Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

James Martin

28th April 2024

Improving learning to program for non-computer science undergraduates by creating a gamified application

Project supervisor: Dr Jian Shi
Second examiner: Dr Sasan Mahmoodi

A project report submitted for the award of
BSc Computer Science

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.

- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.

- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

- I have acknowledged all sources, and identified any content taken from elsewhere.

- I have not used any resources produced by anyone else.

- I did all the work myself, or with my allocated group, and have not helped anyone else.

- The material in the report is genuine, and I have included all my data/code/designs.

- I have not submitted any part of this work for another assessment.

- My work did not involve human participants, their cells or data, or animals.

Acknowledgements

I would like to thank my supervisor Dr Jian Shi for his extensive guidance throughout the duration of the project and for giving up his time to assist me.

Abstract

Many undergraduate students not registered on a programming module are choosing to learn programming to increase their skillset and are using self-learning tools to do so. For these students, the effectiveness of self-learning platforms/applications available varies immensely. Struggles faced by learners include a lack of both motivation and support which can result in ineffectual learning or the learner giving up. Aiming to address these issues, a literature review was performed, a prototype application was developed that utilises gamification, feedback, and abstraction, and a study was performed to evaluate its efficacy. This paper found the developed application to be successful in improving learning and found gamification and feedback to be effective in improving motivation and support.

Contents

1 Introduction

The number of students learning to program at universities in the UK has only been increasing in the past few years; In just a 3-year period from 2019 to 2022, UCAS applications to take computing courses at university increased by over 20% (Figure 1).[1]

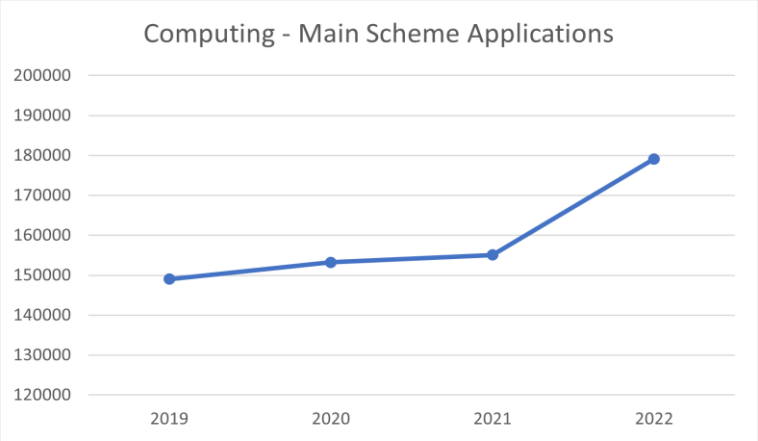


Figure 1: Number of UCAS main scheme applications for computing by year

However, it is not only computing students that learn to program at university – many non-cs courses include AI modules or involve subject-specific languages such as Mathematica [1]. Furthermore, an additional group exists; students who take no programming modules but want to learn to program through self-study. Motivations for this self-study include interest in computing, using programming in a project/dissertation [2], or improved job prospects [3] – a motivation for which the rising use of new computing technologies in nearly every field is responsible [4]. This group is diverse and comes from different educational backgrounds; some will have taken computer science at A level/GCSE while others will not and may lack the foundation of computational thinking ability [5].

Students in this group have the option to do a MOOC (Massive Online Open Course), of which many teach programming, however this cohort still faces many challenges, namely the lack of support as the students don't have the same access to lecturers or experts to ask questions and get feedback that programming students get [6]. Additionally, the lack of examination or deadlines means they rely on their own motivation to continue learning; this is especially a problem when students already face a large workload on their course of study, and it has been shown that worse learning outcomes are achieved when students are not motivated by fixed deadlines [7]. Moreover, Maslow's theory of motivation states that "stimulating a sense of purpose in students is one of the most powerful ways for enhancing learning and skill retention" [8]. Gamification, a term coined in late 2004 by Nick Pelling [9], is the act of applying game related concepts to a system or application to enhance motivation and engagement – hence its widespread use in teaching [10] and self-learning applications [11]. Implementation of gamification when learning to code has been shown to make students engage more and increase student performance, [11] so it is a subject of study in this paper.

[1] UCAS (2023) UCAS Undergraduate Sector-Level End of Cycle Data Resources 2022. Available at: https://www.ucas.com/data-and-analysis/undergraduate-statistics-and-reports/ucas-undergraduate-end-cycle-data-resources-2022 (Accessed 19 November 2023)

1.1 Aim

This paper aims to improve the experience of non-cs undergraduates who are not taking computing modules as they learn to program by envisioning a prototype gamified application that addresses problems that they currently face. This paper does not consider the experience of other students who take computing modules as they have far greater access to their university's relevant resources and lecturers. To this end, this paper has three research objectives:

1) To research and find factors that hinder self-learning when starting to learn to code.
2) To develop a prototype application which could address these factors and motivate student's learning.
3) To evaluate how effective the application is at motivating learning and teaching programming to beginners.

1.2 Structure

Excluding the introduction, this paper consists of four chapters:

1) A literature review where appropriate and recent literature is discussed to identify factors hindering programming self-learning, and their solutions.
2) An application chapter consisting of design documentation, implementation details, and testing of the prototype application.
3) An evaluation of the effectiveness of the application, using a study involving a questionnaire completed by participants.
4) A conclusion that discusses the outcome of the project and benefits found.

1.3 Time Management

To ensure the project is completed in time, it is necessary that a time management plan is created and followed. This involves estimation of how long each section will take and creation of a visual plan of the blocking – a Gantt chart is chosen to represent this information (**Table 1**). A risk assessment is also beneficial as it can ensure each possibility is considered and appropriate mitigations are put in place. A risk assessment was created considering each situation that could lead to failure to complete the project (**Appendix C**).

	OCT	NOV	DEC	JAN	FEB	MAR	APR	MAY
Literature Review								
Design								
Requirements								
Design Artifacts								
Implementation								
Testing								
Report Writing								
Evaluation								

Table 1: Gantt chart showing the project's time management plan

2 Literature Review

To improve learning, first the challenges faced by the learners must be established, then it is important to understand how students learn, such that the subsequently found solutions can be mapped to steps in the learning cycle. This chapter starts by exploring problems that hinder self-learning when starting to learn to code, as per the first report aim. It then explores Kolb's learning cycle to understand how students learn, finally it explores the concepts of gamification and feedback with case studies to find solutions to the listed problems, for implementation in the prototype application.

2.1 Challenges

Learning how to program can be a daunting undertaking for undergraduates not taking computing modules, despite the large number of existing courses and resources available; Not only are there over 270 different programming languages to choose from,[2] but also many courses that that introduce them are paid, meaning learners are discouraged from the outset. Not only is selecting a programming language to learn daunting but selecting an inappropriate one can be detrimental to students' understanding of programming [12]. Due to this many students take the option of following courses that are language-agnostic, however these are often too abstract to be useful for continued learning, as students face a high learning curve as they eventually must make the jump to conventional programming languages [13].

There are many challenges identified by researchers, but the most frequently raised are:

2.1.1 Motivation

One challenge of students undertaking self-learning is that of motivation, a lack of which can be a detriment to learning. For example, research has shown that the absence of "a sense of purpose" when learning can directly cause negativity, boredom, and possibly anxiety [8]. Motivation can be impacted by negative perception towards learning to program; Studies show that preconceived notions of programming such as it requiring an exceptional amount of effort or time to learn, can cause students to fear starting the learning process [15, 11].

2.1.2 Subject Difficulty

Additionally, programming is an inherently difficult skill to learn; it is said that learning to program is a challenging skill to learn for students on computer science courses [11] so for non-cs self-learners this challenge is even greater. A possible cause is that the concepts that students will learn such as variables and data types cannot easily be compared to concepts in real life [16], making them difficult for new learners to grasp. Given the difficulty, it is not uncommon for learners partaking in popular MOOCs or using popular platforms to resort to cheating; if the language taught by the course is widely used then during code writing activities unmotivated learners can easily be tempted to cheat and find answers online, as it is easier and gives the same result [17].

2.1.3 Support

Another significant challenge is that of a lack of support – for students partaking in MOOCs, a large contributor to the dropout rate is the absence of interaction, which leads to a sense of isolation [14]. This means students rely on themselves to stay motivated – a problem also exacerbated by the lack of schedule, as demonstrated by a study that found

[2] Tiobe (2023) Tiobe Programming Community Index. Available at: https://www.tiobe.com/tiobe-index/ (Accessed 22 October 2023)

that the dropout rate of self-paced MOOCs was 3.5 times higher than fixed-schedule MOOCs [14].

In summary, the main factors that hinder self-learning when starting to learn to code are a lack of motivation, a lack of academic support, and the inherent difficulty of learning the subject.

2.2 Learning Cycle

To solve these challenges, it is important to understand how students learn effectively - many educators employ learning cycles to improve learning, one of the most famous

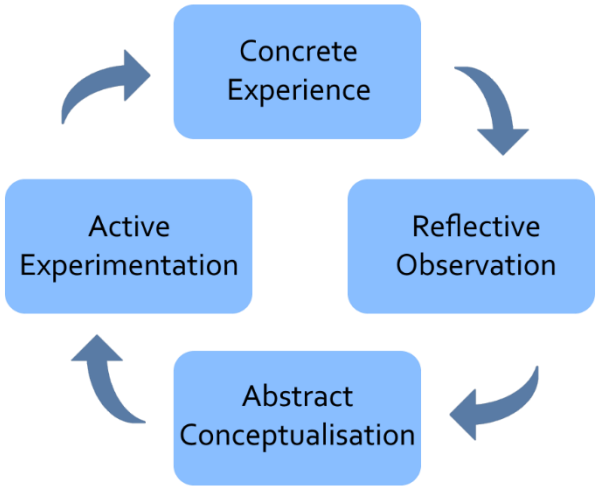


Figure 2: Kolb's Learning Cycle

being Kolb's learning cycle (Figure 2).

The cycle is based on "perception and processing" [18] and consists of the student having an experience, reflecting on the experience, extending the idea through hypothesis, and finally experimenting based on the hypothesis. Being a cycle, this process repeats, and while concrete experience is usually the first step, some researchers state that there is no beginning stage, any stage can be the start point [18]. Applied to programming, the cycle would have students writing and testing code as part of the learning process; this aligns with what researchers suggest, that it is important for students to both learn and practice, especially for unfamiliar skills such as debugging/correcting errors and analysing problems [16]. This is not what always happens in practice however, as students often spend too much time reading theory and not practicing writing actual code [16], therefore it is important that a programming self-learning tool allows the learner to practice and have concrete experiences such that the cycle can continue with reflection. Other cycles exist such as the 5E model, however that is more suited to classroom or instructor-facilitated learning, so a self-learning tool would be more effective with Kolb's cycle as a focus.

2.3 Existing Pedagogies

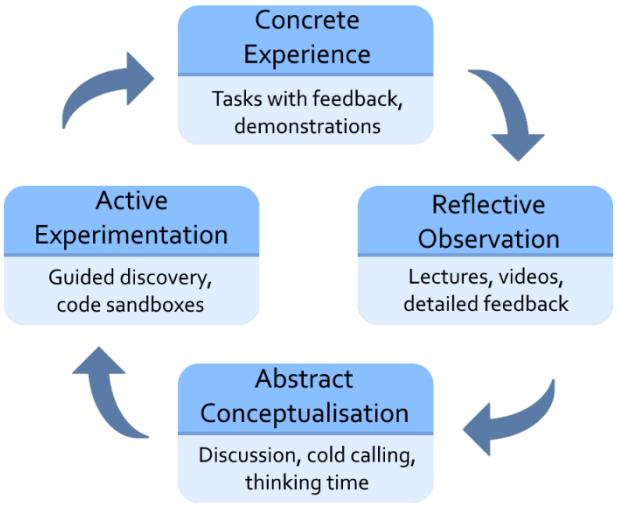


Figure 3: Pedagogies used to further each cycle step

The Reflective Observation part of the cycle is less important for this system as free online lecture notes, explanations, and MOOCs are available to any student [7] whether on a programming course or not. For this reason, it is not necessary for a programming self-learning tool to include such materials but may be helpful to link to them. What many existing tools lack is the environment to try out the concepts being taught and meaningful feedback [7], so Concrete Experience and Active Experimentation should be the focus of such tool. Abstract Conceptualisation is facilitated more by the student than the tool, so in this case the availability of both information and a practical learning environment should allow students to extend their thinking and create then act on hypotheses of their own.

2.4 Gamification and Self-Determination Theory

Aiming to improve the steps of the learning cycle through increasing learner motivation, gamification is a frequently used pedagogy in self-learning applications and involves using game design elements to enhance non-game applications [19]. It can be used to increase engagement and motivation, especially in learning applications [20]. It is used in self-learning platforms to make the process of learning more fun and engaging, which is especially important compared to traditional tutor-lead learning due to the lack of in-person interaction and support - In these situations, motivation is key to keep learners engaged and prevent them from giving up. Research has proven gamification's link to motivation by showing how it provides satisfaction to the learners [20]. Additionally, studies have shown that gamified learning can increase test results in traditional learning situations [10], so learning quality in self-learning situations is likely increased too.

Self-Determination Theory (SDT) is a motivation theory often applied to implement gamification in a learning system. It presents three "psychological needs" to be requirements in achieving both intrinsic (internal) and extrinsic (external) motivation [19], these are: 1) Autonomy, where the learner should feel that they are learning due to their own motivation 2) Competence, where a learner is aware of the progression of ability 3) Relatedness, where the learner experiences belonging and collaboration [19].

SDT is effective as it doesn't only focus on extrinsic motivation but can build students' intrinsic motivation to improve engagement [20]. Some game elements that implement SDT are avatars, application customisability, levels, positive feedback, and chatrooms – most can be applied to a self-learning programming tool to increase motivation, however relatedness elements like chatrooms would be less effective due to the solo and asynchronous nature of self-learning.

2.4.1 Scratch

Scratch is a high-level graphical programming language and gamified block-based coding environment with a focus on introducing young people to programming. Studies have found that Scratch offers many benefits to learners including increased self-efficacy and greater internal motivation [27]. Moreover, a separate study found that students who used Scratch over text-based coding gained a greater understanding of programming and a better perception of the subject [28].

Scratch has many features that ensure it achieves its goal of fostering computational thinking and increasing programming accessibility including:

- Many elements of gamification that enhance motivation, increase interest in learning to program, and keep students learning [28].
- A low barrier to entry due to its intuitive block-based system [29] and the fact that syntax errors are not possible due to the blocks' connection types [30].
- Many computational-thinking concepts supported by the language such as parallelism and synchronisation [30].

Scratch also has limitations however, such as:

- A lack of progression/meta-cognition due to the lack of a structured learning plan and educational activities [30].
- A lack of feedback and explanation of complex program concepts as evidenced by students struggling with complicated loops in studies, a concept Scratch provides no tutorials for [31].
- Difficulty in later switching to traditional languages that use text [32]. Additionally, students often only use the simple features instead of properly engaging with the more complex constructs available [32], demonstrating a lack of extrinsic motivation to do so.

2.5 Feedback

To facilitate the Concrete Experience and Reflective Observation stages, and to aid in the learner support that is often lacked in self-learning applications, feedback is critical for the student to learn from their mistakes and make progress without losing motivation. Feedback can be described as information given to a student based on their performance to aid their understanding [21], or specifically in the context of learning a programming language, feedback can be in the form of instructors giving personal code criticism, compiler comments, or data-driven hint generation [22]. It has been observed that students who receive feedback during the learning process achieve greater results than those who don't, meaning feedback can have a direct impact on learning quality [6]. Furthermore, students who receive expert support go on to achieve more with self-

directed learning [6]. Due to the lack of instructor support in self-learning, feedback is key to prevent learners getting stuck and giving up and can be an effective way to increase motivation [23]. Researchers have noted the importance of the students' understanding of their own progress and process as part of feedback and have stated that, by observing their own personal process, students can "internalize metacognitive observations" [24].

The transactional model of stress and coping describes two cognitive appraisal styles of threat and challenge, which researchers have found to dictate students' responses to different types of feedback [25], meaning it is important to consider both when providing feedback. More specifically, it is the suitability between the feedback valence and the students' appraisal style that determines the response [25], with feedback valence referring to positive and negative feedback; this means that using both feedback types together can ensure that learners of either cognitive style can benefit. Researchers have also identified detailed feedback to be more beneficial to students than correctness feedback [24].

Researchers identified four major types of programming exercise feedback [26]:

- Task Constraints: information on what the task involves, or what the solution is missing.
- Concepts: information on the concept/construct being introduced/used.
- Mistakes: information on solution or style errors the student has made.
- How to Proceed: hints on what to do next, or how to improve.

Specific examples of feedback systems in programming e-learning systems include automatic grading, program flow visualisation, and hints to assist students in completing their task [26].

2.5.1 Codecademy

Codecademy is a self-learning platform that teaches the major programming languages through step-by-step tutorials, many of which are free. The tutorials contain both explanatory text and interactive exercises that get the learner to write actual code to continue [33]. Codecademy uses feedback for user support, and gamification to increase user motivation, involving points and badges that students can earn through passing levels [33].

Beneficial features of Codecademy have been found to include:

- Fast and helpful feedback that guides students in correcting errors [34].
- A highly interactive learning environment since most lesson steps require the learner to write actual code [35] - evidenced by a study which found that the interactivity increased engagement and reduced boredom for the students who tried it [35].
- A guaranteed progression through structured educational content that provides convenience and means beneficial learning outcomes are less reliant on the learner's research ability [35].

Conversely, researchers have also found limitations of Codecademy including:

- Limited freedom/creativity due to answer solution structure being too rigid and strict, resulting in frustration or confusion in learners [34, 35, 36].

- Lack of focus on computational thinking, as language syntax is the focus in tutorials, rather than developing students' ability to work through [35].

2.6 Summary

Based on the analysis above, for a programming self-learning platform to be effective it should address the three key factors: lack of motivation, lack of support, and inherent difficulty of learning the subject. Such a platform should utilise gamification to increase motivation of the learner, provide effective feedback to allow the learning cycle to progress, and employ abstraction to better ease of understanding and develop computational thinking. Developing this platform forms the second research objective of this project.

3 Application

The literature reviewed in the previous chapter and the issues identified demonstrates that a prototype gamified learning tool is needed to improve the experience of non-computer science students learning to program; this is identified as the second research objective of this research. This chapter outlines the prototype application, aiming to addresses the main factors that hinder self-learning when starting to learn to code: a lack of motivation, a lack of academic support, and the inherent difficulty of learning the subject. This application focuses on gamification, feedback, and abstraction to achieve this. First a list of requirements is defined based off the literature review, then design artifacts are established that plan development of this prototype for evaluation. Finally details of how the application is tested are given. The agile methodology is preferred for this project to ensure a working prototype is produced at the end; agile results in multiple prototypes leading up to a final system with all planned features developed.

3.1 Requirements

To apply gamification the application consists of small challenges acting as levels, which the learner will need to progress through to fully unlock, and like a game, a victory sound plays on completing levels. Additionally, there is a customisable profile icon and name to increase learner attachment to the tool. To apply abstraction, the levels are split into two, with the first set using a highly abstracted train track language to introduce programming, and the second set using a less abstracted pseudocode-like language to ease the transition into traditional languages. Finally, to apply feedback the application provides the four major types identified in the literature review: constraints, constructs, errors, and hints.

The availability of educational information online is not one of the issues raised in the previous chapter, so the application focuses on practical learning and practice, while providing links to appropriate online learning resources to be used alongside it.

No.	Requirement	Reference
1	The system can provide links to external free learning materials	See section 2.3
2	The system can use abstract puzzles to teach computational thinking	See section 2.1.2
3	The system can use a pseudocode-like language to teach program concepts/structure	See section 2.1
4	A built-in interpreter can evaluate user's programs/solutions to determine correctness	See section 2.2, 2.5
5	The system can provide feedback including task constraints, concepts, errors, and hints	See section 2.5
6	The system can track and display user progress, and save/load this data	See section 2.4
7	The system can allow user personalisation including name and avatars	See section 2.4
8	The system can allow the user to import and play custom levels	See section 2.5.1
9	A sandbox mode can allow users to experiment without goals	See section 2.2

Agile development consists of sprints; cyclic periods of development after which prototypes of increasing feature count are produced. Sprints are laid out before development in sprint plans, where requirements and subtasks are split into increments. Stakeholders are also defined, and development is catered to their requirements.

3.3 UML Design Artifacts

To plan the structure of the system, UML diagrams are chosen for their simplicity. A class diagram defines the structure of the codebase using connected classes and their methods, and state diagrams set out the two level types' behaviours and states. Additionally, a storyboard is used to plan the UI (**Appendix A**).

3.3.1 Class Diagram

The class diagram shows the rough framework of the application, with three main controller classes that control the main menu, train levels, and pseudocode levels. The prototype employs the MVC design pattern to keep the code compartmentalised and easy to maintain, especially since the application involves many separate scripts. Only the major methods and attributes are shown.

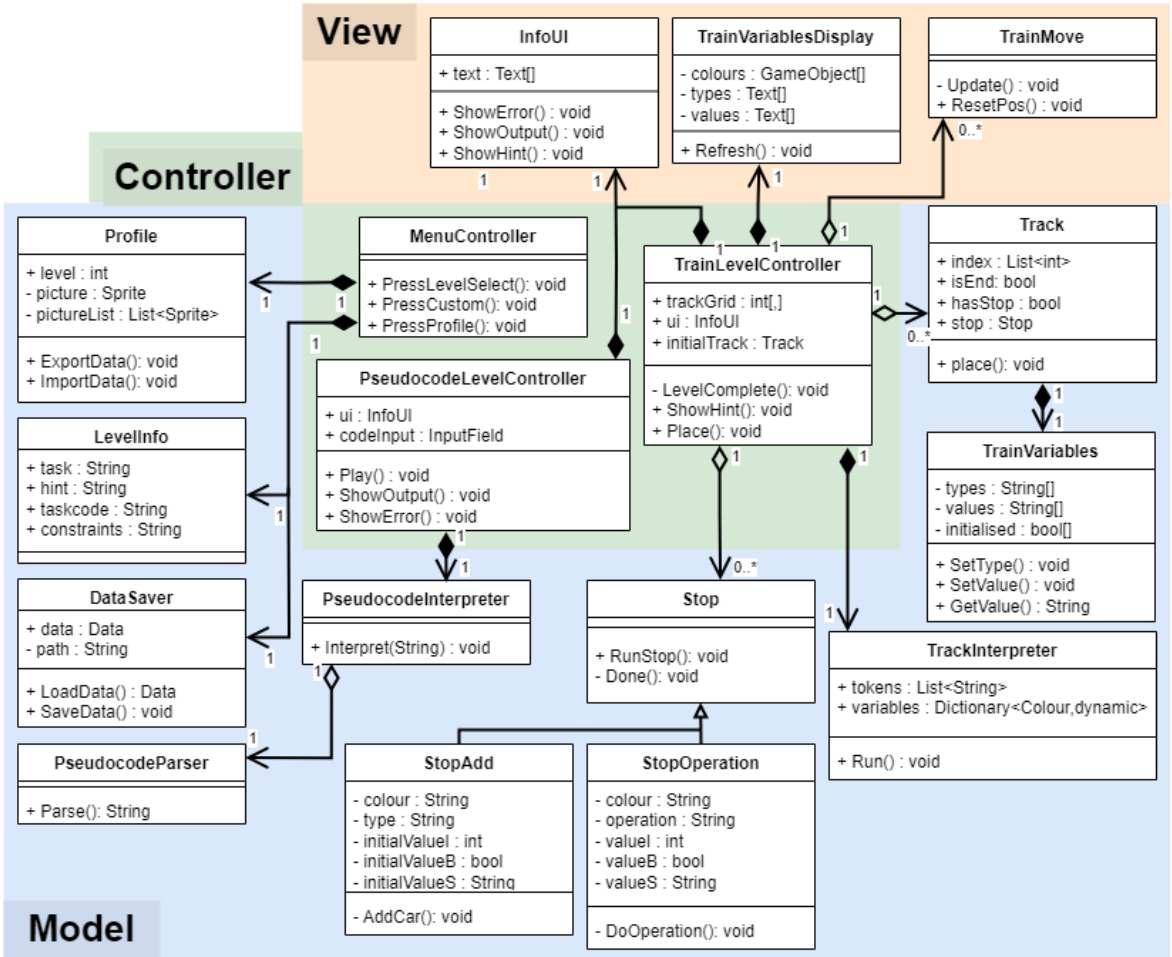


Figure 4: Class diagram of major classes/methods/attributes

3.3.2 State Diagram (Train Levels)

This state diagram shows the main behaviours of the systems that implement the train levels, and the result of the user's possible actions. This includes the track placement, program interpreter, and feedback systems.

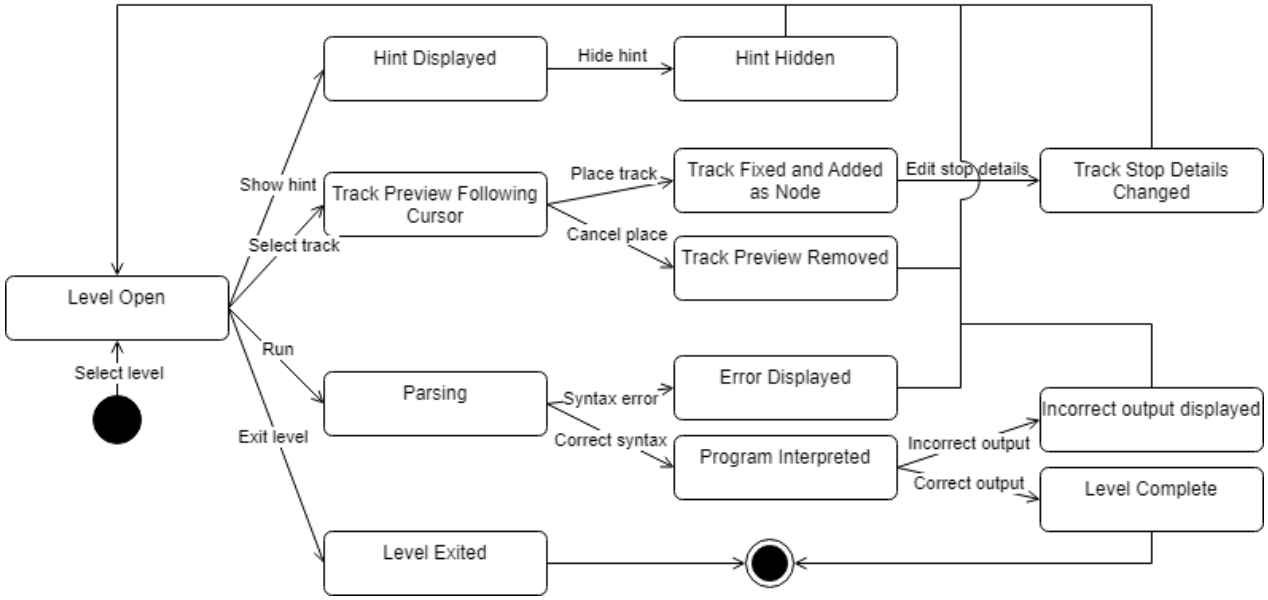


Figure 5: State diagram of train levels

3.3.3 State Diagram (Pseudocode Levels)

This state diagram shows the main behaviours of the pseudocode level systems. The main difference to the train level system is the way code is written, here using plain text with syntax highlighting, as opposed to train tracks of different types.

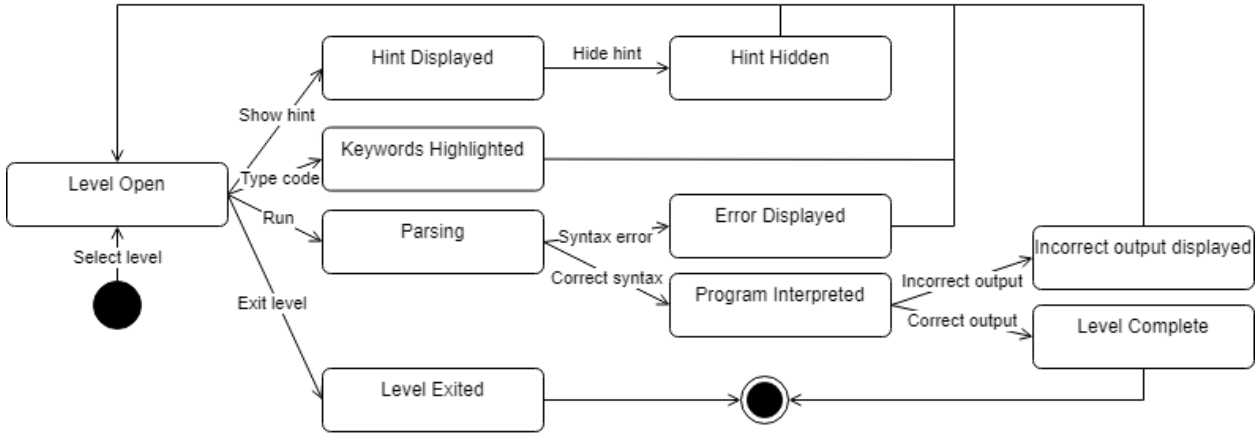


Figure 6: State diagram of pseudocode levels

3.4 Stakeholder and Personas

The application has a sole primary stakeholder: **undergraduate students at the University of Southampton not taking programming modules but who wish to learn to program**. While others are free to use it, the application is designed for those without access to sufficiently motivating, basic learning tools. From the stakeholder definition, two key personas can be defined:

Max, 19
An English Part 1 undergraduate at UoS with no programming experience. They want to use a program to learn the basics of programming to improve their job prospects and increase their skills, but their course contains no programming modules, and they don't want to commit too much time to the MOOC offered by their university. They have tried a few online tools that teach Python but gave up soon after starting due to the increasing complexity causing them to lose their motivation.

Ellie, 21
A Chemistry Part 3 undergraduate at UoS with 2 years programming experience having done computer science at A-Level. They have a solid understanding of the basics but wish to refresh their memory by using a learning tool. They want the tool to be fun to use and motivating so that they can use it alongside their other studies.

3.5 Scenarios

Scenarios are defined based on the personas, describing how they might use the system. They are used during development for scenario testing, to ensure actual users' needs are met.

Scenario 1 – Max

- They launch the application and reach the main menu.
- They progress to the level select page.
- They select a train level they have not yet completed.
- The level screen includes a description of the level requirements.
- They add tracks and stops to create a program.
- They run the program and the train moves along the track, visually updating the variables as it progresses.
- The program completes and results in the level requirements being met, and a victory sound is played.

Scenario 2 – Ellie

- They launch the application and reach the main menu.
- They progress to the level select page.
- No levels are locked, they are all selectable.
- They skip the train levels as they are too basic and select a pseudocode level.

Scenario 3 – Ellie

- They open the folder the application is located in and find an example .lvl custom level file.
- They copy and modify the file to create their own custom level.
- They share the level on the internet for others to download and try.

3.6 Increment and Sprint Plan

Following the agile methodology, the prototype is built in three major increments, with each increment adding new features to the application. The colour indicates task effort, with red being most and green being least.

Increment 1	Increment 2	Increment 3
Train level interpreter	Other feedback	Pseudocode interpreter
Error feedback	Profile customisation	Custom levels
Train levels	User progress and saving	Sandbox mode
	Links to external materials	Pseudocode levels

Three sprint plans can be devised for each increment, with predicted difficulty, effort, and time stated for each subtask to ensure the workload is balanced over development for proper time management.

Increment 1:

Requirement	Sub Tasks	Difficulty	Effort	Time
Train level interpreter	Define supported constructs and data types	S	S	S
	Develop node interpreter	L	L	L
	Display variables and program flow (train)	M	L	L
Error feedback	Define errors and create display	S	S	M
	Develop type checker	M	M	M
Train levels	Define level tasks and constraints	S	S	S
	Develop track placer and snapping	M	M	M
	Prevent control flow jumps	M	S	M

Increment 2:

Requirement	Sub Tasks	Difficulty	Effort	Time
Other feedback	Define hints and create UI	S	S	M
	Display task concepts and constraints	S	M	M
Profile customisation	Create icon selector and icons	S	M	M
	Add name and other personalisation	S	M	S
User progress and saving	Track progress across levels	S	S	S
	Display progress bar	S	S	S
	Implement saving/loading files	S	M	S
Links to external materials	Add links in menu	S	S	S

Increment 3:

Requirement	Sub Tasks	Difficulty	Effort	Time
Pseudocode interpreter	Define supported constructs and data types	S	S	S
	Develop parser and lexer	L	L	L
	Develop interpreter	L	L	L
	Display variables	S	S	M
Custom levels	Create custom file parser	M	M	S
	Add file import UI	S	S	M
Sandbox mode	Add sandbox option without constraints	S	S	S
Pseudocode levels	Define level tasks and constraints	S	S	M

3.7 Implementation and Testing

3.7.1 Tools

For speedy development **Unity** is chosen as the development platform for this prototype, as it is a free and relatively easy to use game engine that has masses of online documentation and is well supported. Unity's UI system is very suitable for this kind of application. **Visual studio** is chosen as the IDE due to its deep integration with Unity and its ease of use with Unity's language C#. The **Unity Test Framework** – a Unity integration of NUnit - is used for unit tests throughout development as it is built into Unity and simple. Finally, the **CSLY** library is selected to generate a lexer for the pseudocode levels; it is a C# lexer/parser generator based on PLY, a python implementation of Lex/Yacc.

3.7.2 Implementation

The most time-consuming parts of the development are the interpreters; one is required for each set of levels to evaluate program outcomes and return errors to the user. While the interpreter for the train track levels must be fully custom, its design can be simple as the tracks are made to only connect and be configurated to form programs without syntax errors – this is inspired by Scratch's block-connection design (**section 2.4.1**). When tracks connect, they update next and for pointers effectively forming a tree-like linked list with the initial track as the root. Therefore, a lexer and parser are unnecessary, as only an evaluator is needed to traverse the nodes and perform type checking and evaluate the result.

The implementation for the pseudocode level interpreter is more involved since raw text must be parsed. This means a lexer is needed to convert the text into a list of tokens, based on predefined lexemes, and a parser is needed to turn the list into an abstract syntax tree (AST). Before either can be developed however, the language grammar must be defined. Much of the syntax is inspired by the OCR pseudocode guide, including the use of ENDIF/ENDFOR/ENDWHILE to simplify parsing and solve the dangling else problem. Creation of the lexer is achieved using CSLY, where only an Enum declaration of a list of the possible lexemes is required as the definition. Creation of the parser however is chosen to be custom so that the error descriptions can be customised to be simple and readable for beginner programmers. development first involves designing what types the AST will consist of and how it will be structured, then the actual parser can be created, using recursion and pattern matching to ensure syntax correctness and build up the AST using classes to represent individual constructs/operations/values. Below is a representative example of an AST generated by this parser (Figure 7).

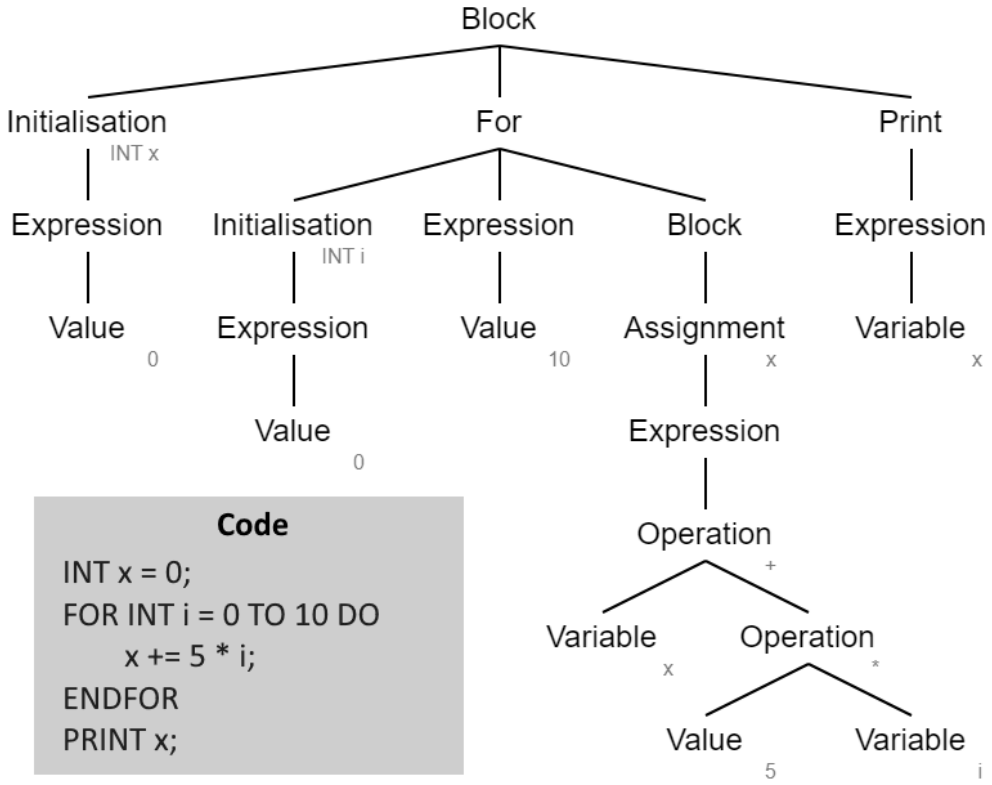


Figure 7: Visualisation of the AST the pseudocode level parser would generate from an example code snippet

3.7.3 Testing

To ensure the application is bug free and works as expected, extensive testing takes place throughout development. This involves three major types of tests: unit tests, integration tests, and acceptance tests.

Unit Testing

This is where smaller parts of the codebase are automatically tested in isolation, across the whole of development. Automated NUnit tests were written after a class is created, then ran multiple times after to ensure no bugs appeared as other classes interacted with it. The pseudocode level interpreter class was the most appropriate for automatic unit tests as the main interpret method can be called in isolation due to a code string being the only parameter required. In each test an assertion was made that an expected output matches the actual output – the test code was passed into the interpret method and the returned result was checked against the expected output, to result in a pass or fail for that individual test.

As part of unit testing, two techniques were used to ensure evaluation correctness:

Boundary Testing

This is where values below, on, and above a boundary are used in tests, because many bugs can occur around boundaries due to off-by-one errors. In the pseudocode level interpreter unit tests, boundary testing was used to ensure that integer calculations were rounding correctly – values in divisions are tested that should round down, result in a whole number, and round up, to cover every case.

Partition Testing

This is where values to be tested are split into groups, and at least one random value from each group is used in a unit test to verify the whole partition. This differs from boundary testing as random test values are used rather than set values. In testing the pseudocode level interpreter, partition testing was used to ensure all cases of conditionals worked as expected.

Unit tests were also used to check specific constructs act as expected (such as loops, nested constructs, negation, operations, comments, and brackets), and test code with syntax and type errors were used to ensure evaluation failed as expected. Below is the outcome of running the unit tests successfully in the Unity Test Framework UI (Figure 8).

Figure 8: Unit test results

Integration Testing

When multiple large parts of an application are completed separately and then combined to form a system, errors can often arise where there were none individually. Integration testing is where large systems are checked for errors, manually or automatically, after each individual sub-part has been unit tested. Manual integration tests for all systems (such as UI and data saving/loading) were performed throughout development on completion of all sub-classes. There are many classes in this application that depend on other classes or require human interaction to run, for which manual integration testing is crucial to ensure validity.

As part of integration testing, two techniques are used:

Black Box Testing

This is where code is tested without knowledge of the implementation, either by using the feature as a user would, or simply checking that provided input gives the correct output. The feature that underwent the most black box testing is track

placement due to the complexity of interactions between tracks. This involved attempts to place tracks and stops in valid and invalid spaces, checking connections between tracks are occurring properly, and that all tracks form a valid linked list. Its important that every track and stop type were checked in case of a bug only being present in one.

White Box Testing

This is where knowledge of the code and implementation of a feature is needed to test it. Inputs and test cases are chosen to test all internal cases and pathways in the code, such as all conditional branches or all methods in a class. An example of this was testing for the train track evaluator, to ensure that all possible error types were caught and displayed to the user, and all evaluation methods were tested – this therefore needed knowledge of the methods inside the evaluator class.

Acceptance Testing

This is the final stage of testing an application, where the system is used in a way to mimic a real user, to ensure at a high level that all features of the application are working – this means checking the system against all user requirements previously defined in the design stage.

Scenario Testing

Part of agile development is using scenario testing to check that the scenarios of how users might use the system, defined in the design section, give the expected outcome. They are followed step-by-step and performed without knowledge of the implementation, to mimic users. Scenario testing was successfully performed at the end of development using the three scenarios defined with the personas, Ellie and Max.

4 Evaluation

To assess any real-world benefits of the developed prototype, an evaluation with individuals not involved with the project is key to get unbiased and empirical results. To evaluate the effectiveness of the application both qualitative (non-numerical) and quantitative (numerical) data is required, therefore the chosen research method must produce both data types. Multiple appropriate methods are available, such as interviews, questionnaires, and focus groups, however for this research an anonymous online questionnaire is chosen as the evaluation method due to its simplicity, anonymity, and allowance for participants to take part at a time that suits them. From the results of the questionnaire, data analysis can determine if the third research objective of this project has been met. This chapter contains an analysis of the questionnaire results, and then a personal reflection on how the project went, what was successful and what could have been better.

The study was performed with ethics approval from ERGO, reference number 78677.A1.

4.1 Questionnaire Results

The questionnaire first provided respondents with the relevant consent and participation information forms, adhering to Southampton University ethics policy. Then a video was displayed giving a demonstration of the prototype, showing features and how it is used by learners. Finally, 10 questions (**appendix A**) were asked with the first gauging the experience of the respondent, the next 8 asking about their experience of the gamification and feedback elements, and the last being open answer for additional comments. For the 8 experience questions, the Likert scale was used, allowing for 4 options ranging from strongly disagree to strongly agree. This scale was chosen as it is widely used, easily understandable, and quantitative. It had 15 respondents with a range of programming experience, who gave a range of responses. 3 respondents additionally gave an extra comment.

Motivation

For all motivation related questions, the most common response was ‘strongly agree’, and over 85% of responses were positive. This indicates that most users found the application to be motivating. The results also indicate that the gamification aspects were successful in making the experience of learning with the application fun; in response to the statement ‘I find the application engaging and fun’, most answered ‘strongly agree’

Figure 9: Bar chart of responses to question asking whether users were motivated

and numerically, the mean response was 3.5 where 4 is the best possible. Therefore, the statement that a programming self-learning platform should use gamification to motivate the learner, as stated in the summary of the literature review, has been successfully addressed.

Feedback

For the two feedback related questions, the most common responses were ‘strongly agree’ and ‘agree’, and overall, 100% were positive. These responses indicate that users found the feedback provided by the application to be helpful, meaning the other statement that a programming self-learning platform should provide effective feedback to allow the learning cycle to progress, as in the summary, has been successfully addressed.

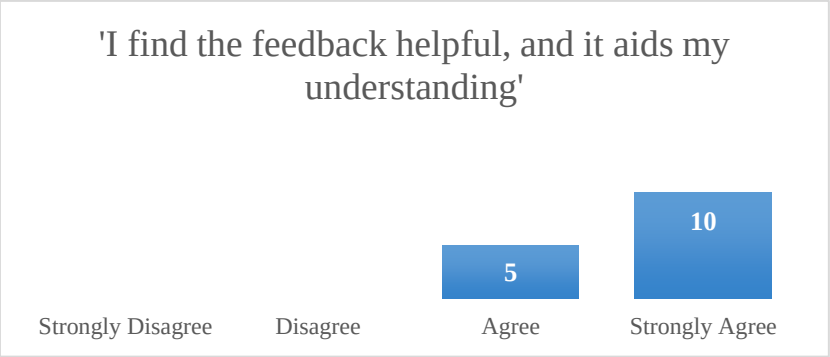


Figure 10: Bar chart of responses to question asking whether users found feedback helpful

Effectiveness

Although the main focusses of this application are gamification, feedback, and abstraction, it is still important that it is effective at teaching programming to beginners. Therefore, the statement ‘My understanding of coding is enhanced through using this application’ was included to gauge the teaching ability. The respondents overwhelmingly put ‘agree’ as their response, but the fact that this response was more negative than that of other statements, and 2 respondents disagreed means the teaching was not as effective as it could have been. This could be because most research was focussed on gamification and feedback, instead of methods to teach concepts, so the educational content suffered slightly as a result, however being a prototype, this is not a failure of the aims of the project.

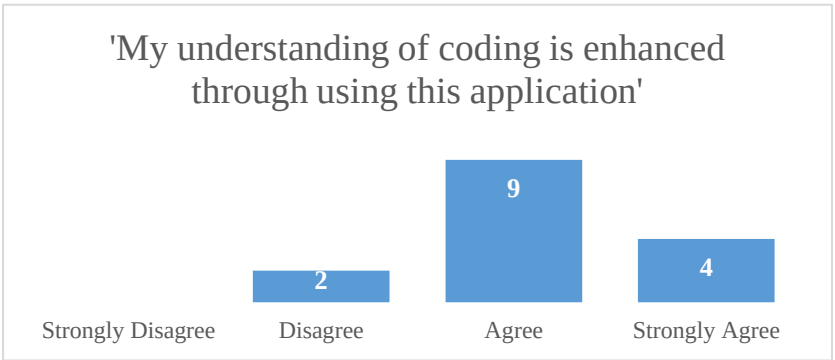


Figure 11: Bar chart of responses to question asking whether users gained a better understanding of coding

Experience

There are several trends in the data where respondents of similar programming experience give similar answers; For example, respondents who indicated higher experience in the first question found their coding to be enhanced less by the application. This was expected as the levels were mostly made with beginners as the focus but does indicate a lack of advanced educational content for those already somewhat familiar with programming. Similarly, those with lesser experience found the feedback provided by the application to be more helpful than those with greater. Again, this was expected as the error messages are meant to be clear and understandable for beginners to help them fix bugs.

Comments

In the open-ended question responses, respondents identified the sandbox mode and level importation as especially helpful features for experimentation and expandability. There was also a criticism of the train level track system, saying that it could be unwieldy and get confusing for larger tasks. This could be the track placement system being too restrictive and becoming overbearing for complex programs, or it could be that the concept of a train track as a representation of a program leads to an excess of redundant tracks that don't contribute to program meaning, for large programs.

Based on the positive responses for both motivation and feedback, it is my opinion that the prototype shows real world benefits and can act as a model for new e-learning platforms that have motivation as a focus. Additionally, the prototype contains replicable features that have proven to be successful, such as the ability to try out code in a sandbox, and a progression from an abstracted language to a text-based one.

4.2 Reflection

This section covers the successes and limitations of the project and developed application and discusses what may have been the cause for each.

Successes

The primary success of this project is that the developed prototype application meets all the requirements defined in the application chapter and was evidenced by the questionnaire to have achieved the second research objective: that the application addresses the identified issues and motivates student learning. Furthermore, the fact that students who tried the application found it to be fun, shows that gamification was successful in improving motivation. Another success is that the prototype is very extensible and can be improved without requiring further development. This is accomplished by the custom level feature allowing for user-created levels to be played, and the pseudocode-like language supporting more constructs and syntax than necessary for just the pre-made levels. Finally, during the research and report write up stages of this project, the time was managed properly, and work was spread out evenly, due to proper blocking and task estimation in the creation of the Gantt chart (**Table 1**).

Limitations

The main limitation of this project is the limited time permitted to complete it, causing the scope to be limited. This meant the sample size of the study had to be small which limits the reliability of the data and increases the likelihood of biases affecting results. Similarly, the stakeholder definition had to be limited to undergraduate University of Southampton students, potentially reducing the diversity of personas considered and limiting the reach any benefit borne from the project could have. An additional limitation of the project is that the train levels did not end up as intuitive as intended and they can become cumbersome with tasks involving large programs, which could limit the effectiveness of the application at increasing motivation. Finally, development of the application ended up taking longer than planned in the Gantt chart. This was due to a design oversight where the requirement for the parser to return custom, beginner-friendly errors was not considered, resulting in the parser being made from scratch rather than using a parser generator as had been initially planned. This did not affect the final application or report however, as extra time had been allotted for such a situation due to a mitigation made in the risk assessment (**Appendix C**).

5 Conclusion

5.1 Summary

The aim of this project is to improve learning to program for undergraduates who are not taking programming modules, by focusing on tackling the lack of motivation, support, and the difficulty of the subject. To achieve this, three research objectives were made:

1) To research and find factors that hinder self-learning when starting to learn to code.
2) To develop a prototype application which could address these factors and motivate student's learning.
3) To evaluate how effective the application is at motivating learning and teaching programming to beginners.

The first objective was fulfilled in the literature review chapter, where the factors hindering self-learning and their solutions were identified. This was achieved through a review of relevant and recent literature to find study-backed findings. Evidence of the second objective's fulfilment forms the application chapter, where the prototype application was designed and developed following the agile methodology, then extensively tested to ensure satisfaction of the requirements. The third objective was fulfilled in the evaluation chapter, which found that the application is very effective at motivating learning and quite effective at teaching programming to beginners, though some small issues were identified that limited its teaching ability – these are discussed in the future work section.

In conclusion, all research objectives have been fulfilled and the resultant application has been shown to be effective in addressing the issues identified in the literature review, therefore presenting a real-world benefit to undergraduate students, not taking programming modules, but attempting to learn to program. The prototype application developed can act as a model for future e-learning tools to involve a transition between different levels of abstraction to make learning to program easier.

5.2 Future Work

Due to the limited time available to complete this project, the research performed, and methods tested to try to improve programming e-learning learning had to be limited themselves. This section discusses further work that was outside the scope of this project, but that could be done in future to improve programming e-learning further.

A simple improvement to the application to extend its benefits could be increasing the constructs it introduces in levels, for example by adding input or recursion support to the pseudocode-like language and adding levels that utilise those. This does not involve major development and would have a large impact on the application's usefulness. A more significant improvement could be further research and/or studies to determine a better abstract representation of a programming language to use to introduce coding, specifically one that does not become cumbersome when creating large programs. This would additionally benefit from a larger study, which this project could not utilise due to the time and resource constraints. Finally, there exist additional research areas that show promise to improving e-learning that could be further investigated, such as utilising

artificial intelligence to create personalised feedback for learners based on their code, or utilising social learning and online collaboration to improve motivation.

References

- [1] UCL (2023) Computing for Mathematical Physics Module Page. Available at: https://www.ucl.ac.uk/module-catalogue/modules/computing-for-mathematical-physics-PHAS0012 (Accessed 29 October 2023).
- [2] Mackay, B. S. (2022) 'Data for the Labelling, Modelling, and Predicting Cell Biocompatibility using Deep Neural Networks'. University of Southampton. Available at: http://eprints.soton.ac.uk/id/eprint/483187 (Accessed 20th November 2023).
- [3] Kadar, R., Mahlan, S. B., Shamsuddin, M., Othman, J., and Wahab, N. A. (2022) 'Analysis of Factors Contributing to the Difficulties in Learning Computer Programming among Non-Computer Science Students', *Proceedings of 2022 IEEE 12th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, Penang, Malaysia, pp. 89-94. doi: https://doi.org/10.1109/ISCAIE54458.2022.9794546
- [4] Soni, N., Sharma, E. K., Singh, N., and Kapoor, A. (2020) 'Artificial Intelligence in Business: From Research and Innovation to Market Deployment', *Procedia Computer Science*, 167, pp. 2200-2210. doi: https://doi.org/10.1016/j.procs.2020.03.272
- [5] Gillott, L., Joyce-Gibbons, A., and Hidson, E. (2020) 'Exploring and comparing computational thinking skills in students who take GCSE Computer Science and those who do not', *International Journal of Computer Science Education in Schools*, 3(4), pp. 3–22. doi: https://doi.org/10.21585/ijcses.v3i4.77
- [6] Morris, T. H. (2019) 'Self-directed learning: A fundamental competence in a rapidly changing world', *International Review of Education*, 65(4), pp. 633–653. doi: https://doi.org/10.1007/s11159-019-09793-2
- [7] Ihantola, P., Fronza, I., Mikkonen, T., Noponen, M., and Hellas, A. (2020) 'Deadlines and MOOCs: How Do Students Behave in MOOCs with and without Deadlines', *2020 IEEE Frontiers in Education Conference (FIE)*, Uppsala, Sweden, pp. 1-9. doi: https://doi.org/10.1109/FIE44824.2020.9274023
- [8] Stella, M., Kapuza, A., Cramer, C., and Uzzo, S. (2021) 'Mapping computational thinking mindsets between educational levels with cognitive network science'. *Journal of Complex Networks*, 9(6). doi: https://doi.org/10.1093/comnet/cnab020
- [9] Pelling, N. (2011) 'The (short) prehistory of "gamification"', *Funding Startups (& Other Impossibilities)*, 9th August. Available at: https://nanodome.wordpress.com/2011/08/09/the-short-prehistory-of-gamification/ (Accessed 30 October 2023).
- [10] Rodrigues, L., Toda, A. M., Oliveira, W., Palomino, P. T., Avila-Santos, A. P., and Isotani, S. (2021) 'Gamification Works, but How and to Whom? An Experimental Study in the Context of Programming Lessons', *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, pp. 184–190. doi: https://doi.org/10.1145/3408877.3432419
- [11] Palaniappan, K. and Noor, N. M. (2022). 'Gamification Strategy to Support Self-Directed Learning in an Online Learning Environment', *International Journal*

of Emerging Technologies in Learning (iJET), 17(3), pp. 104-116. doi: http://doi.org/10.3991/ijet.v17i03.27489

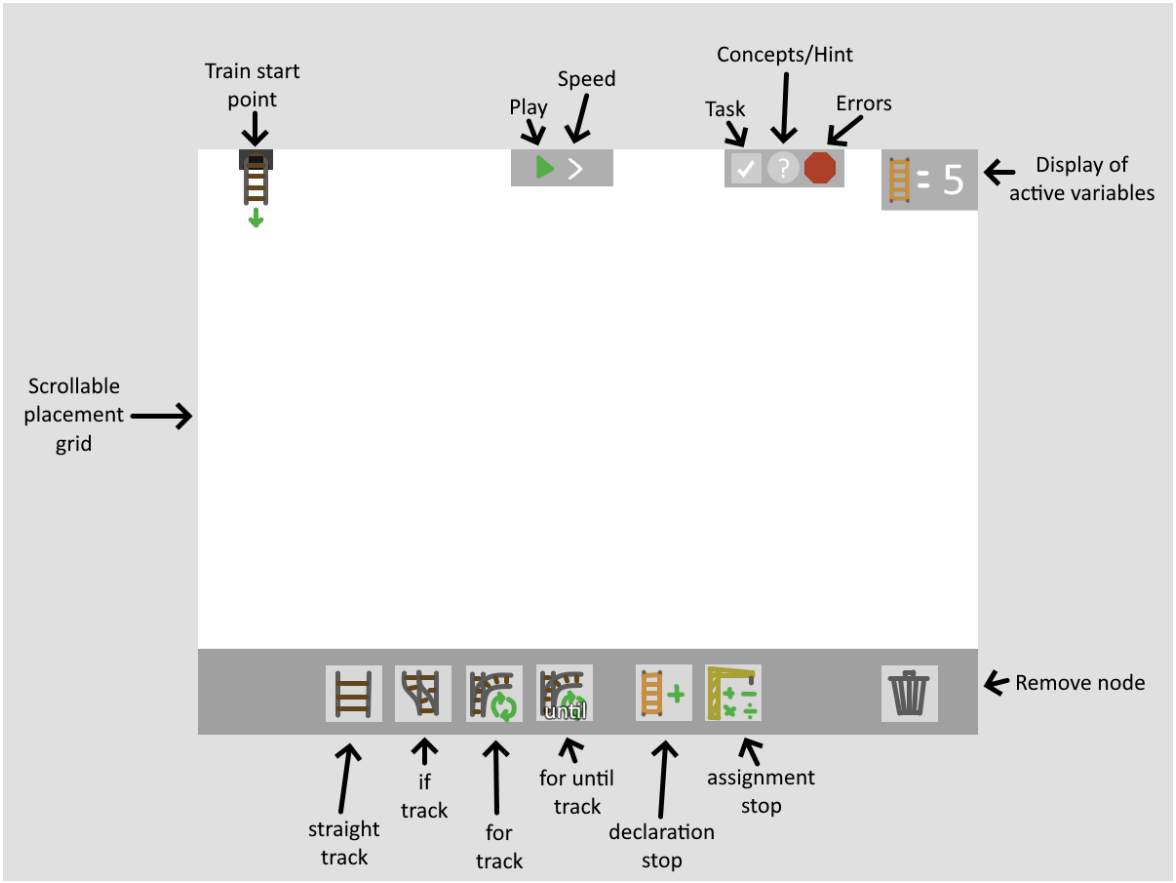
- [12] Kadar, R., Mahlan, S. B., Shamsuddin, M., Othman, J., and Wahab, N. A. (2022) 'Analysis of Factors Contributing to the Difficulties in Learning Computer Programming among Non-Computer Science Students', *2022 IEEE 12th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, Penang, Malaysia, pp. 89-94. doi: https://doi.org/10.1109/ISCAIE54458.2022.9794546
- [13] Aras, S., Gedikli, E., and Ozyurt, O. (2018) 'A Framework Based on Compiler Design Techniques for Programming Learning Environments', *proceedings of the 2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, Malatya, Turkey. pp. 1-6. doi: https://doi.org/10.1109/IDAP.2018.8620736
- [14] Ihantola, P., Fronza, I., Mikkonen, T., Noponen, M., and Hellas, A. (2020) 'Deadlines and MOOCs: How Do Students Behave in MOOCs with and without Deadlines', *2020 IEEE Frontiers in Education Conference (FIE)*, Uppsala, Sweden, pp. 1-9. doi: https://doi.org/10.1109/FIE44824.2020.9274023
- [15] Cheah, C. S. (2020). 'Factors Contributing to the Difficulties in Teaching and Learning of Computer Programming: A Literature Review', *Contemporary Educational Technology*, 12(2), article no: 272. doi: https://doi.org/10.30935/cedtech/8247
- [16] Alhazbi, S. (2014) "Using e-journaling to improve self-regulated learning in introductory computer programming course", *2014 IEEE Global Engineering Education Conference (EDUCON)*, Istanbul, Turkey, pp. 352-356. doi: https://doi.org/10.1109/EDUCON.2014.6826116
- [17] Toti, G., Chen, G., and Gonzalez, S. (2023) 'Teaching CS1 with a Mastery Learning Framework: Impact on Students' Learning and Engagement', *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, pp. 540–546. doi: https://doi.org/10.1145/3587102.3588844
- [18] Behrendt, M., and Machtmes, K. (2021) "Exploring the Catalyst Energizing the Kolb Learning Cycle," *Experiential Learning & Teaching in Higher Education*: 4(11). Available at: https://nsuworks.nova.edu/elthe/vol4/iss1/11 (Accessed 28th November 2023).
- [19] Conejo, G. G., Gasparini, I., and Da Silva Hounsell, M. (2019) "Detailing Motivation in a Gamification Process," 2019 *IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*, Maceio, Brazil, pp. 89-91. doi: https://doi.org/10.1109/ICALT.2019.00031
- [20] Denden, M., Tlili, A., Essalmi, F., and Jemni, M. (2020) "Students' learning performance in a gamified and self-determined learning environment," *2020 International Multi-Conference on: "Organization of Knowledge and Advanced Technologies" (OCTA)*, Tunis, Tunisia, pp. 1-5. doi: https://doi.org/10.1109/OCTA49274.2020.9151840
- [21] Benedikt, W., Klaus, Z., and John, H. (2020) "The Power of Feedback Revisited: A Meta-Analysis of Educational Feedback Research", *Frontiers in Psychology*, 10. doi: https://doi.org/10.3389/fpsyg.2019.03087
- [22] Azcona, D., Hsiao, IH., and Smeaton, A. F. (2019) "Detecting students-at-risk in computer programming classes with learning analytics from students' digital

footprints", *User Modeling and User-Adapted Interaction*, 29, pp. 759-788. doi: https://doi.org/10.1007/s11257-019-09234-7
- [23] Jiménez, S., Juárez-Ramírez, R., Castillo, V. H., Ramírez-Noriega, A., Márquez, B. Y., Alanis, A. (2021) "The Role of Personality in Motivation to use an Affective Feedback System", *Programming and Computer Software*, 47, pp. 793–802. doi: https://doi.org/10.1134/S0361768821080156
- [24] Yan, L., Hu, A., and Piech, C. (2019) "Pensieve: Feedback on Coding Process for Novices", *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*, Association for Computing Machinery, New York, NY, USA, pp. 253–259. doi: https://doi.org/10.1145/3287324.3287483
- [25] Shin, K., Kim, J., Kim, M. S., and Son, Y. (2021) "Effects of cognitive appraisal styles and feedback types on feedback acceptance and motivation for challenge", *Educational Psychology*, 41(7), pp. 902-921. doi: https://doi.org/10.1080/01443410.2020.1725449
- [26] Keuning, H., Jeuring, J., and Heeren, B. (2018) "A Systematic Literature Review of Automated Feedback Generation for Programming Exercises", *ACM Transactions on Computing Education*, 19(1). doi: https://doi.org/10.1145/3231711
- [27] Dohn, N. B. (2020), "Students' interest in Scratch coding in lower secondary mathematics", *British Journal of Educational Technology*, 51(1), pp. 71-83. doi: https://doi.org/10.1111/bjet.12759
- [28] Seraj, M., Katterfeldt, E. S., Bub, K., Autexier, S., and Drechsler, R. (2019) "Scratch and Google Blockly: How Girls' Programming Skills and Attitudes are Influenced", *Proceedings of the 19th Koli Calling International Conference on Computing Education Research (Koli Calling '19)*, Association for Computing Machinery, New York, NY, USA, 23, pp. 1–10. doi: https://doi.org/10.1145/3364510.3364515
- [29] Estevez, J., Garate, G., and Graña, M. (2019) "Gentle Introduction to Artificial Intelligence for High-School Students Using Scratch", *IEEE Access*, 7, pp. 179027-179036. doi: https://doi.org/10.1109/ACCESS.2019.2956136
- [30] Fagerlund, J., Häkkinen, P., Vesisenaho, M., Viiri, J. (2021) "Computational thinking in programming with Scratch in primary schools: A systematic review", *Computer Applications in Engineering Education*, 29, pp. 12–28. doi: https://doi.org/10.1002/cae.22255
- [31] Zhang, L., Nouri, J. (2019) "A systematic review of learning computational thinking through Scratch in K-9", *Computers & Education*, 141. doi: https://doi.org/10.1016/j.compedu.2019.103607
- [32] Amanullah, K., and Bell, T. (2019) "Analysis of Progression of Scratch Users based on their Use of Elementary Patterns", *2019 14th International Conference on Computer Science & Education (ICCSE)*, Toronto, ON, Canada, pp. 573-578. doi: https://doi.org/10.1109/ICCSE.2019.8845495
- [33] Venter, M. (2020) "Gamification in STEM programming courses: State of the art", *2020 IEEE Global Engineering Education Conference (EDUCON)*, Porto, Portugal, pp. 859-866. doi: https://doi.org/10.1109/EDUCON45650.2020.9125395
- [34] Buffardi, K., and Wang, R. (2022) "Integrating Videos with Programming Practice", *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '22)*, Association for Computing

Machinery, New York, NY, USA, 1, pp. 241–247. doi: https://doi.org/10.1145/3502718.3524778

- [35] Shen, R., and Lee, M. J. (2020) "Learners' Perspectives on Learning Programming from Interactive Computer Tutors in a MOOC", *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Dunedin, New Zealand, pp. 1-5, doi: https://doi.org/10.1109/VL/HCC50065.2020.9127270
- [36] Sharp, J. H. (2019) "Using Codecademy Interactive Lessons as an Instructional Supplement in a Python Programming Course", *Information Systems Education Journal*, 17(3), pp 20-28. Available at: https://isedj.org/2019-17/n3/ISEDJv17n3p20.html (Accessed 5 December 2023).

Appendix A – Train Level Story Board



Appendix B – Questionnaire Form

Application Survey

Part I

Please place a tick next to the response you find applies most out of none, a little, some, expert.

How much prior experience do you have with programming?

None: **A little:** **Some:** **Expert:**

Part II

For each statement, please place a tick under the response you find applies most out of strongly agree, agree, disagree, and strongly disagree.

Statements	**Strongly Agree**	**Agree**	**Disagree**	**Strongly Disagree**
I find the application engaging and fun				
I find the progress visualisation encourages me to complete the levels				
I find the train levels are intuitive and completing them is fun				
My understanding of coding is enhanced through using this application				
I find the feedback helpful, and it aids my understanding				
I find the error messages to be clear and educational				
In general, I feel motivated to learn when using this application				
I will recommend this application to my friends				

Part III

If you have any other comments, such as features you thought were missing or other ways you found to use the application, please write them in the box below.

Appendix C – Risk Assessment

Risk	Likelihood	Severity	Risk Exposure	Mitigation
A task ends up taking longer than initially estimated	4	3	12	Allow extra time for each task in the time management plan to account for overruns
Loss of application project files	2	4	8	Periodically back up all project files onto a separate USB storage device
Illness or other circumstance prevents project completion	1	5	5	Apply for a project extension and discuss the circumstances with my supervisor
Lack of internet or device preventing project submission	1	5	5	Utilise university computers to complete and submit the project
Not all development requirements can be met	2	3	6	Prioritise more important requirements during development.