Turtlebot Software for Schools Outreach

Cate Fitzpatrick

Department of Computer Science

Aberystwyth University

Wales

September

2025

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science

Degree: MSc Advanced Computer Science

Module: Dissertation (CSM9060)

Supervisor: Dr Neal Snooke

Abstract

The abstract stands alone as a very short version of the dissertation.

The abstract should state the scope and principal objectives of the project, describe the methods, summarize the results and state the principal conclusions.

**(Max. 500 words.)**

Declaration of originality

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

**Name:** Cate Fitzpatrick   
**Date:** dd/mm/yy

Consent to share this work

* By including my name below, I hereby agree to this thesis being made available to other students and academic staff of the Department of Computer Science, Aberystwyth University.

**Name:** Cate Fitzpatrick   
**Date:** dd/mm/yy

Acknowledgement

To whoever has the patience to read this :-)

This section is customary, but not obligatory. It makes a brief statement of thanks to those who have helped.

Table of Contents

[Chapter 1. Introduction 1](#_Toc205819992)

[Background 1](#_Toc205819993)

[Project aims and objectives 1](#_Toc205819994)

[How the project was tackled 1](#_Toc205819995)

[Chapter 2. Literature review 3](#_Toc205819996)

[Introduction 3](#_Toc205819997)

[The early history of Logo and Turtles 3](#_Toc205819998)

[Logo 3](#_Toc205819999)

[The introduction of Turtles 3](#_Toc205820000)

[The introduction of Turtle geometry 4](#_Toc205820001)

[Logo since 1975 4](#_Toc205820002)

[Later developments in Turtles and other robots 4](#_Toc205820003)

[BBC Buggy 5](#_Toc205820004)

[The Valiant Turtle 5](#_Toc205820005)

[LEGO Mindstorms 5](#_Toc205820006)

[TurtleBot 5](#_Toc205820007)

[Conclusion 5](#_Toc205820008)

[Chapter 3. Requirements 6](#_Toc205820009)

[Project requirements 6](#_Toc205820010)

[Methodology 6](#_Toc205820011)

[Chapter 4. Spike work 7](#_Toc205820012)

[Picking software 7](#_Toc205820013)

[Java 7](#_Toc205820014)

[Python 7](#_Toc205820015)

[Conclusion 8](#_Toc205820016)

[Executing files 8](#_Toc205820017)

[The subprocess module 9](#_Toc205820018)

[The os module 9](#_Toc205820019)

[The exec function 9](#_Toc205820020)

[Threading 10](#_Toc205820021)

[Syntax highlighting 10](#_Toc205820022)

[HTML pages 10](#_Toc205820023)

[tkhtmlview 11](#_Toc205820024)

[Webview 11](#_Toc205820025)

[Conclusion 11](#_Toc205820026)

[Chapter 5. Design 12](#_Toc205820027)

[Use cases 12](#_Toc205820028)

[UI discussion 12](#_Toc205820029)

[Classes 12](#_Toc205820030)

[Files 12](#_Toc205820031)

[Setup wizard 12](#_Toc205820032)

[Backlash 13](#_Toc205820033)

[Wheel diameter 15](#_Toc205820034)

[Axle Length 16](#_Toc205820035)

[Chapter 6. Implementation 18](#_Toc205820036)

[Chapter 7. Testing 19](#_Toc205820037)

[How testing was approached 19](#_Toc205820038)

[Conclusions 19](#_Toc205820039)

[Chapter 8. Packaging? 20](#_Toc205820040)

[Chapter 9. Critical Evaluation 21](#_Toc205820041)

[Chapter 10. Conclusion 22](#_Toc205820042)

[References 23](#_Toc205820043)

[References 23](#_Toc205820044)

[Appendix A 24](#_Toc205820045)

[Appendix B- Manual testing table 25](#_Toc205820046)

List of Figures

[Figure 1- The project's GitHub Board 8](#_Toc205992602)

[Figure 2- Screenshot of turtle made with Java with its code 9](#_Toc205992603)

[Figure 3- Screenshot of turtle made with Python with its code 10](#_Toc205992604)

[Figure 4- Screenshot of input window for testing execution libraries 11](#_Toc205992605)

[Figure 5- Screenshot of code and window it produces for testing Python's threading 12](#_Toc205992606)

[Figure 6- Turtlebot application's use case diagram 15](#_Toc205992607)

[Figure 7- UI initial mock-up 16](#_Toc205992608)

[Figure 8-UI after combining spike work 17](#_Toc205992609)

[Figure 9-Final UI 18](#_Toc205992610)

[Figure 10-Spyder's layout 18](#_Toc205992611)

[Figure 11- Setup wizard flow chart 20](#_Toc205992612)

[Figure 12- Setup wizard flow chart: Setting the backlash 21](#_Toc205992613)

[Figure 13- Setup wizard flow chart: Wheel diameters 22](#_Toc205992614)

[Figure 14- Setup wizard flow chart: Axle length 23](#_Toc205992615)

[Figure 15- Extreme example of "spamming" the run button 25](#_Toc205992616)

[Figure 16-Horizontal original application logo 28](#_Toc205992617)

[Figure 17- Final logo design 29](#_Toc205992618)

[Figure 18-Original turtlebot bird’s eye view 29](#_Toc205992619)

[Figure 19-Picture with PowerPoint blocks overlaid 29](#_Toc205992620)

[Figure 20- Abstract turtlebot GIMP image 30](#_Toc205992621)

1. Introduction

## Background

Before the project began, a turtlebot had been created. This was done by designing it using OpenSCAD and then 3D printing the chassis. The turtlebot has four main features. Two wheels that are controlled y stepper motors to move the turtlebot, a screen display and a component where a pen can be put in which can then be moved vertically by a servo. The turtlebot’s firmware uses an Arduino and also has a Bluetooth component in order to receive and acknowledge commands from the PC.

Images would be nice here of scad

In addition to the turtlebot, some simple software had been made using Java to connect to the turtlebot and then send it commands. These commands included ones to control the pen servo to adjust the pen height and move the wheels. The movement commands have options to move in either millimetres or motor steps (with 4096 steps per rotation). Initially, the command options for moving forward, left, or right.

The intention for the turtlebot (and the ones produced later) was for them to be used for outreach and open day activities so that young people could have a go at controlling them and drawing shapes.

To this end, an application with a simple graphical user interface was desired so that users could type in their code and send it to the turtlebot. The application was also desired to have a virtual turtle, useful for quick testing or if turtles needed to be shared around. The creation of this application was the project.

* Maybe something more about why application?

## Project aims and objectives

The project aims to create an application that beginner programmers can use to develop their coding skills by controlling a turtlebot whether physically or virtually.

To do this the application needed to include a place for the users to input their code, somewhere for them to receive feedback for it (such as for errors) and a virtual turtle. For connecting to the turtlebot, the application needed a way to open and connect to the correct port and ideally show the user if they had managed to successfully connect, so they could then send the turtlebot their code.

Additional requirements for features were added as the project progressed to help the user have a better experience, including the ability to save and load the developed code, as well as an information page so users could find out more about the project and the commands they could use for the turtlebot.

## How the project was tackled

The project was created as a CustomTkinter application in the language Python. It was developed by breaking the project into feature chunks and tackling it in an agile manner. This was partially done to fit better with the project’s changing requirements as multiple were added during the projects lifetime.

First, the ability for the user to code and have a virtual turtle respond was tackled. From then, the turtlebot was introduced, so connecting the turtlebot and sending it commands became the focus. Once these two components were addressed, the project shifted its focus to additional features to enhance the user experience, which is where the additions of features came into play.

* Guide to subsequent chapters.

This paper blardy blar……. :

* Lit review to explore similar stuff
* Requirements
* Spike work
* Design
* Development
* Testing
* Packaging(?)
* Evaluation

1. Literature review

## Introduction

The use of robots to help people develop their coding skills has been a long-standing practice spanning over 50 years. A large quantity of these robots have had options to draw lines and shapes. Although the aims and base function of these systems have not changed drastically, the hardware and software have developed over time to provide a better user experience. Examples of this are the addition of graphical tools and hardware parts becoming smaller and cheaper. The first Turtles were developed as part of the development of LOGO [1]. Many of the more modern versions, software and hardware, have stemmed from this, such as Python’s Turtle Graphics [2][3].

This literature will explore the history of Logo and turtles, as well as other systems with the same aim.

## The early history of Logo and Turtles

### Logo

Logo was first invented in 1966 by Seymour Papert, Wally Feurzeig and Cynthia Solomon. In 1966, Papert visited multiple classes using programming languages and decided that a programming language designed specifically for children was required [4]. This language was then made and used in labs within the same year [2]. By 1967 the language was being tried out in schools.

Logo is a dialect of Lisp with some changes. One of the most important change was the introduction of commands[2]. Before this there were only procedures that could not return values. By introducing the option to return values, the options for programs greatly increased. Other smaller changes were made to make the language more child friendly. Lisp uses many parentheses, putting them around everything. Logo removed these.

Another choice taken to make Logo simpler is that it does not have an IDE. The idea behind this was so that the focus was to learn the language rather than trying to learn an IDE as well[2].

The first Logo programs were used for playing with words and sentences, aiming to help children to play around with them and learn grammar in the process [2]. One example of this was turning inputted words into pig Latin. These programs were originally run on time-shared computers (Digital Equipment Corporation PDP-1) [2].

The new language was seen to be successful, especially when the Logo environment ideology was kept to, which involved a culture of learning by doing [1]. The Logo group was officially formed at the MIT AI Lab in 1969[4].

### The introduction of Turtles

The idea of having physical objects to further develop the project was developed between 1969-70 [2]. Two of the objects developed were Turtles that could draw lines based on the commands they were given. The name turtle was inspired by the two automations that William Grey Walter built, which he called tortoises (Elise and Elmer more specifically)[2]. The original turtles did not have a graphical counterpart, which would come slightly later. The early turtles were connected to the computer via wiring to receive commands and send feedback[5].

The turtles had six commands: FORWARD, BACKWARD, RIGHT, LEFT, PENUP and PENDOWN. These could be combined within procedures to make a sequence of actions in hopes of creating shapes[1]. The layout of such a procedure to draw a triangle would be:

TO TRIANGLE

1. FORWARD 50
2. RIGHT 120
3. FORWARD 50
4. RIGHT 120
5. FORWARD 50
6. RIGHT 120

END

TRIANGLE

It was found that mistakes were more likely to lead to objective thinking as students tended to blame the turtle rather than themselves [1]. This meant that problems were dealt with constructively, focusing on why an issue occurred rather than the error itself.

### The introduction of Turtle geometry

Turtle geometry was introduced in 1970. This allowed a user to view a turtle and control it in the same manner as the Turtlebots as they worked off the same main six commands. Turtle Geometry worked with local geometry based upon the ideas of current position and heading [2]. It used standalone graphics terminals[2].

One example of its use was by Jon Pledge, a 12 year old at the time, who used it to create a universal maze-solving algorithm during an activity (now known as Pledge’s algorithm)[2].

After a conference in 1972 a company called General Turtle was formed in order to make and sell turtles[2]. Roughly three years later Marvin Minsky designed a standalone graphics system for the company [2].

### Logo since 1975

After the boom in personal computers, the development of Logo versions took up speed[2]. As of 2016 Logo has over 300 versions according to Pavel Boytchev’s Logo Tree Project[6]. Some of these are very similar to the original uses while others differed. Various programming languages implement as version of Logo by having their own version of Turtle graphics. The language was also used to create block-based learning environments such as Scratch[6][7]

## Later developments in Turtles and other robots

Similar to how there have been many versions of Logo, there have also been many different robots that be used similarly to turtles, most of which have developed from the original Logo ones. Not all of them have the ability to draw though many have it as an option, using modular sets.

The original turtles relied on a cord for communications. This was because radio links were considered to be too expensive given the turtles were being developed for use in Schools[5]. Since then non wired transmission mediums have become cheaper (and smaller) so are now the normal.

### BBC Buggy

The BBC Buggy launched in 1983[8]. This came in a set that users needed to assemble themselves with a range of programs. This set was modular so kits could be switched out depending on their intended purpose. One such kit was the pen kit that could be used with Logo to draw.

### The Valiant Turtle

Also released in 1983, the Valiant Turtle is controlled using LOGO to draw. It also had an on-screen version[9]. The turtle was controlled with infrared.

### LEGO Mindstorms

In 1987, LEGO started a collaboration with Logo to create turtles for them[2]. The first was the LEGO TC Logo. This required being wired to send commands through its ports. Other ports could be connected to sensors, lights and motors[10].

After this work started to put a processor into a large “brick” so that the bot could store and process its own commands. This led to the LEGO Mindstorms RCX in 1998. This concept was carried forward. Mindstorms are still sold today, though have gone through a few evolutions and now use block-based programming [11].

### TurtleBot

The TurtleBot is more for adult users than the original turtles. They are based upon Logo[12]. People can buy kits to make the robots, and this can lead to diverse applications, including home use and research[13].

## Conclusion

Many of the turtlebots and alternatives have had similar characteristics to the turtlebot for this project. The basic functions are naturally the same as pen controls, and movement requirements have not changed since 1969. The transmission of commands is similar to most other bots mentioned, with the exception of having a different medium, using Bluetooth rather than wired or infrared options. Like the original turtles, the turtlebots produced for the project are also made with the mindset of keeping production costs as cheap as possible.

The turtle graphics library that is used for the project is based upon the original one using the same concepts of position and heading even if the technology has advanced to where it no longer requires a separate graphics terminal.

The aim of turtles has not changed between the originals and this project. They both have the objective of getting beginner programmers to adopt coding by having a go.

1. Requirements

## Basic project requirements

The project started with some simple basic requirements:

*Basic Requirement (BR)1: A user should have an application where they can write code.*

This requires having the application with a writeable textbox. It also implies picking the code language either by creating one or picking an existing one.

*BR2: A user should be able to view a turtle simulation to see the output of their code.*

This requires picking a turtle library (or creating one, but reinventing the wheel is unnecessary). This simulation needs to be placed on the application screen and requires a way to receive the commands from the user-written code.

*BR3: A user should be able to connect to and send commands to the turtlebot.*

The application will require the ability to connect to the turtlebot’s port and send it messages. It will need to be able to receive acknowledgements as well to function well. This means that the code written by the user needs to either match the turtlebot commands or be translated into them before sending.

*BR4: A user can configure the pen height*

This is so the pen can be configured to be just on the page to reduce damage to the pen. The user will require a way to adjust the down value to test out different values.

*BR5: A user should be able to receive error feedback on their code*

The application will require a place to display error messages. This will show the user why their code does not work, so that it may be improved upon rather than leading to frustration from not knowing why.

## Additional project requirements

As the project progressed, more requirements were added beyond the basic functionality as it became apparent that there would be time to do them.

Additional requirements:

*Additional Requirement (AR)1: A user should be able to stop the code when running*

This is in case of the code being incorrect from what the user expected to happen or in the case of needing the stop the turtlebot (e.g. it has run off the paper). This will require a stop button that can stop both the simulation and the turtlebot.

*AR2: The user’s code input box should have syntax highlighting*

This is so the user can get debugging information before trying to run the code. For instance, if what is intended to be a string is not highlighted as such, it visually indicates that the code has an error.

*AR3:A user should be able to access information pages*

This is so that the user can access additional information about the turtlebot. They can also view the possible commands that they can type. For this purpose, the pages should be viewable from a separate window so that the user can go back and forth easily between viewing them and the main application to type the commands.

*AR4: A user should be able to change the font size of the code text and output text*

The application should allow the user to change the font size. This allows the application to be used and the code to be read on different screens, and additionally increasing application accessibility.

*AR5: A user should be able to save their current code and load their code from a file*

The application should allow the user to save their current code that they have been developing as well as load it back in. This means that the user will need to buttons to access these functions. The user should also be able to name their file when saving so they can recognise it when trying to load it later.

*AR6: A user should be able to set up the configurations of the turtlebot so that the drawings will be precise.*

The application requires a setup wizard-type process so the turtlebot can be configured with accurate values for its axle length, wheel diameter and motor backlashes. This is not to be done by the general user but will be required during the initial setup of the turtlebot and potentially later if drawing imprecision is found.

This will require a separate window guiding a user through drawing steps and what to measure. This separate window will need to be accessible through the main application.

*AR7: A user should be able to insert premade characters and shapes.*

This will allow users to quickly generate words and shapes. One benefit of this is that beginner users can view how to make a shape and then play around with it to match their purposes. This introduces a stepping stone into learning the system for those who are struggling to get started with the program.

To do this, the application will require a list of available shapes that the user can select from. The code to make the selected shape will then need to be retrieved or generated and then inserted where the user types their code.

All requirements can be found in Appendix B.

## Methodology

To meet these changing requirements, a fairly agile approach was taken using a GitHub board to track progress. The requirements were broken down into smaller chunks.

A screenshot of a computer

AI-generated content may be incorrect.

Figure - The project's GitHub Board

As seen in Figure 1, the board was loosely based upon scrum with the idea of picking items to do that week from the Project to-do list/backlog. Some requirements required more than one week, and some others went back into the to-do list once it became apparent that they needed redoing.

As the project progressed, the board was used less as the to-do list reduced, so tracking project progress became simpler, especially given its nature as an independent project. During later stages of the project, a simpler to-do list was kept, just crossing off items as they were done rather than using the whole board. Newer requirements were added to this list as required.

1. Spike work

## Picking software

The two obvious choices for writing the software were using either Java or Python.

### Java

Java had the benefit of already being proved to be able to communicate and send commands to the project.

Java also has a turtle library which can be used to create a virtual turtle and program it with it commands (as seen in Figure 2), and with an additional library such as JavaFX, an application could be produced using this.

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Screenshot of turtle made with Java with its code

One major downside of using Turtle with Java is that there is limited documentation for it with no central place with most needed information. This makes it harder to develop a more complicated system with it.

### Python

Python also has a turtle library which can be used in a similar way to the Java one (as seen in Figure 3). It also has formal Turtle graphics documentation[3] which can be easily accessed. Spike work was additionally done to check that the turtle could be embedded into a window which could have additional features added to it such as a button. CustomTkinter was used for this given previous familiarity with it.

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Screenshot of turtle made with Python with its code

By creating the application in Python then it allows the users to also write their code in Python. This achieves the application's aim of getting users coding with a proper programming language, whilst making development easier as no translation would be needed between languages.

Python is quite commonly taught in schools at secondary schools so a fair few potential users may already have some familiarity with it. It is popular since it resembles spoken English in many ways so can be easier to pick up. Given this it may prove easier for activities for outreach and open days.

The downside of Python would be that connecting to the turtlebot has not previously been worked out, but given Python’s popularity and large amount of documentation, there will be something out there to achieve this purpose.

### Conclusion

Python was chosen for the reasons listed above. The CustomTkinter library was chosen to build the GUI as a turtle could be easily embedded. CustomTkinter builds upon the Tkinter library which is one of Python’s basic GUI building libraries. CustomTkinter allows Tkinter to be used alongside newer elements such as themes and more modernised GUI features.

Mention Python’s compile ability(SEE BELOW FOR FURTHER DETAIL)

## Executing a user’s code

For a user to be able to create code and then run it within the application it is crucial to have a way for the code to be retrieved and run dynamically. Python has multiple libraries and ways to do this with their own benefits and drawbacks.

These were explored for their suitability by having a simple python file that would print hello world if executed correctly. Another file with the chosen method to read in the file and execute it was then created. These were tried both with the hello world file being correct and when it has errors.

Once the options had been explored this way, they were tried with a CustomTkinter window containing a textbox where “print(“Hello world”)” was written and a button that called a command to run the code that was written. This can be seen in Figure 4. The command was changed to use each library option.

A screenshot of a computer error

AI-generated content may be incorrect.

Figure - Screenshot of input window for testing execution libraries

### The subprocess module

The subprocess library has multiple functions that were tested out. Call, run and Popen were all tested out. These all did not output the expected result of “hello world”. By using the check\_output function and then printing out its results, it can be found to return "b'hello world\r\n'”. After investigation, the subprocess library seems to be more for running command line processes rather than Python files making it not suitable for this project.

### The os module

The os module offers the system function that can be used to read and execute files. It proved to be effective for the first text, printing “Hello world” to the console. Once CustomTkinter was introduced it stopped functioning.

### The exec function

The exec function is built into Python’s basic functions and executes the string of code it is given. To use this the python file must be opened and read then the contents passed to the exec function. All these functions are within Python’s standard library. This method worked successfully for both tests. For the first test, code errors in the hello world file caused the system to stop. This was not a problem once CustomTkinter was introduced as the application would keep running, although throwing errors when the code was wrong.

This method has the original benefit that it does not require the input to come from a file, as the exec function is looking for a string. This means that the code can be retrieved straight from the textbox and passed to it, skipping the middle steps.

The main issue with the exec function is that it will try to run any code it is given. This means it has security issues though since the use of the application is likely to be monitored it should be a minimal issue for this application.

The exec function shall be used as it works for desired application.

## Threading

Research was done into threading with Python, as the application would require multiple processes to be handled at once. Python is single threaded by default, and many libraries rely on being on the main thread, such as CustomTkinter. Self-made processes can be run on separate threads. This was attempted, utilising the turtle module to get two turtles to run in parallel to test using the threading library (which can be seen in Figure 5). This was successful, so it will be usable for more complicated parts of the final system.

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure - Screenshot of code and window it produces for testing Python's threading

## Syntax highlighting

For a better experience, coding it was decided that syntax highlighting would be useful for the box where users enter their code. This could be done for known keywords, such as for or strings, but would be better done with a library, as all the main keywords and syntax would already be known. There are a few libraries that serve this purpose, two of the most common ones for use with Tkinter are Idlelib and Pygments. Both libraries worked for their purpose when tried out, with the same result from the user's point of view.

Given that the application is intended to be kept simple and easily maintainable for developers, Pygments proved bulkier and required more lines of code to achieve the same purpose. Idlelib is more streamlined from a developer using its point of view and allows for easy choosing of colours for each kind of syntax highlighting that is wanted.

## HTML pages

The user will require some guidance about what code they can write and may wish to learn more about the software and hardware. To this end, some About pages would prove beneficial. To do this, it was decided to write the information in HTML for the application to then open and display on a separate window. This way, the pages could be updated without prior Python knowledge and without having to open the main code. A library that could provide this function was therefore required.

There are two main libraries that can open HTML pages inside Tkinter windows, both of which have their own benefits and drawbacks.

### tkhtmlview

The tkhtmlview library is specifically designed to work with Tkinter. HTML pages can be opened using it by putting files into objects such as its HTMLScrolledText, which can then be packed and displayed in a window. The main drawback of this library is that it doesn’t support the navigation tag. When used, it tries to open an unreachable webpage.

### Webview

Webview works by creating a new window using the HTML file. This works when it stands alone, and the navigation tag can be used. Problems arise when attempting to move between the About pages and the main window. Doing that causes the application to crash. Since the pages are designed to be referred to for command options whilst typing them, this is less than ideal. The Webview library only functions on the main thread, so putting it onto an alternative thread was not an option.

### Conclusion

The tkhtmlview library was chosen to be used since it mostly worked, and its drawback was minor in comparison.

1. Design

## Users

The application and turtlebot will need to be used by different types of users. These can be broken down into different groups, though there may be some overlap.

User types:

* Young and/or prospective students via open days or outreach activities who are new to coding
* Other students who have had more coding experience
* People demonstrating the turtlebot’s capabilities to draw more complex designs or leading activities using the turtlebot.
* The person who is doing the initial configuring of the turtlebot (using a setup tool)

The varying range of coding experience means that the application needs to be designed in a way that is beginner-friendly friendly with additional guidance available, whilst allowing those who are more familiar with the concept to get on with coding. This means that the application should allow the users to access more information rather than have it forced upon them, i.e. having them have to complete a step-by-step tutorial before starting. Help/ Information pages are useful for beginners, along with an example which can be provided as a default.

Given that some students will have had experience, it is better to have the application laid out in a way that is not too dissimilar to similar applications so that they can navigate easily without having to relearn aspects. It should also include some features that they would expect, such as syntax highlighting.

For those demonstrating the software, they may require having the screen display on different sizes, so the ability to alter the text size so it could be read across a room would be beneficial. It may also prove useful to be able to insert or load pre-made drawings/code to be able to demonstrate a design practised on a prior occasion.

The person configuring the turtlebot will need a way to step through the stages to do so. More information on this is available later in the chapter.

### Use cases

Based upon the requirements discussed within the requirements section and the types of users, certain use cases became apparent (see Figure 6).

UC1 Writing a program: When a user wants to write a program they have various options they can do within this.

UC1.1 Enter code: The user can click and type their code into a textbox

UC1.2 Run code: The user can click a run button to run their code on the simulation (and the turtlebot). The user can also choose to stop the program running whilst it is running in case of an issue or undesired behaviour.

UC1.3 Clear program: The user can choose to clear the code textbox and start with an empty one by clicking the clear program button

UC1.4 Reset turtle: The user can choose to reset the virtual turtle to its original starting position

UC2 Connecting the turtlebot: When a user wants to connect a turtlebot, they can pick an available port and the application will display if it is successfully connected with the turtlebot. The application will then send commands to the turtlebot as well when the user says to run the code.

UC3 Setup wizard: When a user wishes to configure the turtlebot they can open a setup wizard that takes them through steps to configure the wheels’ backlash, diameter and the axle length.

UC4 Adjust the pen height: The user can change the height of the turtlebot’s pen until they are happy by selecting different ones.

UC5 Change the font size: The user can open the menu and select a new font size to change the text size of the code text and output text.

UC6 Save code: The user can select to save their current code from the menu and name the file.

UC7 Load code: The user can select to load a file. They can then pick a file to load into the code textbox.

UC8 Insert shape: The user can select a shape or character to load into the code textbox via the menu

UC9 Reading the About page: The user can click on the about button on the top menu to load an additional window with more information about the project, for those curious, and with extra directions about coding and what commands are available to them.

A diagram of a diagram

AI-generated content may be incorrect.

Figure - Turtlebot application's use case diagram

## User Interface discussion

The user interface (UI) needed to be kept as simple as possible whilst still being attractive to passers by and young people. This meant intuitive buttons, colour all whilst not overcrowding the application. The less baseline features were planned to be kept near the top of the screen where they would be out of the way of main use.

To get an idea of the initial design, Microsoft PowerPoint was used to create some mock-ups based on the initial requirements; one of these mock-ups can be seen in Figure 7.

A screenshot of a computer

AI-generated content may be incorrect.

Figure - UI initial mock-up

From the beginning, it was clear that the user would require a large place to type their code, a place to view it in action via the simulation and various buttons to run the code, pause/stop it and to reset it. These buttons would later be moved to the right-hand side.

During the spike work process, an output box proved useful to give notifications when there were errors with the code. This was deemed an important requirement for the end UI, so it was incorporated into the design as can be seen in one of the early iterations of the design. Figure 8 shows a design with this new box incorporated.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -UI after combining spike work

Over time other features were added to the design to support the added features. This mainly consisted of what can be found on the top menu now.

The colour green was picked as the application's main colour. This was picked, given that turtles are green so it made sense.

The final design can be seen in Figure 9. This includes the applications icon which is based upon Python’s turtle library’s turtle.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Final UI

The final UI layout shares a lot of similarities with Spyder’s layout, which was used to create the code. This can be seen in Figure 10. There is a chance of unconscious bias towards this design; however, it does seem to work for Spyder. There are some design differences, for instance, the run button is in a different place, and Spyder does not have a virtual turtle.

A screenshot of a computer

AI-generated content may be incorrect.

Figure -Spyder's layout

## Classes

*CLASS DIAGRAMS AND DESCRIPTIONS HERE*

* *Contents/Purpose of each class*
* *How classes link/are used by others*
* *Any additional files they use*

### Classes for programming and operating the turtle

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

## Other Files

The project consists of eleven Python files. In addition to this, there are four other types of files used. Graphics (Portable Network Graphics and ICO) files for the graphics used within the application and for the application’s icon. Most of these file are found within the project’s graphics folder.

A HyperText Markup Language file is used for the information pages. This is so it can be edited and updated without the need to change the application’s code. It also has the benefit of the editor not needing to be familiar with Python. This file contains information about the turtlebot as well as guidance on how to use the application, including code command information. This file can be found within the project’s html\_info folder along with the images it uses.

The other file is a JSON file. This defines the application’s colour themes for CustomTkinter aspects. This theme is used by the application’s main class and defines the button colours and the top menu colour among other elements. The file can be found within the main src folder.

Text files are found within the characters folder’s subfolders. These contain code that can be inserted into the code editor of the application for quick design building.

## Setup wizard

The setup wizard is designed so that each turtlebot can be configured to the correct dimensions. These dimensions can vary slightly mainly due to shrinking as the plastic cools down once the parts are 3D printed. When the dimensions are slightly off it can lead to results such as circles and squares not matching up or lines drawn not being the expected length. Although this is minor was smaller drawings, when larger drawings are attempted this error is significant.

Why can’t just use a ruler?+ Explain what backlash is and why important (large impact for motor steps (high precision, low accuracy?))

To stop these problems a setup wizard is produced so that a user use it to work through various steps to calibrate the turtlebot to the correct dimensions and values. The dimensions in question are both wheels diameters and the distance between the two wheels (axle length). It is also important to measure the backlash of the motors so they can be factored into any lines to be drawn.

The setup wizard works in three stages: working out the backlash, working out the wheels’ diameter and working out the axle length (see Figure 11). After the dimensions are calibrated to a sufficient level they are saved to the EEPROM. After this, the turtlebot loads these values to use within the main program upon future startups.

A diagram of a process

AI-generated content may be incorrect.

Figure - Setup wizard flow chart

### Backlash

The first calibrations are done for the turtlebot’s backlash. This is done by having the turtlebot move backwards and then try to move it forward. If the turtlebot moves then it was instructed to move more than its backlash.

A diagram of a flowchart

AI-generated content may be incorrect.

Figure - Setup wizard flow chart: Setting the backlash

As seen in Figure 12, the turtle originally moves back by a large amount of steps. The increment starts as a fifth of the max amount. The turtle attempts to move forward by this amount.

If no forward movement occurs, then the turtle attempts to move forward by another increment amount. If movement does occur and a fifth of the increment is a whole number then the increment becomes a fifth of itself. The turtle then moves backwards by the last value it moved forward by and the moving forward process starts again.

If the increment becomes less than one, then the new backlash has been found and is saved. This is because the backlash is saved as the number of motor steps, which can only be a whole number.

### Wheel diameter

The wheel diameter can be worked out whether a line drawn is the correct length. If a straight line drawn is shorter than expected then the wheel diameters are smaller than expected. A longer line indicates that the diameters are bigger than expected. We assume that the diameters of both wheels are equal. This is likely as they are printed at the same time so should have been effected equally by factors. If this is not true, it also becomes apparent during the line drawing test, though it cannot be fixed using the setup wizard.

A diagram of a flowchart

AI-generated content may be incorrect.

Figure - Setup wizard flow chart: Wheel diameters

As seen in Figure 13, the user prompts the turtle to draw a line that should then measure 300mm. The user then uses a ruler to measure the actual length of the line and inputs this value. If this value matches the expected value, then the diameters are already correct, so the wizard can move to the next step. If this value does not match, then the new diameters are calculated by the fraction that the length was off by multiplied by the expected diameter. This new diameter is saved and the user can have the turtle draw the line again to see if the new diameter value is correct.

### Axle Length

Axle length becomes important when the turtlebot attempts turns. If the axle length is shorter than expected then the turtlebot will under turn and vice versa. This leads to shapes not connecting properly, either forming a gap, overlap or cross shape.

A diagram of a flowchart

AI-generated content may be incorrect.

Figure - Setup wizard flow chart: Axle length

As seen in Figure 14, a sequence of steps is undertaken to calibrate this. The turtlebot first draws two circles. Each circle uses one wheel as a pivot point while moving the other wheel to draw a circle. Each circle is calculated by working out how many wheel motor steps need to be taken to do a circle with the diameter of the axle. This is done by dividing the circumference of an axle diameter circle by the circumference of a wheel diameter circle and then multiplying this by the number of steps in a full wheel rotation. An example of the circles drawn can be seen in Figure \_\_\_\_.

*INSERT PICTURE OF AXLE DRAWING*

The user is then prompted to input two length values. If there is a gap in the circle the user inputs how much by as a negative. If there is an overlap the user puts in how much by as a positive number. If the circles are good then the user can input two zeros, in which case the setup wizard will move onto finishing. If there is a gap or overlap, the two inputted values are averaged. The expected axle value then has this averaged value divided by pi subtracted from it to get the new axle value. The user can then repeat the drawing process in hopes of seeing an improvement.

1. Implementation

## Introduction

Due to the nature of the project, many of the aspects were done in parallel. At the beginning of the project, there was no access to the turtlebot, so the code input and turtle simulation were the focus. As time progressed, a turtlebot was acquired. Once the basic functionality was achieved, additional features were added, along with the firmware being changed to support aspects such as the saving of turtlebot dimensions, which could then lead to a setup wizard being able to be created.

For the purpose of describing the implementation of the project, it has been broken up into blocks of functionality or of significance in implementing, with the addition of some key events.

## Basic functionality (pre-turtlebot)

After the spike work, multiple components existed that, if combined, would give a virtual turtle that could be moved via code that a user wrote. The components already made were a textbox with syntax highlighting, a virtual turtle simulation, a text box that showed error information and a run button. These were combined together resulting in three classes: a main class that ran the application, a turtle simulation class that handled the code and a code input class that created the textbox. After combining, the user could write their turtle code in, and when they pressed run, the simulation would run their code. The inputted code matched Python’s Turtle library.

A reset button for the turtle simulation and a clear all button for clearing the current program were then added. The application at this stage can be seen in Figure 8. The clear all button was later moved to be within the input box to make it clearer what it was clearing.

It was then found that some accidentally interesting results could be produced as the run button could be pressed repeatedly. This interrupted the current running code and replaced it with the new code again. An extreme example of this can be seen in Figure 15. To fix this, disabling the run button until the program had finished running was introduced.

A screenshot of a game

AI-generated content may be incorrect.

Figure - Extreme example of "spamming" the run button

## Responsive design

Responsive design was added to the project after the basic features were created so that the code boxes and simulation boxes always took up half the screen each, even as the window size changed. Additionally, code was added to alter the run and stop button layout when it can no longer fit horizontally to display them vertically.

## The challenge of stopping a running program

Being able to pause or stop the program lead to many of the design decisions of hw to have the code run. Originally the code was run by creating a thread for it and passing it into the exec() function within this thread. In Python there is no way to stop a thread. The closest you can get is to pause it for a given number of seconds. Given that a user does not want there program to unexpectedly start running again an alternative solution was required.

Python has a multiprocessing library which works in a similar manner to its threading library but can be stopped or paused. This library cannot be used successfully with Tkinter as it causes the application to stop working. No other libraries were successful for achieving the desired result so an alternative approach to handle the code was chosen.

### Taking a line by line approach

A new approach to handling the code was chosen. This was the process the code line by line to the turtle simulation so that it could have a conditional statement saying whether it should be paused. If paused, it would not do the action. This version also did not use threading.

Code was checked using Python’s compile() function to check for syntax errors. It was then converted into a list of lines of code. These lines were then passed to a exec one by one to be carried out and printed to the output box.. Within this an exception was added to check for name errors that the original compile could not catch. An additional step was later added to convert each line to turtlebot commands to put into another list using a dictionary.

Once additional language features were added in, this method reduced in benefit. To process more complex code features such as for loops this resulted in creating a half-made compiler that could unroll the for loops to put them into the list correctly. Additional tweaks had to be done so the code could handle lines with just whitespace.

Later it became apparent that variables would be useful so they were introduced by saving their names and values into a dictionary. Before the code was run the names were substituted for their values. A recursive function was then added to handle nested variables.

By choosing this method, the program could be stopped, however, it greatly reduced the code features that could be used as each one had to be added in separately which as a learning to code is counterproductive. A new approach was therefore required.

### Using an additional class approach

The code was changed so that it used a new class which knew how to handle each command. This version went back to putting the code into the exec function to run. The exec function handles to features of Python so the code does not need to do itself. The new class acts as a middleman to then process and translate as needed to send the commands to the classes that handle communicating with the turtle simulation and turtlebot. The functions within this class match up to the functions that a user can type to control the turtle.

This version was far more streamlined than the previous version, with the same benefits and all the normal Python features available, including being able to import modules so users can use features such as randomly generated numbers.

To stop the turtle, an else condition was added to the turtle simulation’s check if it was stopped. If triggered, an exception is raised, stopping the running program.

## Adding the turtlebot to the project (Connecting the bot+basic move+ U/D slider)

* Serial port: list and read (theoretically)
* Managed read to robot
* Turtlebot write throwing timeout on bot display.
* Got turtlebot running (had to wait till OKPC ack received, does by probing). Connected up to UI with dict to translate into turtlebot commands
* Refresh button. Show if usb and turtle are connected

## Moving the pen

* Added in pen u/d for turning (used 2D array). U/D seems to require a significant difference to actually move. Worked on pen up/down sliders. Problems with bot not always responding when message is received

## UI developments

* After main functionality achieved UI tweaks
* Started working on colour and suggested UI tweaks (Enter code->Enter Python Program (later scrapped)), introduce File|Settings, ability to change text size, default code
* (Sort of) fixed pop up icon overwrite issue (overwriting turtle with default ctk icon).
* Theme: #based on one taken from the CustomTkinter documentation https://customtkinter.tomschimansky.com/documentation/color then altered to fit needs
* A screenshot of a computer

  AI-generated content may be incorrect.

### Graphics

#### Creating the logo

All graphics were made using GIMP [ref?]. Microsoft PowerPoint was additionally used for producing the abstracted turtlebot image for the setup wizard.

The application’s logo was made using GIMP. It was originally made horizontally using Python’s Turtle as a rough visual guide for proportions. This was then turned into a blocker version, as seen in Figure 16.

A green puzzle piece with a white background

AI-generated content may be incorrect.

Figure -Horizontal original application logo

The logo was then rotated to be diagonal to fill the square space better. It was then also smoothed out further. There is the option to have a green edge or a white one, depending on use. The application uses the green one, and the user manual uses the white outline. The final GIMP file design can be seen in Figure 17.

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Final logo design

#### Creating the setup wizard graphics

Multiple graphics were created for this. The path drawings were made using GIMP’s pencil tool. The abstract turtlebot was a little more complicated.

A picture of the top of the turtlebot was taken, see Figure 18. Key shapes blocks where they picked and overlaid with PowerPoint shapes, see Figure 19. The shapes were then simplified further, and outlines were added with GIMP. The final result can be seen in Figure 20. This image was then added to other images to demonstrate how the turtlebot would move during the setup wizard.

A green robot with wires

AI-generated content may be incorrect.

Figure -Original turtlebot bird’s eye view

A green and blue object with black lines

AI-generated content may be incorrect.

Figure -Picture with PowerPoint blocks overlaid

A white and green camera

AI-generated content may be incorrect.

Figure - Abstract turtlebot GIMP image

## The creation of the top menu

### Menu buttons

* Looked into menu bars-issue was wanted to keep the showing of connection state on the same line which bar didn’t allow. Bar contains Menubuttons
* Looked into menubuttons and got a dropdown menu working. Got change text size working from that. Menubuttons added to own top frame already had so could keep layout.

### Font size issues

* Originally couldn’t do loop to make options based on list of sizes (always just did last size option
* Added i=I to be able to use a loop for font size picking without it defaulting to the last one lambda i=i: change\_textsize(i).

### Insert characters

* Added insert file at current position so now can add premade numbers.
* Get list of numbers etc dynamically so shapes etc can be added without changing code.
* Introduced loop for insert so can have all three types.

### Load/Read file

* Looked into loading a file with Tkinter file explorer and saving code (did have to work out had to write with "w" after brief struggle)

### HTML pages

* HTML pages for instructions. Sadly no nav in tkhtmlview, links go outside of window. Can't find a way to link within window aka for contents (works as HTML on own)
* Tried webview (pip install pywebview) instead and contents work, presumably because it treats it as a web page so more interactive. Causes app to crash. Must be run on main thread
* Scratch the contents thing. It would appear that once linked up to the main app, the choices are a non-working contents part or an application that crashes if you try to click on it when the html window is open. I'm going to prioritise the not crashing

### Port picking/swapping

* Might not need refresh anymore. Added wait a sec before checking for new ports when don't have one (2/07?).
* List ports and let user pick, Can't have dynamically change as not in Tkinter thread and says no (well "Runtime Error: main thread is not in main loop"). Could do as pop up dropdown instead without constant checking
* Done as pop up with refresh button for now
* 3 hour meeting(3/07?) trying to get swapping ports working. Turned out set\_port needed to call change\_port to actually reset the port
* Port Manager was later changed so that rather than defaulting to the first port, the user now needs to pick the port, which will then start a thread to read the input. When a new one is selected, it will close the old one, open the new one and start a thread to read from it.
* A label was also added to the top bar to show the last port selected
* Supervisor tried on mac /dev/ needed to be added to name to work so if not starting with COM it is added

## Open day tweaks

* A few people tried it out including one prospective student. It was also used as an opportunity. Particularly given the pre students attempts to use it, it seemed like having the project be able to handle variables would be a useful tool at least in a limited capacity. It would also support the projects usefulness as an educational tool
* The difference in sizes between txt box and console were more obvious on the B23 screen -> change to have same size and both change at same time
* Turtle easily runs off sim screen -> find way to add scrollbars
* Look into style for consistency ->json file
* changed to make menu buttons text green as the button turns white on mac
* Issue found as load was looking for "\*.txt\*" accidentally and mac can't handle "\*.txt"
* (for loop unrolling)
* Ctrl z added

## Scaling

* Objective: keep turtle on screen
* Turtle has no awareness of canvas edges. No level of turtle does and no functions to help. Surprisingly no people having similar issues on stack overflow
* Therefore we make up a method
* Since no edge awareness we scale turtle to fit
  1. Get commands so can work out final design size
  2. Get canvas size and approximate where turtle is on it also get padding
  3. For NESW keep track of scale required to stay on screen in that direction (start with 1) and amount moved in that direction
  4. Start facing East
  5. For each line:
     + If forward work out the horizontal and vertical components given the current angle. Add values to directions moved in
     + If curve (see curve write up) then do same as forward to collapse then add
     + If right or left add or subtract from current angle (%360)
  6. For each direction f turtle has moved in that direction work out scale required to fit
  7. Find smallest scale and if smaller than current scale change current to new
* Fixed scaling

## Setup wiz (maths and doing)

* Much maths to find shape options before going with supervisor equations anyway of percent off
* Maths! Can use arc to work out wheel diameter. Needs axel though and vice versa
* Worked out pentagon for working out axis, still no idea how to work out diameters if different
* Backlash basic UI that moves backwards then forwards in smaller and smaller increments.
* Start of diameter drawing (draw straight line)
* Axel basics
* Reworking backlash as didn't make sense. Axel work with simplified equation
* Formatting using place for standard look
* Fiddled with making images and mock-ups of wiz
* Added in images and tweaks to axle equation
* Corrections based on meeting backlash slow down + f not F. Steps are whole numbers so round() first
* A screenshot of a computer

  AI-generated content may be incorrect.
* Need round for wheel diameter value
* Round wheel diameters to 6DP as probable max double size.
* Change to axle equations (2nd one) as circle drawn is half axle not whole given pen position
* A white board with writing on it

  AI-generated content may be incorrect.
* Results in actual\_axle = instead of actual\_axle =. Given equation by supervisor incorrect
* Actually works and improves axle (circles can be made to match up rather than getting further away from correct)

## Curves

* New movement
* Added function to User\_Turtle class
* Work out maths to get radius from arc\_length and angle to pass to Python’s turtle.circle(r,a). Rest as normal
* Had to work out new scaling for curve to find extent it goes horizontally/vertically.
* Worked out can do per 90°.
* A white board with orange writing on it

  AI-generated content may be incorrect.

1. Testing

## How testing was approached

Testing was carried out via a manual testing table, which can be found in Appendix C. Manual testing was picked due to the heavy UI element of the application, as well as monitoring that the turtlebot did the correct action.

Unit testing was considered. However, there are very few occasions when they are suitable for use in the project, so testing was done solely using manual testing. Very few of the functions within the application return anything, and when they do, it is generally a UI element which wouldn’t be suitable for Python’s unit testing. Instead, it is more sensible to manually test by seeing if it shows on the application correctly.

## Conclusions

* Tests passed/Most passed apart from x,y and z, this is because…

1. Packaging?

* Packaging was done with auto2pyexe library to make into exe with a directory.
* Directory needed to locate files e.g. HTML. Means HTML can be changed on packaged version without need to repackage.
* Steps required to package: …

1. Critical Evaluation

The critical evaluation consists of a discussion, leading to conclusion. It is an essential part of a master’s degree.

It shows that you can not only carry out a substantial piece of work, but that you can reflect on it, and think critically about how you might have done it better.

Examiners view the critical evaluation as very important.

Critical evaluation should contain.

* Strengths and weaknesses of your project
* If you were unable to attain any deliverables, then why.
* What are the future plans for your project if you are to continue

You will be presenting this during demonstration but here you need to discuss them in detail.

* Achieved aims and more.
* Some elements could be better e.g. scaling turtle
* Usable and those who’ve tried it enjoyed
* Future -integration of svg stuff?
* Library choices and whether worked well
* Changes to bot for curves

1. Conclusion

A brief summary of all that has gone before, including the key results of the project.

May also include some directions for future work.

# References

|  |  |
| --- | --- |
| [1] | I. Jones, “New kinds of red ink,” *Inky Journal of Pigments,* vol. 336, no. 5, pp. 55-58, March 2010. |

# References

[1] S. Papert, “A COMPUTER LABORATORY FOR ELEMENTARY SCHOOLS,” MASSACHUSETTS INSTITUTE OF TECHNOLOGY A.I. LABORATORY, Oct. 1971. Accessed: Jul. 30, 2025. [Online]. Available: <http://hdl.handle.net/1721.1/5834>

[2] C. Solomon *et al.*, “History of Logo,” *Proceedings of the ACM on Programming Languages*, vol. 4, no. HOPL, pp. 1–66, Jun. 2020, doi: <https://doi.org/10.1145/3386329>.

[3] Python, “turtle — Turtle graphics — Python 3.7.4 documentation,” *Python.org*, 2019. <https://docs.python.org/3/library/turtle.html> (accessed Aug. 06, 2025).

[4] C. Solomon, “Logo Things,” logothings, 2016. https://logothings.github.io/logothings/Home.html#HOME (accessed Aug. 06, 2025).

[5] S. Papert and C. Solomon, “TWENTY THINGS TO DO WITH A COMPUTER,” pp. 1–5, Jun. 1971. Accessed: Aug. 06, 2025. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/5836/AIM-248.pdf?sequence=2&isAllowed=y

[6] P. Boytchev, “About,” Uni-sofia.bg, 2015. https://pavel.it.fmi.uni-sofia.bg/logotree/about.html (accessed Aug. 06, 2025).

[7] Scratch, “Scratch - Imagine, Program, Share,” *Scratch*, 2007. <https://scratch.mit.edu/> (accessed Aug. 07, 2025).

[8] N. Fazakerley, “BeebControl’s BBC Buggy Page,” Riscy.uk, 2025. http://www.riscy.uk/beebcontrol/buggies/bbc\_buggy/index.html (accessed Aug. 06, 2025).

[9] The Centre for Computing History “Valiant Turtle - Peripheral - Computing History,” *Computinghistory.org.uk*, 2025. https://www.computinghistory.org.uk/det/39597/Valiant-Turtle/ (accessed Aug. 07, 2025).

[10] “LEGO® TC logo.” Accessed: Aug. 06, 2025. [Online]. Available: https://ia802903.us.archive.org/4/items/lego-tc-logo-reference-guide/lego-tc-logo-reference-guide.pdf

[11] “LEGO® MINDSTORMS® About | Official LEGO® Shop GB,” *www.lego.com*. <https://www.lego.com/en-gb/themes/mindstorms/about> (accessed Aug. 07, 2025).

[12] Evan Ackerman, “TurtleBot Inventors Tell Us Everything About the Robot,” *IEEE Spectrum*, Mar. 26, 2013. https://spectrum.ieee.org/interview-turtlebot-inventors-tell-us-everything-about-the-robot

[13] TurtleBot, “TurtleBot,” *www.turtlebot.com*. <https://www.turtlebot.com/> (accessed Aug. 07, 2025).

Appendix A

Generative AI

1. No AI was used for the project.

Third Party Code and Software Libraries

1. .NET’s ASP.NET libraries have been used for this project. The document template created by the dotnet CLI tool produces a set structure for the Model-View-Controller parts of the project. The CLI tools were also used to generate the Entity Framework Core code in most Controllers. The CLI generated code was then adjusted to make it relevant to this application.

ADD USER MANUAL TO APPENDICIES

Appendix B- Project Requirements

## Basic Requirements (BR)

*BR1: A user should have an application where they can write code.*

*BR2: A user should be able to view a turtle simulation to see the output of their code.*

*BR3: A user should be able to connect to and send commands to the turtlebot.*

*BR4: A user can configure the pen height*

*BR5: A user should be able to receive error feedback on their code*

## Additional Requirements (AR)

*AR1: A user should be able to stop the code when running*

*AR2: The user’s code input box should have syntax highlighting*

*AR3:A user should be able to access information pages*

*AR4: A user should be able to change the font size of the code text and output text*

*AR5: A user should be able to save their current code and load their code from a file*

*AR6: A user should be able to set up the configurations of the turtlebot so that the drawings will be precise.*

*AR7: A user should be able to insert premade characters and shapes.*

Appendix C- Manual testing table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Ref | Test Content | Input | Output | Pass Criteria | Pass/ Fail |
| T-01 | Does the application start | Click on the Turtlebot.exe | The application opens with the default values and layout as shown: | No port is currently displayed in the top right, along with two red boxes. The default code of “turtle.down()  turtle.forward(20)  turtle.right(90)  turtle.up()  turtle.forward(20)” is in the textbox |  |
| T-02 | Can the default code be run on the simulation? | Click on run after launching the application | The right-hand side shows the turtle drawing and the box underneath the run button displays the commands in the input screen as they are carried out |  |  |
| T-03 | Can the virtual turtle be reset? | Click on the Reset button in the turtle box’s top right corner after state left by T-02 | The turtle returns to its default position, and the output text disappears | No more lines are visible |  |
| T-04 | Does a dialogue appear when trying to clear the program? | After the state left by T-02, click the “Clear Program” button in the top right of the code box. | A confirmation dialogue is shown asking “Are you sure you want to clear the program?” | The user is presented with two buttons on the dialogue, one for Yes and one for No |  |
| T-05 | Can clearing the program be done? | After T-04, click Yes | The dialogue box disappears, and the code box is now empty | The turtle is reset, and no drawings can be seen. The output box is also blank |  |
| T-06 | Can clearing the program be cancelled? | After T-04, click No | The dialogue box disappears. | No other change occurs |  |
| T-07 | Can a pop-up be opened to select a port? - no ports available | After launching the application:   1. Click settings. 2. Click Select port when no USB ports are available | A dropdown is shown with “No ports” and a Refresh ports button | Shows until Refresh ports is pressed |  |
| T-08 | Can refresh be pressed to see new available ports? | After T-07:   1. Insert the USB dongle. 2. Click Refresh ports | Select port is now shown on dropdown | Ports are available if click on dropdown |  |
| T-09 | Can a pop up be opened to select a port? - ports available | After launching the application:   1. Click settings. 2. Click Select port when USB ports are available | A pop-up containing dropdown is shown with “Select port” and a Refresh ports button | Ports are available if click on dropdown |  |
| T-10 | Can a port be selected? | After the state of either T-08 or 09:   1. Click the dropdown. 2. Click on COM3 | The popup disappears, COM3 is displayed in the white box on the top bar | The red box next to the word USB in the top bar turns green |  |
| T-11 | Can the application show that the USB connection has been stopped? | After T-10, unplug the USB dongle | The green box next to the word USB in the top bar turns red | The box next to Turtle also turns red |  |
| T-12 | Can the turtlebot be connected? | After T-10, press the button on the turtlebot on the right of the display | The box next to the word Turtle turns green from red. | The turtlebot displays the word “Hello” |  |
| T-13 | Can the turtle move the pen down? | Following the state after T-12:   1. Type “turtle. down()” in the code box 2. Press run. | The turtlebot moves its pen down and, “turtle.down()” is displayed in the output box. | No visual differences to the virtual turtle. |  |
| T-14 | Can the turtle move forward? | Following the state after T-13:   1. Type “turtle. forward(20)” 2. Press run | The virtual turtle moves forward and “turtle.forward(20)” is displayed in the output box. | The turtlebot moves forward 20mm. |  |
| T-15 | Can the turtle move forward? | Following the state after T-13:   1. Type “turtle. forward(55)”   Press run | The virtual turtle moves forward and “turtle.forward(55)” is displayed in the output box. | The turtlebot moves forward 55mm. |  |
| T-16 | Are spelling errors for the turtle caught? | Following the state after T-13:   1. Type “turte. forward(20)” 2. Press run. | The output box displays “name 'turte' is not defined” | Nothing else happens |  |
| T-17 | Are spelling errors for methods caught? | Following the state after T-13:   1. Type “turtle. foward(20)” 2. Press run | The output box displays “'User\_Turtle' object has no attribute 'foward'” | Nothing else happens |  |
| T-18 | Are syntax errors caught? | Following the state after T-13:   1. Type “turtle. forward(20” 2. Press run | The output box displays “'(' was never closed (<string>, line 2)” | Nothing else happens |  |
| T-19 | Can the turtle move backwards? | Following the state after T-13:   1. Type “turtle. forward(-20)” 2. Press run | The virtual turtle moves backwards, “turtle.forward(-20)” is displayed in the output box. | The turtlebot moves backwards 20mm. |  |
| T-20 | Can the turtle move the pen up? | Following the state after T-12:   1. Type “turtle.up()” 2. Press run | The turtlebot moves its pen up and, “turtle.up()” is displayed in the output box. | No visual differences to the virtual turtle. |  |
| T-21 | Can the turtle move 90° right? | Following the state after T-12:   1. Type “turtle.right(90)” 2. Press run | The turtlebot turns 90° to the right and, “turtle.right(90)” is displayed in the output box. | The virtual turtle also turns right so it is now facing south. |  |
| T-22 | Can the turtle move 180° right? | Following the state after T-12:   1. Type “turtle.right(180)” 2. Press run | The turtlebot turns 180° to the right and, “turtle.right(180)” is displayed in the output box. | The virtual turtle also turns right so it is now facing West. |  |
| T-23 | Can the turtle move 360° right? | Following the state after T-12:   1. Type “turtle.right(360)” 2. Press run | The turtlebot turns 360° to the right and, “turtle.right(360)” is displayed in the output box. | The virtual turtle also turns right so it is now facing East. |  |
| T-24 | Can a negative value be entered for moving right? | Following the state after T-12:   1. Type “turtle.right(-90)” 2. Press run | The turtlebot turns 90° to the left and, “turtle.right(-90)” is displayed in the output box. | The virtual turtle also turns left so it is now facing North. |  |
| T-25 | Can the turtle move 90° left? | Following the state after T-12:   1. Type “turtle.left(90)” 2. Press run | The turtlebot turns 90° to the left, and “turtle.left(90)” is displayed in the output box. | The virtual turtle also turns left so it is now facing north. |  |
|  | Can the turtle move 180° left? | Following the state after T-12:   1. Type “turtle.left(180)” 2. Press run | The turtlebot turns 180° to the left, and “turtle.left(180)” is displayed in the output box. | The virtual turtle also turns left so it is now facing West. |  |
|  | Can the turtle move 360° left? | Following the state after T-12:   1. Type “turtle.left(360)” 2. Press run | The turtlebot turns 360° to the left, and “turtle.left(360)” is displayed in the output box. | The virtual turtle also turns left so it is now facing East, its original position. |  |
|  | Can a negative value be entered for moving left? | Following the state after T-12:   1. Type “turtle.left(-90)” 2. Press run | The turtlebot turns 90° to the right, and “turtle.left(-90)” is displayed in the output box. | The virtual turtle also turns right, so it is now facing South. |  |
|  | Can the turtle move in using the curve function? | Following the state after T-12:   1. Type “turtle.curve(0, 0)” 2. Press run |  |  |  |
|  |  | (0,360) |  |  |  |
|  |  | (360,0) |  |  |  |
|  |  | (100,60) |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| T-22 | Can a variable be used for a forward value? | Following the state after T-12:   1. Type “x=50” 2. Add a new line. 3. Type “turtle.forward(x)” 4. Press run | The virtual turtle moves forward, “turtle.forward(50)” is displayed in the output box. | The turtlebot moves forward 50mm. |  |
| T-23 | Can a variable be used for a right value? | Following the state after T-12:   1. Type “x=180” 2. Add a new line. 3. Type “turtle.right(x)” 4. Press run | The turtlebot turns 180° to the right and, “turtle.right(180)” is displayed in the output box. | The virtual turtle also turns right so it is now facing west. |  |
| T-24 | Can a variable be used for a left value? | Following the state after T-12:   1. Type “x=180” 2. Add a new line. 3. Type “turtle.left(x)” 4. Press run | The turtlebot turns 180° to the left and, “turtle.left(180)” is displayed in the output box. | The virtual turtle also turns left so it is now facing west. |  |
| T-X | Can for loops be used? | Following the state after T-12:   1. Type “for i in range(4):” 2. Add a new line. 3. Add a tab. 4. Type “turtle.forward(20)” 5. Add a new line. 6. Add a tab. 7. Type “turtle.right(90)” 8. Press run | The turtlebot draws a square of dimensions of 20mm by 20mm. The output box displays “turtle.down()  turtle.forward(20)  turtle.right(90)  turtle.forward(20)  turtle.right(90)  turtle.forward(20)  turtle.right(90)  turtle.forward(20)  turtle.right(90)” | The virtual turtle draws a square: |  |
| T-26 | Are indent errors caught? | Following the state after T-12:   1. Type “for i in range(4):” 2. Add a new line. 3. Type “turtle.forward(20)” 4. Add a new line. 5. Type “turtle.right(90)” 6. Press run | The output box displays “expected an indented block after 'for' statement on line 2 (<string>, line 3)” | Nothing else happens |  |
| T-27 | Can the turtle be stopped? | Straight after T-X press stop | The turtlebot stops after the current command. The output box says stop safter the current command. | The virtual turtle stops after the current command. Nothing else is added to the output box. The run button is available to click again. |  |
| T-Y | Can a save as dialog be opened | Following the state after T-X:   1. Click on File on the top bar 2. Click Save |  |  |  |
| T-29 | Can the current code be saved? | Following the state after T-Y:   1. Enter “my\_code” as the filename 2. Click Save |  |  |  |
| T-30 | Can saving the file be cancelled | Following the state after T-Y:   1. Click Cancel |  |  |  |
| T-31 | Can code be loaded? |  |  |  |  |
| T-32 | Can loading code from a file be cancelled? |  |  |  |  |
| T-33 |  |  |  |  |  |
| T-34 |  |  |  |  |  |
| T-36 |  |  |  |  |  |
| T-37 | Insert number |  |  |  |  |
| T-38 | Change font size |  |  |  |  |
| T-39 |  |  |  |  |  |
| T-40 |  |  |  |  |  |
| T-43 | Pen height |  |  |  |  |
| T-44 | Pen height up |  |  |  |  |
| T-45 | Pen height- down |  |  |  |  |
| T-46 | Setup wizard- no port |  |  |  |  |
| T-47 | Setup wizard port |  |  |  |  |
| T-48 | Setup wizard- backlash |  |  |  |  |
| T-49 | Setup wizard- backlash |  |  |  |  |
| T-50 | Setup wizard- backlash |  |  |  |  |
| T-51 | Setup wizard- diameter |  |  |  |  |
| T-52 | Setup wizard- short |  |  |  |  |
| T-53 | Setup wizard- long |  |  |  |  |
| T-54 | Setup wizard- correct |  |  |  |  |
| T-55 | Setup wizard- diameter |  |  |  |  |
| T-56 | Setup wizard- short |  |  |  |  |
| T-57 | Setup wizard- long |  |  |  |  |
| T-58 | Setup wizard- correct |  |  |  |  |
| T-59 | About | After T-01, click on the About button on the top bar | A window opens with additional information which matches the layout defined in the HTML file. | The main window can be used (clicked on and typed on) whilst the About window is open |  |