CodeBot Quest: Teaching Programming Using a Virtual Robot

Final report

Submitted for the BSc in

Computer Science with Software Engineering

April 2025

by

Patricia Isabel Rodrigues Cotovio

Word count: XXXXX

Abstract

This project presents CodeBot Quest, a 2D educational platformer game designed to introduce young learners to basic programming concepts through an engaging, fun and interactive game environment. The game aims to explain programming for young school-aged children and/or beginner programmers by having in-game mechanisms connected to some programming concepts like loops, variables, and conditionals. The game is developed using Unity and scripted in C# with Visual Studio. CodeBot Quest has gamification elements such as life systems and, progressive challenges to increase motivation and retention of users.

The software development followed an Agile methodology, allowing iterative design implementation and user feedback. Testing was conducted throughout the game development, with a small sample of adult participants who had no prior programming experience with the aim to simulate the intended beginner audience of this game. Gameplay observation and post-play feedback were the evaluative methods used. The results helped shape the final product and demonstrated that a narrative-driven, game-based approach can significantly help in the comprehension and retention of core programming principles.

The final product consists of a working prototype that includes a login system connected to a database, a main menu, a loading screen, and a single playable level. This demo level integrates guided panels and instructional prompts to assist users in completing the programming challenge. Alongside the software, custom assets and an evaluation report have been produced. The project contributes to the field of educational technology by showing how serious games can present abstract computing principles in a more accessible and engaging setup, encouraging motivation and early self-directed learning among novice programmers.

Contents

[Abstract i](#_Toc195721056)

[1 Introduction 3](#_Toc195721057)

[1.1 Background to the project 3](#_Toc195721058)

[1.2 Aims and objectives 3](#_Toc195721059)

[1.3 Research question 4](#_Toc195721060)

[2 Literature review 5](#_Toc195721061)

[2.1 Game-Based Learning Platforms 5](#_Toc195721062)

[2.2 Physical Coding Tools 5](#_Toc195721063)

[2.3 Visual Programming Languages 5](#_Toc195721064)

[2.4 Effectiveness and Engagement 5](#_Toc195721065)

[3 Requirements 6](#_Toc195721066)

[3.1 Product requirements 6](#_Toc195721067)

[3.2 Functional requirements 6](#_Toc195721068)

[3.2.1 Interfaces 6](#_Toc195721069)

[3.2.2 Functional Capabilities 6](#_Toc195721070)

[3.2.3 Performance Levels 6](#_Toc195721071)

[3.2.4 Data Structures/Elements 6](#_Toc195721072)

[3.2.5 Safety 6](#_Toc195721073)

[3.2.6 Reliability 6](#_Toc195721074)

[3.2.7 Security/Privacy 6](#_Toc195721075)

[3.2.8 Quality 6](#_Toc195721076)

[3.2.9 Constraints and Limitations 6](#_Toc195721077)

[3.2.10 Performance requirements 6](#_Toc195721078)

[3.3 Design constraints 6](#_Toc195721079)

[4 Design 7](#_Toc195721080)

[4.1 Software design 7](#_Toc195721081)

[4.2 Hardware design 7](#_Toc195721082)

[4.3 Experimental design 7](#_Toc195721083)

[5 Implementation and testing 8](#_Toc195721084)

[5.1 Implementation 8](#_Toc195721085)

[5.2 Testing 8](#_Toc195721086)

[6 External concerns 9](#_Toc195721087)

[6.1 Legal, social and ethical issues 9](#_Toc195721088)

[6.2 Commercial issues and exploitation 9](#_Toc195721089)

[7 Evaluation and discussion of results 10](#_Toc195721090)

[7.1 Programme specific concerns 10](#_Toc195721091)

[8 Conclusion 11](#_Toc195721092)

[8.1 Project management 11](#_Toc195721093)

[8.2 Risk management 11](#_Toc195721094)

[8.3 General conclusions 11](#_Toc195721095)

[References 12](#_Toc195721096)

[Appendix A – Interesting but not vital material 14](#_Toc195721097)

[Appendix B – Other things which may be useful 15](#_Toc195721098)

# Introduction

This section introduces the project, defining its context, objectives, and expected results. It sums up the challenges associated with teaching programming to beginners and describes how *CodeBot Quest* tries to address these challenges via an interactive and educational game-based solution. The section defines the project’s background, aims and objectives, and research question, setting the foundation for the technical and evaluative content that follows.

## Background to the project

In recent years, computer science has become a huge part of modern education in the UK and around the world. More and more school systems are integrating it into their curriculum. This is because computer science teaches problem solving and logical thinking, which are both very important for young learners. Because of this, it's important that students understand how computer science works from an early age.

Many individuals may think that learning programming is difficult because of the current traditional way its taught, which mainly involve textbooks and require students to write code from scratch, often with minimal feedback or interactivity which demotivates students, leading them to give up because they find the subject unengaging and complex.

Educational games have the potential to make learning more fun and exciting for everyone. They are a good tool to help students understand coding in a way that’s easier and more interactive through feedback, rewards, and a sense of accomplishment which helps keeping individuals engaged and motivated to keep learning.

*CodeBot Quest*, is an educational game that aims to improve the learning experience for students trying to understand programming. It is a 2D platform game that teaches coding using a robot as the main character. The game was fully designed to help people learn programming logic through an engaging and fun play.

## Aims and objectives

The main goal of this project is to help beginners, especially school pupils, learn programming in a way that is fun, interactive, and easy to understand. The game focuses on a “learning by doing” approach rather than relying on reading theory or memorising syntax, rendering it more appropriate for those who are completely new to coding.

To achieve this goal, the following objectives have been set:

Develop a working 2D game that introduces basic programming concepts such as loops, variables, and conditionals.

Use a robot character in the game to help visualise and demonstrate the programming logic chosen by the player.

Implement gamification features, including a life system where each level gives the player three lives. These lives convert into stars that are tied to the player's account to help encourage progress and motivation.

Continuously test the game with beginners or individuals who have no previous coding experience, in order to collect responses and make improvements based on that feedback.

## Research question

This project was drove by the hypothesis that learning programming can be made easier, enjoyable and playful through educational games. Since many beginners struggle with traditional approaches such as writing code from scratch or reading through textbooks, the goal was to explore whether a different, more interactive approach could help. The research question helped shape the design of the game and the way testing was carried out, especially when collecting feedback from people with no coding experience.

The main question this project tries to answer is:

**Can a game-based approach help beginners with no programming experience understand basic coding concepts effortlessly in an engaging and clearer way?**

# Literature review

The integration of game-based learning (GBL) into computer science education has gained prominence as an effective strategy to engage young learners. This section reviews various tools and initiatives that employ GBL to teach programming concepts to children, focusing on platforms like CodeCombat, Tynker, Cubetto, and Scratch. The effectiveness of these tools in enhancing student engagement and learning outcomes is examined through existing studies and reports.

## Game-Based Learning Platforms

Platforms such as CodeCombat and Tynker have been developed to introduce programming concepts through interactive gameplay. As illustrated in Figure 1, the "Go Go Go" level in CodeCombat allows learners to write code to navigate characters through challenges, providing immediate feedback that enhances learning (CodeCombat, 2019). Similarly, Tynker offers a block-based coding environment that simplifies complex programming concepts for young learners (Tynker, 2025). Studies have shown that these platforms effectively maintain student interest and improve programming skills (Elsawah & Thabet, 2022).



Figure 1: CodeCombat "Go Go Go" Level Interface. Screenshot by Author.

## Physical Coding Tools

Cubetto, a wooden robot designed for children aged three and up, teaches programming through tactile interaction without the need for screens. Children input commands using coloured blocks, which Cubetto executes, thereby introducing them to sequencing and logic (Primo Toys, 2016). This hands-on approach aligns with Montessori principles and has been praised for making abstract programming concepts accessible to young learners (Learning in Hand, 2017). Figure 2: illustrates the Cubetto robot in action, demonstrating its capability to teach coding concepts to young children through tactile interaction (Primo Toys, 2016).

[](https://www.youtube.com/embed/Kp1p2lh2D64?feature=oembed)

Figure 2: Cubetto

Robot Demonstration. Source: Primo Toys (2016).

## Visual Programming Languages

Scratch, developed by the MIT Media Lab, is a visual programming language that enables children to create interactive stories and games. Its block-based interface simplifies coding syntax, allowing learners to focus on logical structures and problem-solving (CodeYoung, 2024). The use of Scratch in educational settings has been associated with increased creativity and improved understanding of programming fundamentals (STEM Genius, 2023).

## Effectiveness and Engagement

Research indicates that GBL tools not only make learning programming more engaging but also enhance comprehension and retention of concepts. A scoping review by Barradas et al. (2023) concluded that educational games provide authentic learning experiences and immediate feedback, which are crucial for developing computational thinking skills. Furthermore, these tools have been found to be effective in diverse educational settings, accommodating various learning styles and needs.

# Requirements

This project focus on a 2D educational platform game that is developed and designed to teach beginner-level programming concepts through gameplay. The requirements were originally outlined in the Project Definition Document (PDD) submitted and have been refined based on development progress and practical considerations that occurred during development and implementation.

The game is being developed in Unity and targets children or beginners with no prior programming experience. As a result, the requirements emphasise simplicity, accessibility, and intuitive interaction. Due to the developer’s limited time and resources, and the decision to prioritise a playable prototype over a full multi-level product, some deliverables have been scaled accordingly.

The requirements have been divided into product requirements, functional requirements, and design constraints, as detailed below.

## Product requirements

* A single working level (level 3) demonstrating a basic programming concept (e.g., sequence, conditionals).
* A login and registration system connected to a MySQL database.
* A user interface that includes a main menu, loading screen, level selection panel, and game UI.
* Educational prompts and panels that guide the player throughout the level.
* A life system (3 lives per level) linked to a star reward mechanic.
* Integration of accessibility features such as clear fonts and simple colour contrast.

## Functional requirements

### Interfaces

* Unity-based UI using Canvas system.
* User login and registration that are handled via PHP and backed by MySQL.
* Visual prompts and tooltips in-game.

### Functional Capabilities

* User account creation and login validation on the sign in scene.
* Game control: moving the player character, jumping, triggering events.
* Game logic linked to basic programming tasks.
* Star calculation based on lives remaining.
* Navigation between game screens (menu > level > game > game over > return to menu).

### Performance Levels

* Game should run at 60 FPS on standard school-level Windows PCs.
* Level loading time should be under 5 seconds on average.

### Data Structures/Elements

* User credentials and star progress stored in a relational database.
* Game state (lives, stars, hints shown) held in memory per session.

### Safety

* No personal or sensitive data collected beyond login name and password.
* Inappropriate usernames filtered.

### Reliability

* Game should recover gracefully if player loses all lives
* Show a game over screen,
* Possibly give the player the option to retry or return to the menu,
* Reset things cleanly without breaking any core functionality.
* Backend login must handle invalid or repeated login attempts safely
* If a user types a wrong password or username, the system should just return a safe message like “Incorrect login” — not crash or show sensitive errors.
* It should also stop someone trying to spam login attempts, like rate-limiting them or locking after X failures, if implemented.
* It must avoid exposing errors that might help someone exploit your database.

### Security/Privacy

* Passwords stored securely using hashing (if implemented).
* Anonymous gameplay supported for testing.

### Quality

* Smooth and bug-free gameplay in the playable level.
* Interface tested for ease of use by non-programmers.

### Constraints and Limitations

* The game includes only one playable level due to limited development time and scope prioritisation.
* All testing was carried out with adult users who had no prior coding experience, as access to pupils (the target audience) was not available.
* The majority of assets were created manually by the developer; however, some visual elements were purchased from the Unity Asset Store due to time and skill constraints.
* Without the purchased assets, the game's final visual quality would have been significantly lower or would have required considerably more development time.
* The original development period (October 2024 – April 2025) was disrupted due to delayed Student Finance support. The university services advised pausing studies until the issue was resolved, which pushed back the project start to November 2024.
* As a working student, additional pressure was placed on managing multiple first-trimester modules, which had more immediate deadlines, reducing the amount of time available for focused development on this project.

### Performance requirements

* Must run on machines with basic integrated graphics and at least 4GB RAM.
* Game must not freeze or crash under standard use.

## Design constraints

Several constraints influenced the design and development process of CodeBot Quest. The game was developed using Unity, Visual Studio, and Adobe Photoshop, all under student subscriptions.

Budget limitations also had a substantial impact in the design decisions and development of this game, as some assets were purchased from the Unity Asset Store while a few others were created manually. This implementation was necessary to maintain a good visual standard and optimize time efficiency during development. Without these purchases, the overall appearance of the game would have been much more basic, or it would have required significantly more time to develop original assets from scratch, time that was not available.

These financial and time constraints shaped the visual scope of the game, limiting some elements of polish or additional features that may have otherwise been included in a fully resourced environment.

# Design

If your project involves the development of software and/or hardware, then you will need to include a section in which you describe its design in detail. If you conduct any experiments (either in a research-oriented project or simply doing user evaluation) then you should describe their design and methodology here.

Delete the red paragraph and replace this one with your content (use the “Normal” paragraph style).

## Software design

Typical content will be detailed software design, from architecture to implementation level. As well as your text, you should include UML diagrams, including class structures, activity and sequence diagrams as appropriate. Don’t just drop diagrams in willy-nilly, though. Use them strategically to illustrate points in your text. Remember that ‘a picture is worth a thousand words’ (we don’t apply this rule literally) but pictures on their own don’t explain everything.

If your project requires user interface design, don’t forget to include that. Screenshots, wireframes and other diagrams are welcome.

Delete the red paragraph and replace this one with your content (use the “Normal” paragraph style).

## Hardware design

If your project involves building hardware, give full details about the process here. Include diagrams as appropriate Use them strategically to illustrate points in your text. Remember that ‘a picture is worth a thousand words’ (we don’t apply this rule literally) but pictures on their own don’t explain everything.

If your project requires electronics and/or mechanical design, don’t forget to include that. Photos, CAD drawings, electronic schematics and other diagrams are welcome.

Delete the red paragraph and replace this one with your content (use the “Normal” paragraph style).

## Experimental design

If you are going to evaluate your software or hardware by means of any tests or surveys, then explain their design here. If you are doing other experiments (for example measuring the performance of algorithms, extracting data from environmental monitoring systems or evaluating the performance of mechanisms) then you should explain how you have designed the experiments, how they must be conducted and what you expect to learn from them. This is especially important for research-oriented projects.

Delete the red paragraph and replace this one with your content (use the “Normal” paragraph style).

# Implementation and testing

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

## Implementation

In this section you will describe what you did, and why you made the important decisions affecting your actions. It’s not a diary – don’t write a blow-by-blow account of every little thing that happened. Be selective and report those choices and techniques which made a difference. Make sure you discuss what options you considered. Explain how the criteria and methodology you used to select amongst different options (which tools are most appropriate, for example).

It may help to imagine that you are reading this project in the future, trying to replicate the work without making the same mistakes along the way. What would you need to know to make your job easier, and what is unimportant or obvious? Explain how you implemented the design in the previous chapter.

This is the place in which you would explain any novel or especially complex algorithms, data structures or systems you have used.

Make it clear what you have done, and what is pre-existing. For example, if you are using third party software libraries, describe how you have used them, and how they have benefited your project rather than simply what they do. If you have built on a framework, make it clear how you have developed new functionality.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

## Testing

If you are developing software or hardware, you must test it. This section should explain how your work will be (or has been) tested.

You should have a test plan at the very least (full details of it and its results if required can go in an appendix). Ideally, you will have automated tests for any software you build. You will also define user acceptance tests, or something similar which can be used to determine whether your output meets the requirements stated earlier. Explain how and when the tests should be conducted.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

# External concerns

This section considers factors external to your project. In your PDD, you considered external factors which might influence or be influenced by the process of undertaking your project. In this section, you should consider the way in which your project, its results, any knowledge gained from it or its commercial exploitation might affect the wider world.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

## Legal, social and ethical issues

In your PDD, you considered the ethics involved with *doing* your project. Now you should identify and discuss the legal, social, ethical and professional issues raised by the project as a whole, and any artefacts you produced. If you created software to track individuals, or a laser-wielding robot hardware, what effects on the world in general need to be considered? How will you address these issues. Many things can be used for good or bad purposes. Can yours, and what will you do about it?

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

## Commercial issues and exploitation

Think about what you have created during this project. It may be hardware, software, knowledge or a combination of these. How could you exploit these commercially? Discuss the merits of your work, other commercial entities which might be similar, and consider the costs and potential profits of using your work commercially.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

# Evaluation and discussion of results

This section evaluates the *software, hardware or other artefact* you have developed. You should compare it with the original specification and see how well it satisfies the requirements. You may wish to refer back to your aims and objectives at this point. You should report the results of user testing and a summary of feedback if that has been collected.

Evaluation should be rigorous and objective. Avoid opinion and unsubstantiated statements.

If you have done experiments, then the results of these should be reported and discussed here.

If you have involved people in doing user evaluations, that information should be include here.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

## Programme specific concerns

Although this does not have to be a section on its own, your report must make clear how the work you have done demonstrates the programme specific competency for your degree programme. There is a full list of these in the assignment specification.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

# Conclusion

In this section you should evaluate the *project* as a whole, and the *process* by which you undertook it.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

## Project management

What methodology did you actually use? Was it the same as you originally planned? Was your methodology appropriate (and did you stick to it)? Was your time planning good? Did you meet regularly with your supervisor? What changes did you make as a result of your midway review meeting? What have you learned from the process? What would you do better/differently if you had more time? Include the final version of your Gantt chart and comment on its use.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

## Risk management

In your PDD you conducted an initial risk management exercise. How appropriate did this turn out to be? If any of the predicted problems occurred, how good were your mitigation plans? If other events happened which caused problems, what could you have done to predict and mitigate them?

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

## General conclusions

In this section you should evaluate the *project* as a whole, and draw conclusions from the work you have done. Ask yourself what the project has achieved – what is its contribution? Has it met its initial aims and objectives? If not, why? How does the work you have done enhance the field in general? What has been learned from the project? If you have a well defined research question, has it been answered? What do the results mean?

Sometimes, it’s appropriate to include a subsection on ‘Further work’, making suggestions of how to proceed and what could be done to enhance the project in future.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

References

References must be formatted in the correct manner. For this assignment you must use the University of Hull’ approved variant of the Harvard referencing style (Fallin 2019), fully described at https://libguides.hull.ac.uk/referencing/harvard. Note that the details of the expected format vary depending on the type of document being referenced. Make sure you are familiar with them. If you use reference management software such as Zotero, EndNote or RefWorks, then you should be able to export a table of references in the correct format, which will save you work.

Every reference should have at least one citation in the text. Most will probably be in the ‘Background’ or ‘Literature review’ sections.

Remember that there is a difference between references and a bibliography. You will certainly need references, but a bibliography is optional.

There is much more information and guidance about referencing on the library’s website at https://libguides.hull.ac.uk/referencing/home

Some examples, illustrating different types of source:

Bahraini, M.S., Bozorg, M., Rad, A.B., (2018). SLAM in dynamic environments via ML-RANSAC. *Mechatronics* 49, 105–118.

Fallin, L., (2019)*. LibGuides: Referencing your work: Harvard Hull.* Available online: http://libguides.hull.ac.uk/referencing/harvard (accessed 10/10/2019).

Janis, I., (1972). *Victims of Groupthink: A psychological study of foreign-policy decisions and fiascoes.* Houghton Mifflin, Boston.

Office For Students (2018) *. Securing student success: Regulation framework for higher education in England*. Available online: https://www.officeforstudents.org.uk/media/1406/ofs2018\_01.pdf (accessed 10/10/2019)

Schmuck, P., Chli, M., (2019). CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams. *Journal of Field Robotics* 36, 763–781.

Barradas, A., et al. (2023). Game-based learning in computer science education: a scoping review. *International Journal of STEM Education*, 10(1). <https://doi.org/10.1186/s40594-023-00447-2>​

CodeCombat. (2019). CodeCombat Implementation Study Summary. *CodeCombat*. <https://codecombat.com/images/pages/impact/pdf/CodeCombat_ImplementationStudy_Summary.pdf>​

Elsawah, W., & Thabet, R. A. (2022). The Effectiveness of Tynker Platform in Helping Early Ages Students to Acquire the Coding Skills Necessary for 21st Century. *ResearchGate*. <https://www.researchgate.net/publication/364668329_The_Effectiveness_of_Tynker_Platform_in_Helping_Early_Ages_Students_to_Acquire_the_Coding_Skills_Necessary_for_21st_Century>​

Learning in Hand. (2017). Start Youngsters on a Coding Adventure with Cubetto. <https://learninginhand.com/blog/cubetto>​[Learning in Hand with Tony Vincent+1Wikipédia, l'encyclopédie libre+1](https://learninginhand.com/blog/cubetto?utm_source=chatgpt.com)

Primo Toys. (2016). Meet Cubetto, the Wooden Gadget Teaching Children to Code. <https://pressreleases.responsesource.com/news/91782/meet-cubetto-the-wooden-gadget-teaching-children-to-code/>​[pressreleases.responsesource.com](https://pressreleases.responsesource.com/news/91782/meet-cubetto-the-wooden-gadget-teaching-children-to-code/?utm_source=chatgpt.com)

STEM Genius. (2023). Scratch Coding: Essential Insights for Beginners. <https://stem-genius.com/what-is-scratch-coding-essential-things-you-need-to-know/>​[stem-genius.com](https://stem-genius.com/what-is-scratch-coding-essential-things-you-need-to-know/?utm_source=chatgpt.com)

Tynker. (2025). Coding for Kids. *Tynker*. <https://www.tynker.com/>​

CodeYoung. (2024). Everything You Need to Know About Scratch Programming for Kids. <https://www.codeyoung.com/blog/scratch-101-everything-you-need-to-know-about-scratch-programming-for-kids>

Primo Toys. (2016). *Meet Cubetto* [Online Video]. Available at: <https://www.youtube.com/watch?v=Kp1p2lh2D64> (Accessed: 16 April 2025).

Appendix A – Interesting but not vital material

Appendices are used to include information which may be of interest but is not necessary for the reader. *You do not have to include appendices if there is no need for them*.

You might, for example, want to include some details of a particular piece of software (an API, perhaps) or hardware which your project uses. This might be something that a reader might wish to consult, but you wouldn’t want to include in the main body of the report. You could also put raw data from experiments in an appendix, or perhaps survey results. It should still be information of relevance, but nothing that everyone would be expected to read.

If you wish to refer to elements of your PID, you could include them in appendices.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

Appendix B – Other things which may be useful

You can have more than one appendix, or none at all. Give them meaningful names and titles (not the ones given here), so that you can refer to them in the text, and so that they appear in the table of contents.

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).