/news/app/304930/view/5760697935049329373> , accessed: 26 Feb 2024.
- [59] Kevuru Games, Unity - What Makes it the Best Game Engine?, <https://kevurugames.com/blog/unity-what-makes-it-the-best-game-engine>, accessed: 11 Oct 2023.
- [60] Unity Forum, <https://forum.unity.com/>, accessed: 11 Oct 2023.
- [61] Youtube, Brackeys, <https://www.youtube.com/@Brackeys/videos>, accessed: 12 Oct 2023.
- [62] Unity Learn, <https://learn.unity.com/>, accessed: 12 Oct 2023.

- [63] Unity Asset Store, <https://assetstore.unity.com/>, accessed: 12 Oct 2023.
- [64] 6sense, Unity Market Share, <https://6sense.com/tech/game-development/unity-market-share>, accessed: 12 Oct 2023.
- [65] HackerNoon, DeveloperNation, Unity, Unreal Remain Popular Game Engines Among Developers But What Do Indie Studios Use?, <https://hackernoon.com/unity-unreal-remain-popular-game-engines-among-developers-but-what-do-indie-studios-use>, accessed: 12 Oct 2023.
- [66] Hoyoverse, Genshin Impact, <https://genshin.hoyoverse.com/en/>, accessed: 13 Oct 2023.
- [67] ActivePlayer, Genshin Impact Live Player Count and Statistics, <https://activeplayer.io/genshin-impact/>, accessed: 25 Oct 2023.
- [68] HelpLama, Among Us Usage and Statistics 2023, <https://helplama.com/among-us-usage-and-statistics/>, accessed: 25 Oct 2023.
- [69] ActivePlayer, Escape from Tarkov Live Player Count and Statistics, <https://activeplayer.io/escape-from-tarkov/>, accessed: 25 Oct 2023.
- [70] Cinemachine, <https://unity.com/unity/features/editor/art-and-design/cinemachine>, accessed: 26 Feb 2024.
- [71] Synthetik 2, https://store.steampowered.com/app/1471410/SYNTHETIK_2, accessed: 26 Feb 2024.
- [72] Masterclass, Roguelike Games: 5 Elements of Roguelike Video Games, 14 Sep 2022, https://store.steampowered.com/app/1471410/SYNTHETIK_2, accessed: 26 Feb 2024.
- [73] TheMotionMonkey, Pixel Art Definition, <https://www.themotionmonkey.co.uk/definitions/pixel-art/>, accessed: 26 Feb 2024.
- [74] Unity Learn, Introduction to Tilemaps, <https://learn.unity.com/tutorial/introduction-to-tilemaps/>, accessed: 26 Feb 2024.
- [75] GameRant, Naughty Dog's Obsession With Yellow Explained, Martin Wood, 25 Jul 2022, <https://learn.unity.com/tutorial/introduction-to-tilemaps/>, accessed: 01 Feb 2024.
- [76] Warframe, <https://www.warframe.com/>, accessed: 3 April 2024.
- [77] Unity Documentation, Prefabs, <https://docs.unity3d.com/Manual/Prefabs.html>, accessed: 25 Dec 2023.
- [78] Neil Fleming, David Baume, et al. Learning styles again: Varking up the right tree! *Educational developments*, 7(4):4, 2006.
- [79] Neil D Fleming and Colleen Mills. Not another inventory, rather a catalyst for reflection. *To improve the academy*, 11(1):137–155, 1992.

- [80] Shaylene E Nancekivell, Priti Shah, and Susan A Gelman. Maybe they're born with it, or maybe it's experience: Toward a deeper understanding of the learning style myth. *Journal of Educational Psychology*, 112(2):221, 2020.
- [81] Philip M Newton and Atharva Salvi. How common is belief in the learning styles neuromyth, and does it matter? a pragmatic systematic review. In *Frontiers in Education*, volume 5, page 602451. Frontiers, 2020.
- [82] Ivica Nikolić, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th annual computer security applications conference*, pages 653–663, 2018.

Appendix A

Questionnaire

A.1 Functional Programming Education Questionnaire

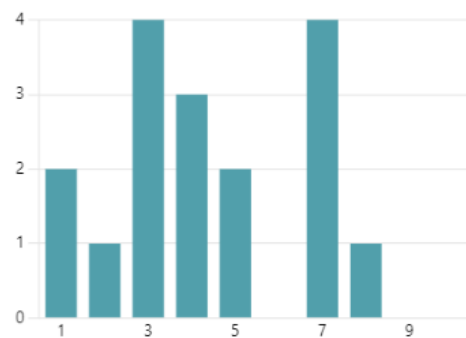
1. Have you done any functional programming before INF1A?

● Yes 3
● No 14

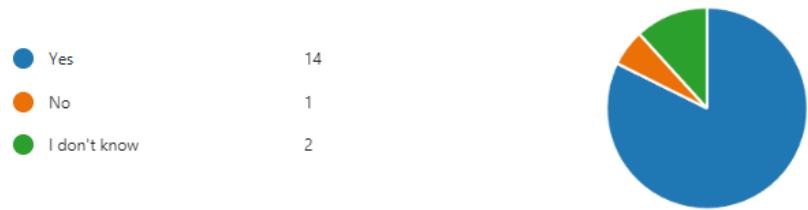


2. How difficult did you find Haskell when studying it? [1 = Easy, 10 = Hard]

4.35
Average rating



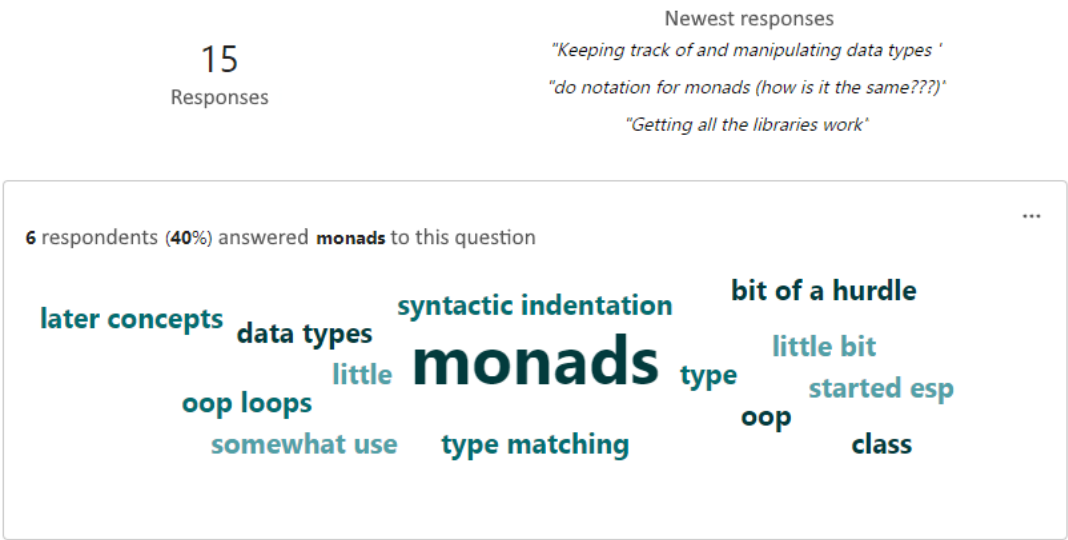
3. Do you think Haskell helped your understanding of programming in general?



4. Are you happy to have learned Functional programming? (as opposed to exclusively learning Object Oriented programming e.g. python,java)



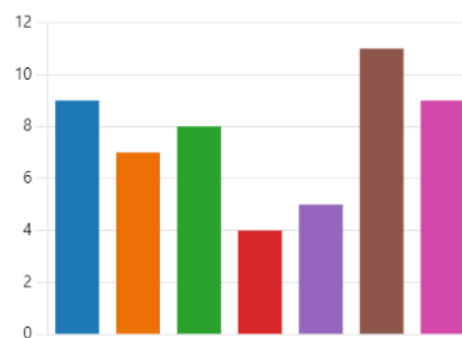
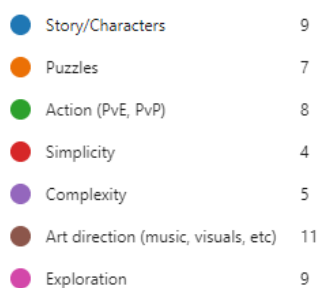
5. What is the most difficult part of Haskell?



Full responses to question 5:

1	anonymous	monads
2	anonymous	No variables was a little bit of a hurdle but honestly it was quite digestable
3	anonymous	Thinking without oop loops
4	anonymous	Monads
5	anonymous	type matching and getting the hang of that.
6	anonymous	The later concepts
7	anonymous	Monads. Despite covering them in class, I'm not sure I understand them.
8	anonymous	getting started esp wk 1-4
9	anonymous	I don't really get what monads are, though I can somewhat use them.
10	anonymous	syntactic indentation
11	anonymous	monads
12	anonymous	Fixing the errors because the logic is a little different to OOP.
13	anonymous	Getting all the libraries work
14	anonymous	do notation for monads (how is it the same???)
15	anonymous	Keeping track of and manipulating data types

6. What do you enjoy most in games? [Multiple options]



A.2 Participants' information sheet

Page 2 of 3

Specify:

- Kinds of data being collected: experience with learning Haskell & opinion about games, no personal data.
- Means of collection: questionnaire
- Duration of session: approximately 5min, depended on participant
- If participant audio/video is being recorded: No
- How often, where, when: Once, online.

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

No

What will happen to the results of this study?

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

Data protection and confidentiality.

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

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

What are my data protection rights?



THE UNIVERSITY of EDINBURGH
informatics

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

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Daniel Segboer s2080162@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>.

[NB: the PI should notify the Ethics panel on inf-ethics@ed.ac.uk to upload any updated PIS to the website]

Alternative formats.

To request this document in an alternative format, such as large print or on coloured paper, please contact Daniel Segboer s2080162@ed.ac.uk.

General information.

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

A.3 Participants' consent form

Participant number: _____

Participant Consent Form

Project title:	HaskellQuest
Principal investigator (PI):	Don Sannella
Researcher:	Daniel Segboer
PI contact details:	Don.sannella@ed.ac.uk

By participating in the study you agree that: [

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

Please tick yes or no for each of these statements.

1. I allow my data to be used in future ethically approved research.

<input type="checkbox"/>	<input type="checkbox"/>
Yes	No

2. I agree to take part in this study.

<input type="checkbox"/>	<input type="checkbox"/>
Yes	No

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

Appendix B

Testing

B.1 Alpha Testing Feedback Form

1. I allow my data to be used in future ethically approved research

● Yes	10
● No	0



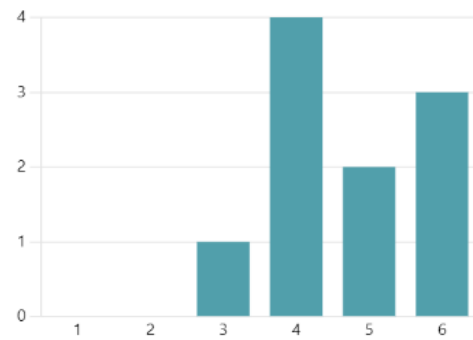
2. I agree to take part in this study.

● Yes	10
● No	0



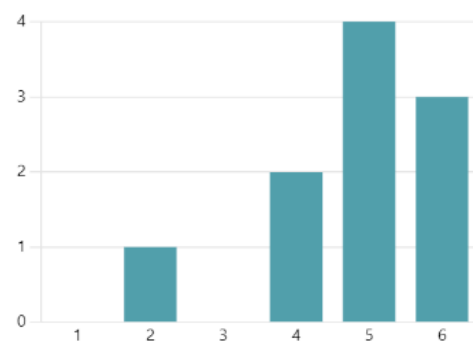
3. On a scale of 1 to 6, how fun was HaskellQuest? [1 = Not Fun at All, 6 = Very Fun!]

4.70
Average rating

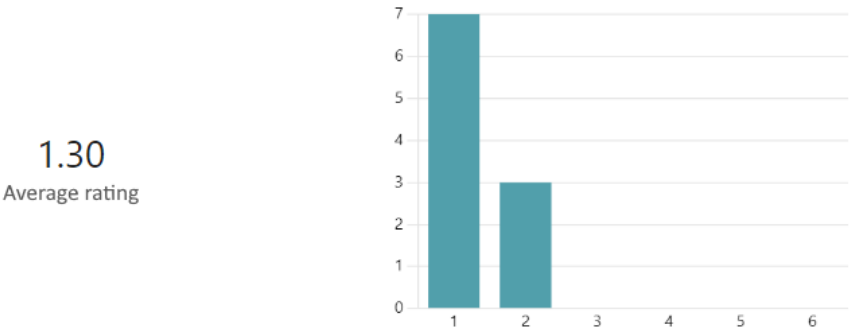


4. On a scale of 1 to 6, how helpful do you think HaskellQuest would be to helping people understand the basics of Haskell? [1 = Not at All, 6 = Very Much so]

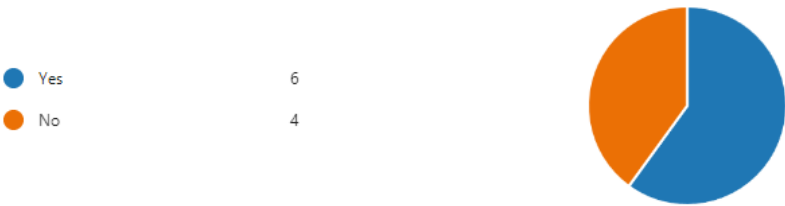
4.80
Average rating



5. On a scale of 1 to 6, how much bugs and issues did you encounter during your time playing? [1 = None, 6 = Many] Please report any issues on the Bug Report form
- https://forms.office.com/Pages/ResponsePage.aspx?id=sAafLmkWiUWHiRCgaTTcYeZ_toS2kWplq49Soc4ClgxURVJLN1A5N1lEWtNSWTc2SVJM000U5QkZOUi4u



6. Does the game cover enough aspects of Haskell?



7. If not, what parts of Haskell should future chapters cover?

3

Responses

Newest responses

Full responses to question 7:

1	anonymous	carry, monad
2	anonymous	monad
3	anonymous	Monad

8. What do you like most about HaskellQuest?

5

Responses

Newest responses

"no coding required, very beginner friendly"

1 respondents (20%) answered **beginner friendly** to this question

...

input **beginner friendly** **art style**
Simple and fun **interactions with functions**

9. Is there anything else you would do to improve HaskellQuest?

5

Responses

Newest responses

"More levels covering topics like monads"

3 respondents (60%) answered **level** to this question

...

monads **mini** **level** **topics**
ability **map** **shooting function**
way **enemies** **function in our character**
excellent

Full responses to question 8:

1	anonymous	The art style
2	anonymous	The moving around and throwing the input
3	anonymous	interactions with functions
4	anonymous	Simple and fun!
5	anonymous	no coding required, very beginner friendly

Full responses to question 9:

1	anonymous	Some way of seeing more of each level such as a mini-map or the ability to zoom
2	anonymous	I didn't really understand what the shooting function in our character could be used for, but other than that, excellent!
3	anonymous	more levels
4	anonymous	more enemies
5	anonymous	More levels covering topics like monads

B.2 Alpha Testing Bug Report Form

1. Describe the bug alongside any details (where it occurred, how to reproduce it, etc.)

2

Responses

Newest responses

"the fourTail function in Map 5 does not return the block after an empty str..."


"Turret bullets are blocked by the left edge of the block they sit on."


2. If possible, please upload any relevant screenshots.

2

Responses

Newest responses

 Screenshot 2024-01-31 at 14.10.35_anonymous.png

 Screenshot 2024-01-30 191533_anonymous.png

Full responses to question 1:

1	anonymous	Turret bullets are blocked by the left edge of the block they sit on.
2	anonymous	the fourTail function in Map 5 does not return the block after an empty string is input (presumably because the function is not made to handle empty lists), but after one pass through "mapDiv" and "filt" an empty list can be produced, so a way to handle that error would improve gameplay

Screenshots provided in question 2:

B.3 Beta Testing Feedback Form

1. By participating in the study you agree that:
- I have read and understood the Participant Information Sheet (attached to email) 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.

● Yes	11
● No	0



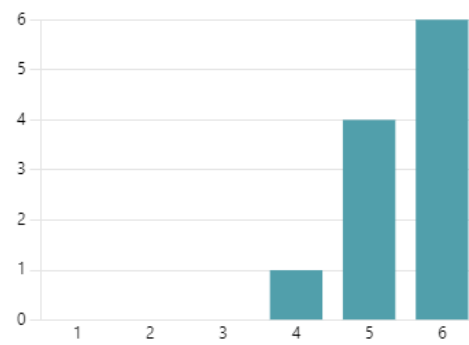
-
2. I agree to take part in this study.

● Yes	11
● No	0



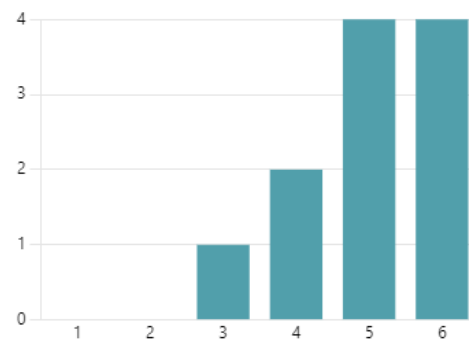
3. On a scale of 1 to 6, how fun was HaskellQuest? [1 = Not Fun at All, 6 = Very Fun!]

5.45
Average rating



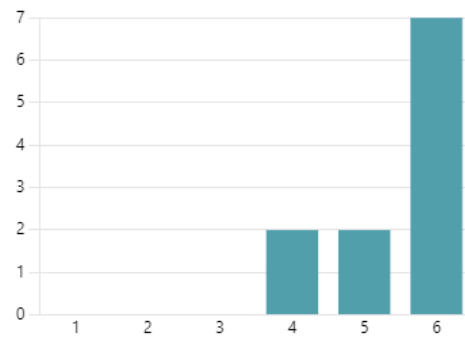
4. On a scale of 1 to 6, how visually appealing is HaskellQuest? [1 = Not at All, 6 = Very Much so]

5.00
Average rating



5. On a scale of 1 to 6, how helpful do you think HaskellQuest would be to helping people understand the basics of Haskell? [1 = Not at All, 6 = Very Much so]

5.45
Average rating



6. Does the game cover enough aspects of Haskell?

● Yes 9
● No 2

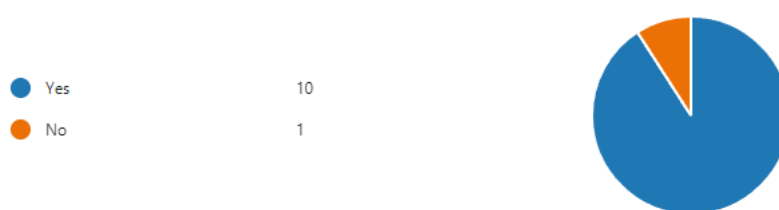


7. Did medals encourage you to replay levels and beat your time?

● Yes 8
● No 3



8. Did the zoom functionality (Left Shift) help you with navigation?



9. [Open Ended] Is there anything else you want to say about your experience? Did you encounter any bugs? Would you like to see anything added to HaskellQuest? What aspects of Haskell could the game cover in the future?

6
Responses

Newest responses
"HaskellQuest does not go in depth enough with Haskell but it'd be very g..."

2 respondents (33%) answered **absolute beginners** to this question

absolute beginners
levels game
good fun

Full responses to question 9:

1	anonymous	i didnt know there was an option to zoom out when playing. it was hard to navigate some levels
2	anonymous	I like that I didn't have to actually write any code, this game would defo be good for absolute beginners!
3	anonymous	medals were fun to get, i got all gold in the first 2 chapters.
4	anonymous	wish there were more levels the concept is very fun
5	anonymous	cool game
6	anonymous	HaskellQuest does not go in depth enough with Haskell but it'd be very good for absolute beginners.

B.4 Participants' information sheet

Page 1 of 3

Participant Information Sheet

Project title:	HaskellQuest
Principal investigator:	Don Sannella
Researcher collecting data:	Daniel Segboer
Funder (if applicable):	

This study was certified according to the Informatics Research Ethics Process, reference number 715532. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

This is an undergraduate research project by Daniel Segboer. It is supervised by Don Sannella.

What is the purpose of the study?

The purpose of this study is to create a game that helps people grasp the basic concepts of Haskell and Functional programming in general. The results of the form will help me understand what students enjoyed about the game in its current state, and help iterate on it.

Why have I been asked to take part?

Students that have taken the INF1A course that taught functional programming are the research target group.

Do I have to take part?

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

What will happen if I decide to take part?

Specify:

- Kinds of data being collected: experience with and opinions about Haskell game, no personal data.
- Means of collection: questionnaire
- Duration of session: approximately 10min, depended on participant
- If participant audio/video is being recorded: No
- How often, where, when: Once, online.

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

No

What will happen to the results of this study?

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

Data protection and confidentiality.

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

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

What are my data protection rights?

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

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Daniel Segboer s2080162@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>.

[NB: the PI should notify the Ethics panel on inf-ethics@ed.ac.uk to upload any updated PIS to the website]

Alternative formats.

To request this document in an alternative format, such as large print or on coloured paper, please contact Daniel Segboer s2080162@ed.ac.uk.

General information.

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



B.5 Participants' consent form

Participant number: _____

Participant Consent Form

Project title:	HaskellQuest
Principal investigator (PI):	Don Sannella
Researcher:	Daniel Segboer
PI contact details:	Don.sannella@ed.ac.uk

By participating in the study you agree that: [

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

Please tick yes or no for each of these statements.

1. I allow my data to be used in future ethically approved research.

<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------

Yes No

2. I agree to take part in this study.

<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------

Yes No

Name of person giving consent

Date
dd/mm/yy

Signature

Name of person taking consent

Date
dd/mm/yy

Signature

Appendix C

HaskellQuest Game Design Document

User Interface

Ingame & Menus

Title

Play

Options

Help

Options

Option 1

Option 2

Option 3

Return

Chapter Select

Chapter 1

Chapter 2

Chapter 3

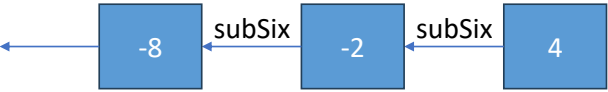
Return

Reset



```
subSix :: Int -> Int
subSix x = x - 6
```

2/4



Level Select

Level 1

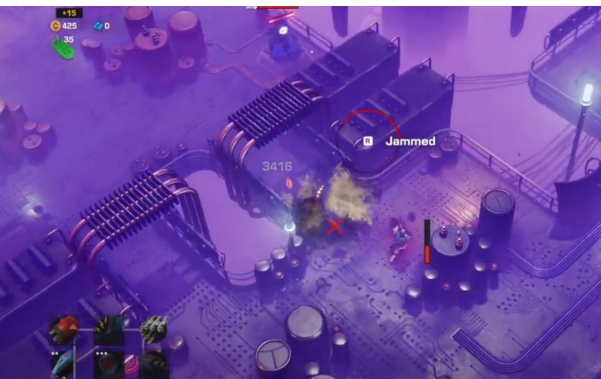
Level 2

Level 3

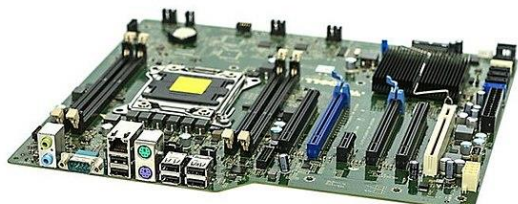
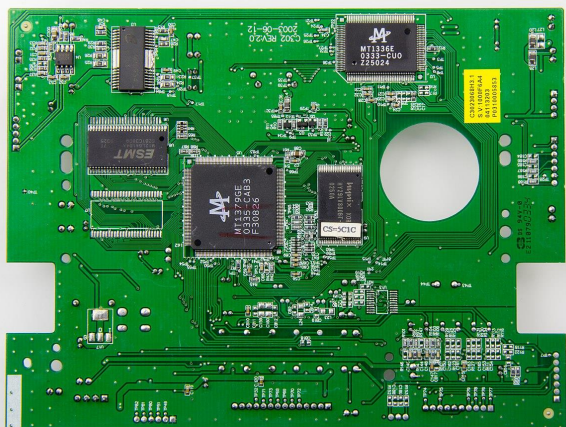
Return

Aesthetic Inspiration

PC Hardware



```
[root@localhost ~]# ping -i 10 text.patpa.wikimedia.org
PING text.patpa.wikimedia.org (208.80.152.2): 56(84) bytes of data:
6 packets transmitted, 6 received, 0% packet loss, time 6ms
rtt min/avg/max/mdev = 540.529/540.529/540.529/0.000 ms
[root@localhost ~]# pwd
/root
[root@localhost ~]# cd /var
[root@localhost ~]# ls -la
total 72
drwxr-xr-x. 18 root root 4096 Jul 30 22:43 .
drwxr-xr-x. 23 root root 4096 Sep 14 20:42 ..
drwxr-xr-x. 2 root root 4096 May 14 00:12 account
drwxr-xr-x. 11 root root 4096 Jul 31 22:26 cache
drwxr-xr-x. 3 root root 4096 May 18 16:03 db
drwxr-xr-x. 3 root root 4096 May 18 16:03 empty
drwxr-xr-x. 2 root root 4096 May 18 16:03 games
drwxr-xr-x. 1 root root 4096 Jun 2 18:39 gdm
drwxr-xr-x. 36 root root 4096 May 18 16:03 lib
drwxr-xr-x. 2 root root 4096 May 18 16:03 local
lrwxrwxrwx. 1 root root 11 May 14 00:12 lock -> ../run/lock
drwxr-xr-x. 14 root root 4096 Sep 14 20:42 log
lrwxrwxrwx. 1 root root 18 Jul 30 22:43 mail -> spool/mail
drwxr-xr-x. 2 root root 4096 May 18 16:03 nis
drwxr-xr-x. 2 root root 4096 May 18 16:03 opt
drwxr-xr-x. 2 root root 4096 May 18 16:03 preserve
drwxr-xr-x. 2 root root 4096 Jul 1 22:11 report
drwxr-xr-x. 1 root root 6 May 14 00:12 run -> ../run
drwxr-xr-x. 14 root root 4096 May 18 16:03 spool
drwxr-xr-x. 4 root root 4096 Sep 12 23:50 tmp
drwxr-xr-x. 2 root root 4096 May 18 16:03 yp
[root@localhost ~]# yum search wiki
Loaded plugins: langpacks, presto, refresh-packagekit, remove-with-leaves
perfusion-free-updates
perfusion-free-updates/primary_db
perfusion-free-updates/primary_db
updates/metalink
updates
updates/primary_db
```



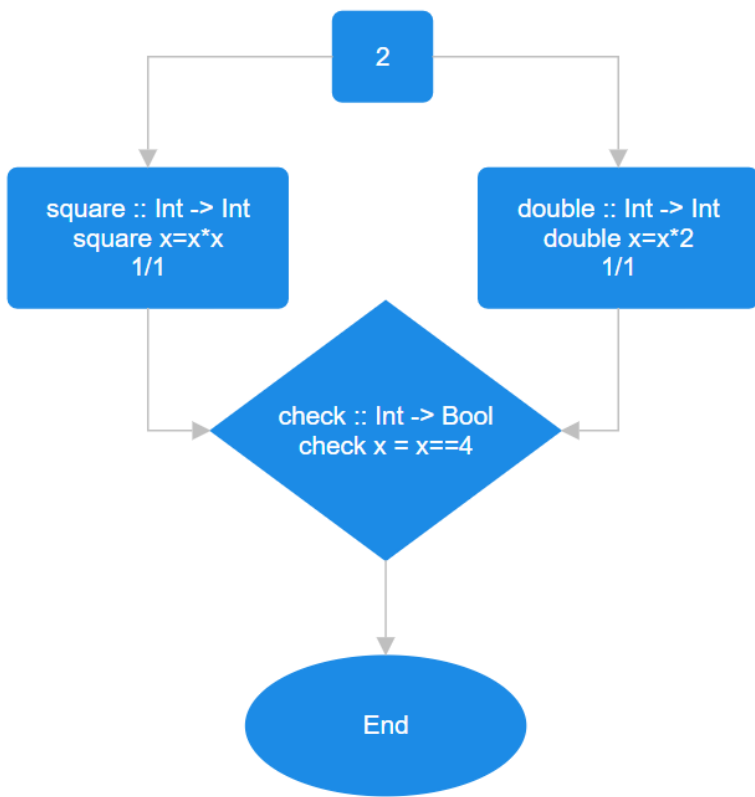
```
File Edit Selection View Go Run Terminal Help extensions - myfirstextension - Visual Studio Code
EXPLORER
  OPEN EDITORS
    ts extensions.ts src 1
    tsconfig.json
  MYFIRSTEXTENSION
    > vscode
    > node_modules
    > src
    > test
  ts extensions 1
  package-lock.json
  package.json
  README.md
  tsconfig.json
  vsc-extension-quickstart.md
OUTLINE
TIMELINE

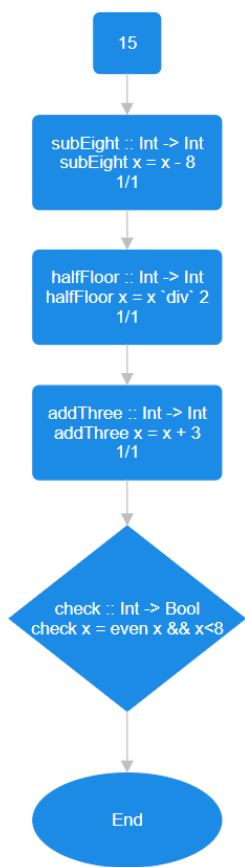
src > ts extensions > activate
1 // The module 'vscode' contains the VS Code extensibility API
2 // Import the module and reference it with the alias vscode in your code below
3 import * as vscode from 'vscode';
4
5 // this method is called when your extension is activated
6 // your extension is activated the very first time the command is executed
7 export function activate(context: vscode.ExtensionContext) {
8
9   // Use the console to output diagnostic information (console.log) and errors (console.error)
10  // This line of code will only be executed once when your extension is activated
11  console.log('Congratulations, your extension "myfirstextension" is now active!');
12  context.subscriptions.push(
13    vscode.commands.registerCommand('myfirstextension.helloWorld', () => {
14      // The 'vscode' environmentVariableCollection
15      // Now p extension
16      // The extensionMode
17      let disp @ extensionPath
18      // I @ extensionUri
19      // D @ globalState
20      vsco @ globalStorageUri
21      // logUri
22      @ secrets
23      context.storageUri
24      @ subscriptions
25
26  // This method is called when your extension is deactivated
27  export function deactivate() {}
28
29  (method) ExtensionContext.asAbsolutePath X
  ePath(relativePath: string): string
  Get the absolute path of a resource contained in the extension.
  Note that an absolute uri can be constructed via uri.joinPath and extensionUri, e.g. vscode.Uri.joinPath(context.extensionUri, relativePath);
  @param relativePath — A relative path to a resource contained in the extension.
  @return — The absolute path of the resource.
```

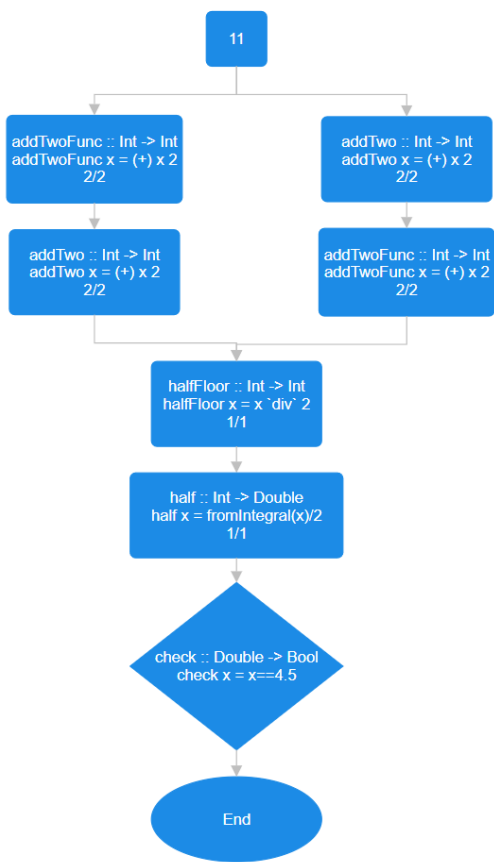


Chapter 1

Haskell basics

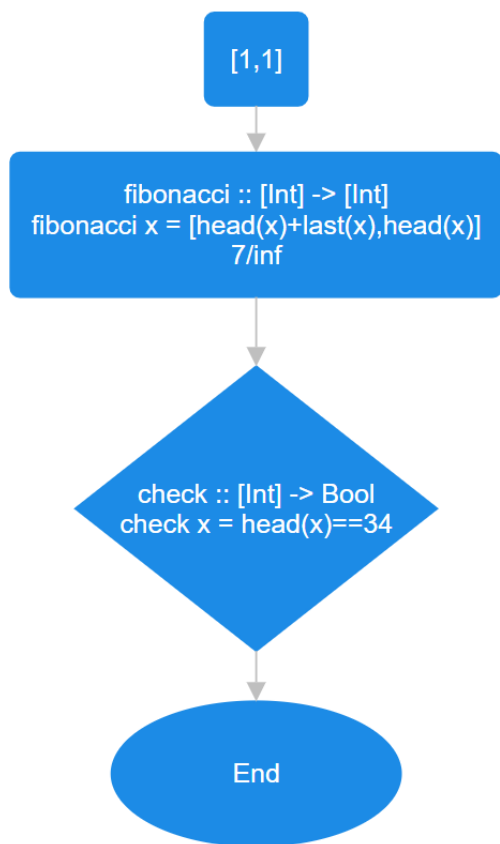


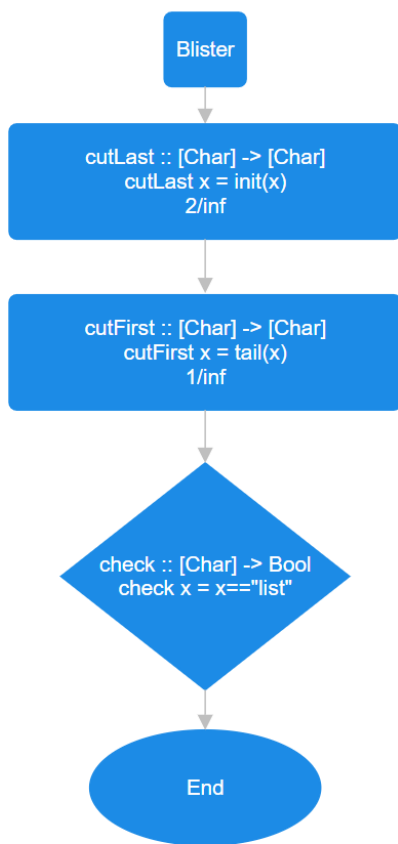


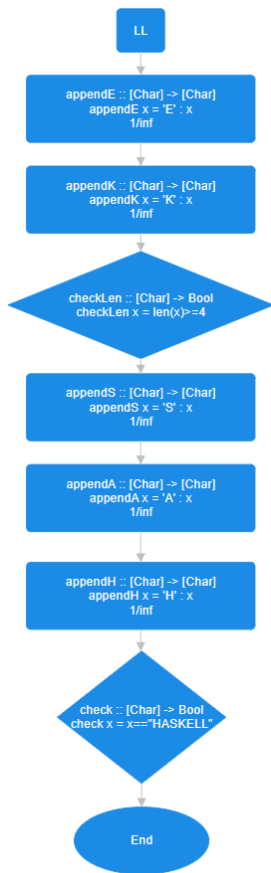


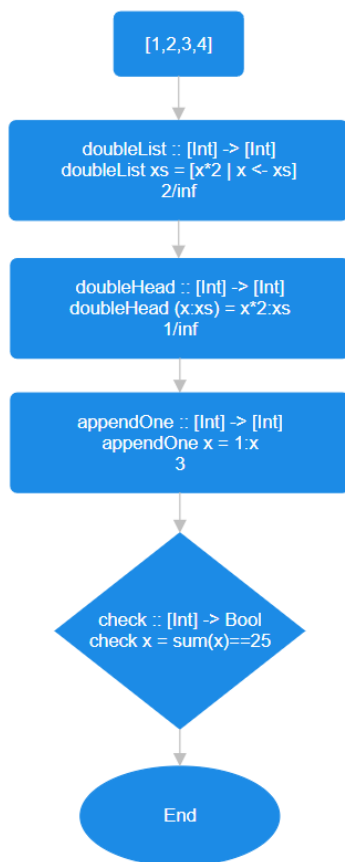
Chapter 2

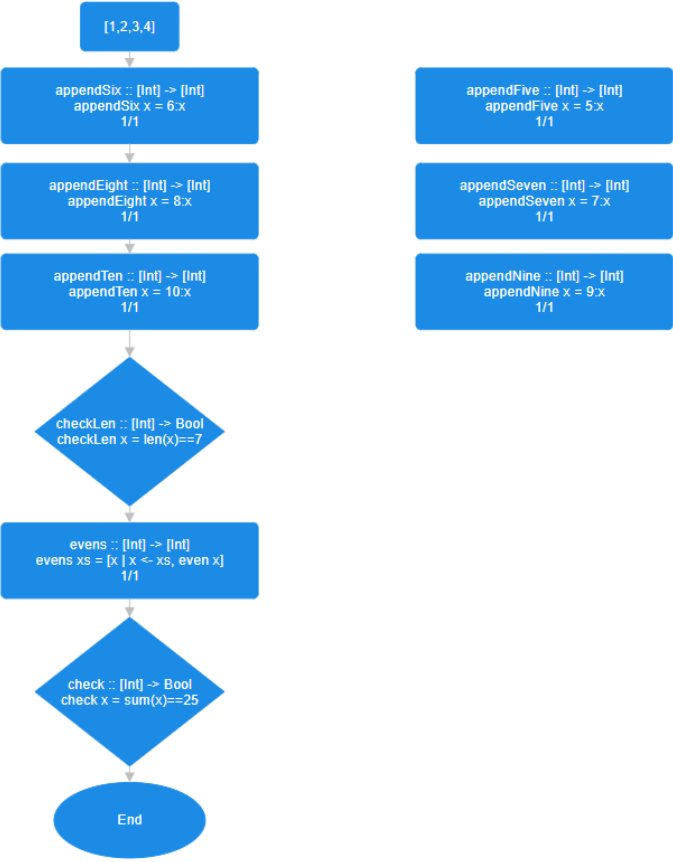
Lists





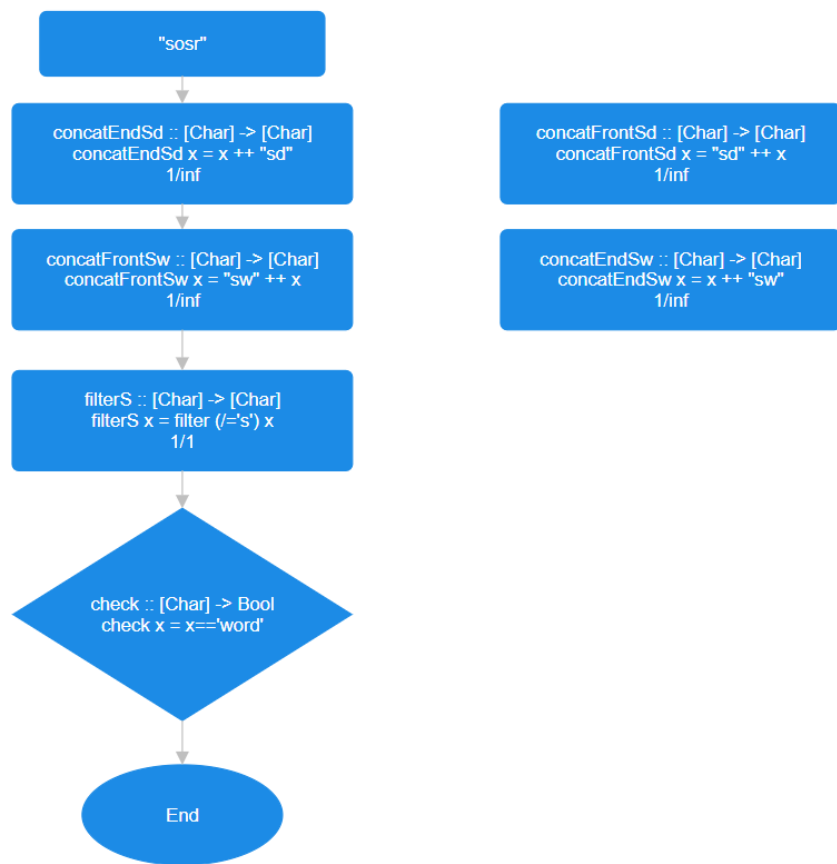


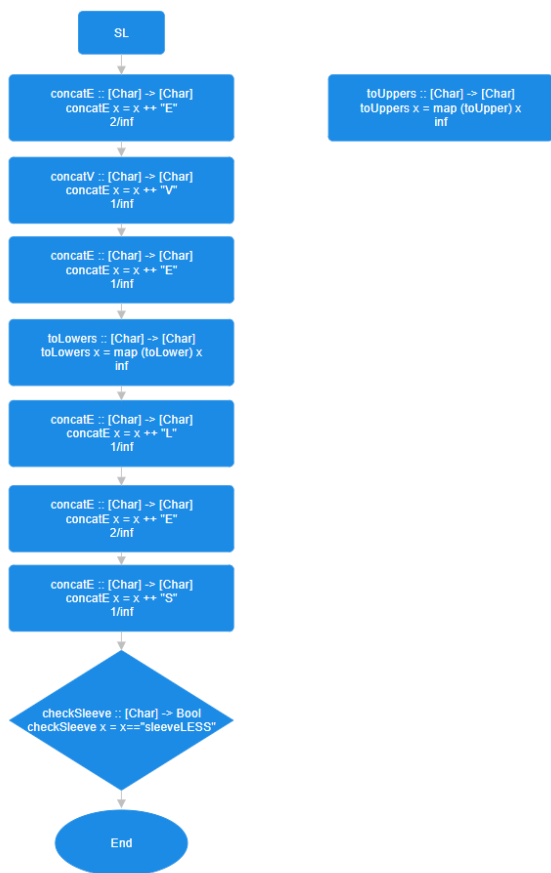


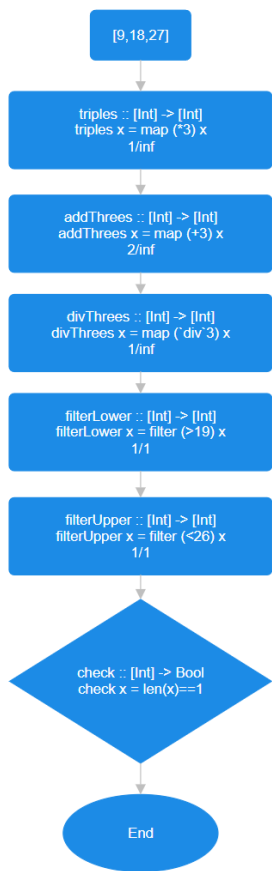


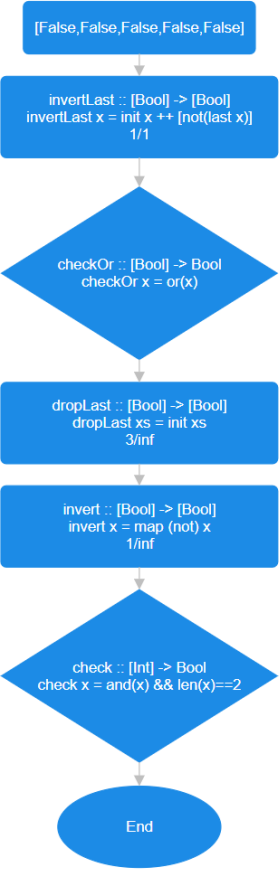
Chapter 3

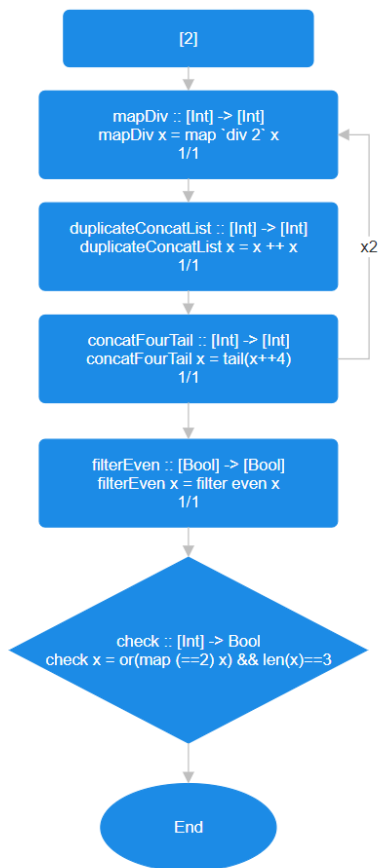
Map/Filter & And/Or











$[2]$
 $[1]$
 $[1,1]$
 $[1,4]$
 $[0,2]$
 $[0,2,0,2]$
 $[2,0,2,4]$
 $[1,0,1,2]$
 $[1,0,1,2,0,1,2]$
 $[0,1,2,1,0,1,2]4$
 $[2,2,4]$

Enemies

Ingame & Menus

Turret

- Static
- Shoot in bursts
- Red glowing weakpoint

Binary Glitch Enemy

- Randomly toggles between 0/1
- Weakpoint when it's ==1
- Explodes on contact
- Constantly travels towards player

Shooty guy

- Head is weakpoint
- Walks towards player, stops when gets line of sight & within some distance
- Shoots individual shots

Appendix D

HaskellQuest Rough Timeline

Can a fun game teach you something?

SEMESTER 1

Week 1 & 2: PLAN

- Plan
- Finalise game idea

Week 3 & 4: RESEARCH

- Workshop : Background writing Mon 2nd Oct 17:10 – 18:00 AT-LT1
- Functional programming in industry
- Functional programming in education
- Serious games
- Games – what makes them fun? Gamification of programming languages, previous work - Duolingo
- Industry standards for game design
- Game design document
- Sprints/Agile/Devops
- Vertical Slice
- Alpha/Beta testing
- Minimum Viable Product
- Nelson Sexton & Vilaskis
- Google form: For those who have studied Haskell (INF1A)
- Did you encounter functional programming before INF1A?
- On a scale of 1 to 10, how difficult do you find Haskell/functional programming?
- On a scale of 1 to 10, how much did Haskell help you understand functional programming in general?
- What is the most difficult part of Haskell?
- What is your favourite way to learn new programming languages?
- Have you ever used Haskell, or any other functional programming language in work/internships/education after INF1A?

Week 5: DESIGN

- UI wireframes – Figma
- Levels
- Functions (note for later: perhaps final boss could be defeated with an error? Function outputting error explodes)
- Art – music/visuals/story(extension)
- Enemies

Week 6 & 7: SPRINT #1 – Vertical Slice

- Background chapter for supervisor feedback Fri 20th Oct
- Movement
- Enemies & combat-Functions & value manipulation

- 1 level OR 1 chapter
- Saving
- Settings
- Menus

Week 8 & 9: SPRINT #2 – Extra features

- Boss
- Timer/score
- Random functions
- Extra chapters

Week 10 & 11 & 12: SPRINT #3 – Wrapping up

- Meet second marker for feedback Mon 27th Nov – Fri 1st Dec
- Finish all levels (if needed)
- Early Q&A (me, friends)
- Wiggle room in case stuff gets delayed

SEMESTER 2

Week 1 & 2:

- Practice project presentations Mon 22nd Jan – Fri 26th Jan
- Alpha testing : bugs
- Google form: bugs (mandatory)
- Where did you encounter the issue? (menus, ingame, installation)
- What are the steps to replicate the bug?
- Hardware/OS/etc?
- Attach a screenshot if possible.
- Bug fixing

Week 3 & 4:

- Workshop : Dissertation Writing Fri 9th Feb 13:10 – 14:00
- Beta testing : gameplay feedback
- Google form: Bugs (as above) (optional)
- Google form: Previous Haskell knowledge (mandatory)
- Simple -> difficult Haskell functions, ask testers what the output value is
- Google form: Feedback (mandatory)
- Same function question as “Previous Haskell knowledge”
- Background questions (Haskell/coding experience, occupation, age, etc)
- Was the game fun? 0-10
- How difficult was the game? 0-10
- Do you think this would be a useful tool to improve people’s Haskell intuition? 0-10
- How would you improve the game?
- What do you like most about the game?

- What do you dislike most about the game?
- + any specific questions that I might come up with later

Week 5 – 7:

- Write dissertation
- Improve the game based on feedback (if time allows)

Week 8 & 9:

- Dissertation for supervisor feedback Mon 11th Mar – Fri 15th Mar
- Project Day Wed 13th Mar 17:00 – 19:00

Week 10 & 11:

- Dissertation submission deadline Thu 4th Apr 12:00
- Project viva with supervisor and second marker Mon 22nd Apr – Fri 26th Apr