A Graphical Programming Language Editor

Candidate No. 198719

November 3, 2020

Contents

1 Introduction

Graphical programming can be a fantastic and intuitive way to introduce new programmers to the scene. When programmers ask others for assistance, those helping typically do so in a visual style, using whiteboards, drawing flowcharts, with boxes and arrows indicating the flow of the program. Why can't we make programs in the same style if we find it so helpful to read? The concept behind graphical programming is specifying the elements of the program graphically rather than textually [3].

Popular examples of graphical programming include Scratch, as well as a personal favourite that I used during my GCSE Computing education, App Inventor! What's clever about Scratch is its simplicity due to the block-based visual programming, aimed towards younger children to help them get into coding!

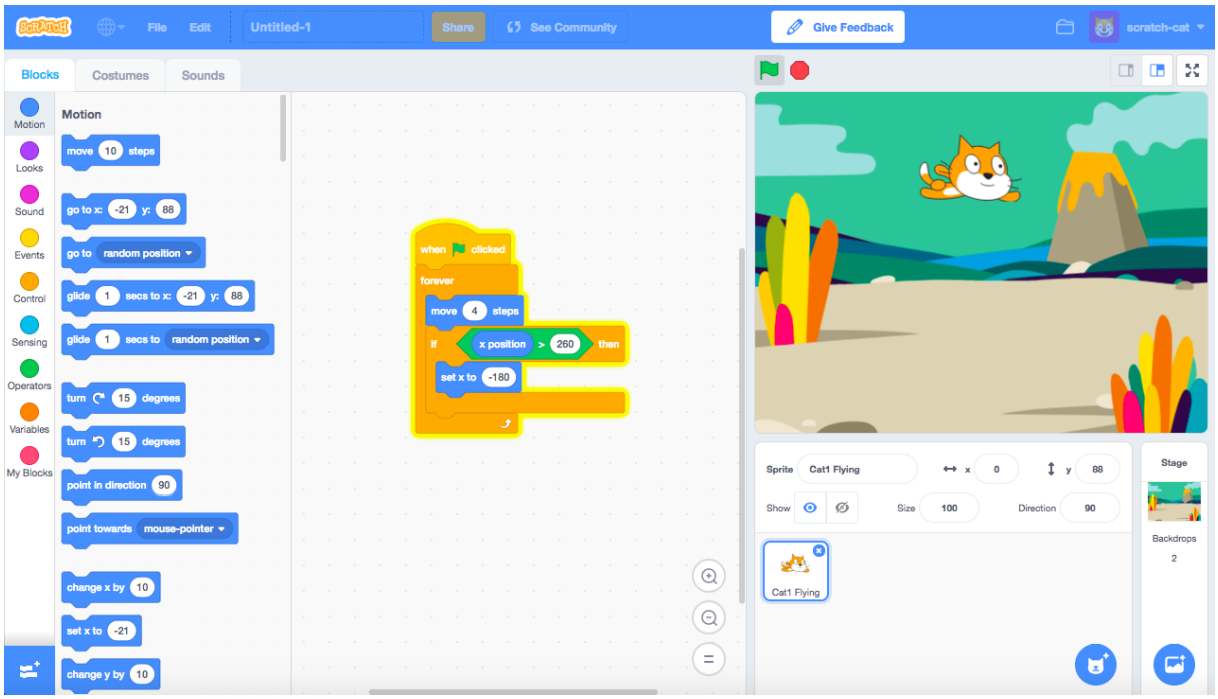

Figure 1: Scratch's visual scripting over a white canvas [9].

MIT App Inventor is one of my personal favourites after experiencing it myself during GCSEs, having to building an on-campus application for a University! It uses a graphical user interface, similar to Scratch, as well as providing the blocks that you can use, visible on the left hand side.

Figure 2: MIT App Inventor graphical programming [1].

Both what Scratch and App Inventor do superbly is colour code their blocks to identify exactly what they are representing. There are colour coded blocks for variables, events and controls, and so on! For instance in App Inventor, variables are coloured in Orange blocks, and procedures and show in a big purple block, encapsulating all the blocks like a function.

1.1 Aims

The aim of this project is to design and implement a simple interface for the user to graphically program basic functions. Simple functions may include:

- loops.
  - this includes `for` and `while` loops.

- conditional statements, such as `if` statements.
- implementing and interacting with data structures, such as lists.
- procedures like methods/functions.
- variables!

This is necessary as it provides another programming viewpoint for the user, which can aid them understand the process and flow of how said program may be running. It is also friendly and less intimidating for newer users, relative to a textual/command line script.

I am going to program this is Java, my most experienced language, making use of Java's smart GUI design feature, whilst learning and building knowledge on said feature for my own experience.

1.2 Objectives

The main objectives are:

1.2.1 To investigate system requirements to produce a requirements specification.

1.2.2 To select and justify an appropiate simple design for the project.

1.2.3 Implement a simple interface for users which is easy to understand.

1.2.4 Functional and working loops, conditionals and data structures that can be interacted with.

1.3 Problem Area

The main challenge of implementing an interface to be able to program on is the required time and skill set. This project may never be fully complete as more work can always be done to improve the functionality of the interface, or making the interface more user-friendly.

The project can be broken down into key areas that require focusing on:

- Graphical Design - modelling, interface, simplicity, GUI.
- Functionality - programming.

I have limited GUI design in Java from my 2nd year module, Further Programming, therefore this process can be said to take up a large amount of time. Careful consideration is needed to not overspend my time on the GUI design process, and rather get the functionality of said interface to work.

Functionality requires programming to be clean and efficient, so not to go back over the code and forget what was written. Code must be documented and commented along

the way to make sure anyone who reads the code, including myself in the future, is able to understand what was written and why it was decided.

1.4 Expected Outcomes

The expected outcomes of this project are a fully working, functional graphical programming language interface. The user should use the tool independent from the back-end Java textual code, and be able to perform the certain tasks:

- loops - create and work with basic for & while loops.
- data structures - create and interact with data structures such as lists.
- variables - create, store data and interact with variables.
- procedures - create functions/methods to perform certain tasks inside them!
- interface - an interface for users to be able to interact with and perform the above outcomes.

2 Professional and Ethical Considerations

Below you will find the relevant sections of the BCS Code Conduct that apply to this project. Since this project will not require any human participation, no ethical review is necessary.

2.1 BCS Code of Conduct

1.1 - have due regard for public health, privacy, security and wellbeing of others and the environment;

This project does not feature any distressing visuals that may harm the user.

2.1 - only undertake to do work or provide a service that is within your professional competence;

This project is within my professional skillset to complete.

2.3 - develop your professional knowledge, skills and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards that are relevant to your field;

Throughout this project I will maintain my skills, but further learn and research new ideas or more experienced methods to make my skills more proficient within the field.

2.5 - respect and value alternative viewpoints and seek, accept and offer honest criticisms of work;

I will accept feedback by my supervisor and others and use the criticisms to develop and build upon the project further. Results of feedback will not be edited.

3.2 - seek to avoid any situation that may give rise to a conflict of interest between you and your relevant authority;

Throughout the project I will be respect to my colleagues and my supervisor.

3 Related Work

3.1 Scratch

The key goal that Scratch sticks by is trying to introduce programming to those with no programming experience knowledge whatsoever. Scratch has been widely distributed to school systems and education organisations [6]. Scratch also couldn't be any more clearer to use, with a heading of categories for you to choose and select the difference blocks that provide different functionality.

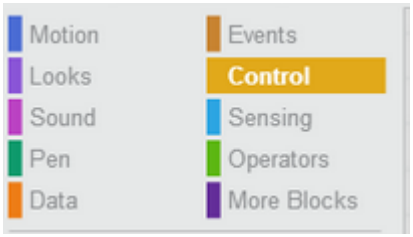

Figure 3: Scratch's categories of blocks that you can select from.

The user doesn't require any documentation to understand how to use Scratch, rather they are able to learn on the go, hence why it is suitable to new programmers trying to get a basic feel of the approach and what programming can do, as well as what you are able to do with it!

3.2 MIT App Inventor

MIT App Inventor is another graphical programming language for building apps for Android devices. App Inventor is used across 195 countries, in education organisations and also self-taught [11].

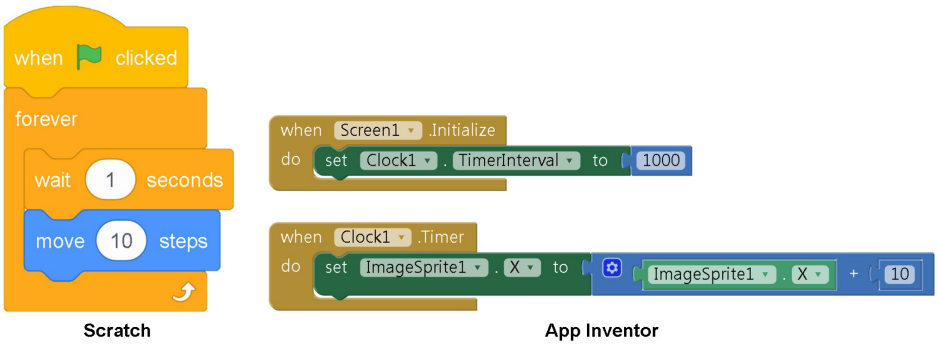

Figure 4: Graphical style of Scratch vs App Inventor [7]. App Inventor is slightly more sophisticated.

Scratch and MIT App Inventor are the two most widely used block-based graphical programming languages for students and/or pupils. As of August 2019, there were 44,981,198 registered users on Scratch, and 8,200,000 registered users on App Inventor [7].

Both Scratch and App Inventor share the common goal of providing an educational programming language for new comers. Scratch is mainly used as a foundation to teach younger children the concepts of code, whereas App Inventor is used as the *"step up"*, but both are used globally to make programming a bit more fun and interactive. App Inventor in particular is rewarding as upon running the code, an interactive mobile phone displaying your app is shown as the output!

3.3 Blockly

Blockly is probably by far the most modern out of the three I have proposed, and the most intuitive. It's a free, web-based, open-source project by Google, built upon the idea of Scratch. But what's intuitive about Blockly is the textual programming output on the right-hand side of your graphical code. It can generate code in the following languages [5]:

- Javascript.
- Lua.
- Dart.
- Python.
- PHP.

It can also be edited to generate code in a different textual programming language!

Blockly is aimed towards new programmers to get them onboard and addicted to the visual scene first. By providing the textual version of the code, perhaps this might spark a passion within the user to follow through with programming. By being provided with the textual code of their graphical app, they are learning and understanding what the code will really look like and how they could potentially implement it themselves.

A key part of Blockly that the developers undertook was user testing. The conditional and loop blocks were in the same category, with the same colour, confusing the users who expected the program to run a certain way, when it ran differently to what they were expecting [4]. The developers understood this and learnt from the feedback quickly, moving the conditionals to a different category, as well as changing the colour too. This removed the confusion instantly. These key ways to improve from feedback received is key to building a successful project.

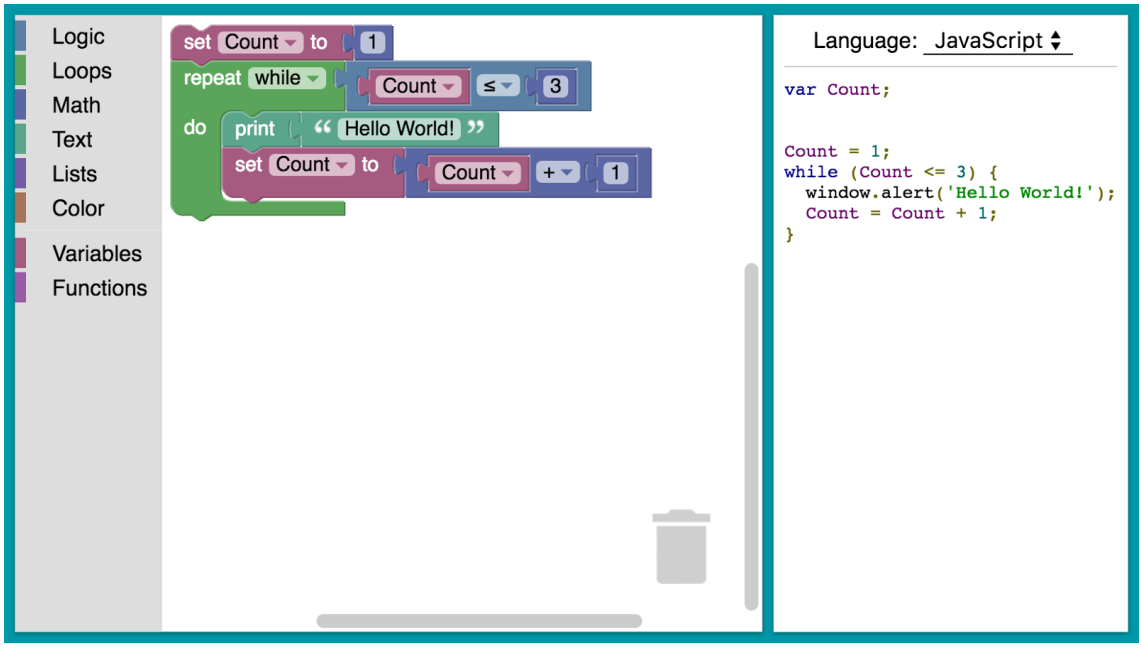

Figure 5: Blockly's modern and simple block programming design, as well as the option to choose the programming language that it will syntactically output as [10].

3.4 Graphical Programming

What all three graphical programming editors do together is follow the convention of lowercase naming of variables, functions and so on that is followed in textual-based programming languages, for instance *"if"* and *"while"*.

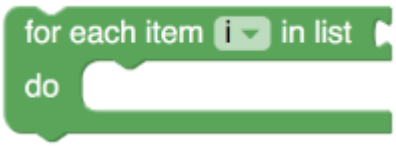

Figure 6: The lowercase convention, as followed by Blockly as an example [8].

Graphical programming is not a permanent solution to programming, but rather an introduction [2].

4 Requirements Analysis

In this section, I have taken into careful consideration the aims and objectives for this project. Below you will find a compiled list of software requirements, split into two categories: **mandatory** requirements, the fundamental requirements that is needed for the project to work, and **desirable** requirements, optional further requirements that can be met to provide further quality of life experiences for the user.

4.1 Mandatory Requirements

Mandatory Requirements	
Requirement	**Specification**
1. Application will work on any java-run environment.	Whether the user is on a Windows, macOS or Linux system, the application will work.
2. The interface shall accept user input via keyboard and mouse.	User will be able to use keyboard and mouse to create and interact with the canvas.
3. The program shall run until is it killed.	The process will not stop running until either it is killed via user or forced shutdown.
4. The program will output any errors, if any.	If there are any run-time errors, the terminal will output those for the user to see.
5. The user shall be able to use functional loops such as for and while loops.	The user will have the ability to interact with loops in the interface.
6. The user shall be able to create variables.	The user will be able to create and interact with variables.
7. The user shall be able to create functions or procedures.	Users will have the ability to create functions /methods that perform a variety of tasks.
8. The user will be able to create data structures.	Data structures such as lists will be available for the user to implement and interact with.
9. A run button.	A button to run to be able to run their program.

4.2 Desirable Requirements

Desirable Requirements	
Requirement	**Specification**
1. Colour coded blocks.	The designs can be colour coded to visually aid the user what they represent. For instance, orange blocks for variables.
2. Interactive sound design.	A potential sound that can be producing by interacting with the program. For instance, dragging and dropping, clicking, etc...
3. Highlight when mouse over	When the mouse is hovered over an object, said object could be highlighted by a glowing fade, or perhaps an outline of the object.
4. Recycling bin.	A recycling bin in the bottom corner to show users where they can drag and drop blocks that are not needed. This could only be visible when an object is being dragged, or it could always be visible.
5. Undo button.	A button to undo an action.
6. Redo button.	A button to redo an action if undone.

My aim is to attempt to achieve all the mandatory requirements as they are underlying foundations of the program. All the requirements work together to build the program. Without any of these requirements, this will fail to work effectively. The desirable requirements that I would like to be able to implement include the undo + redo button, as well as the recycling bin! Thinking about the user's goals here, these would be incredibly impactful towards benefiting their quality of life experience. Having the ability to undo/redo any mistakes you make, as well as simply drag and drop to the bin to delete unnecessary content, only makes the lives of the user easier.

5 Project Plan

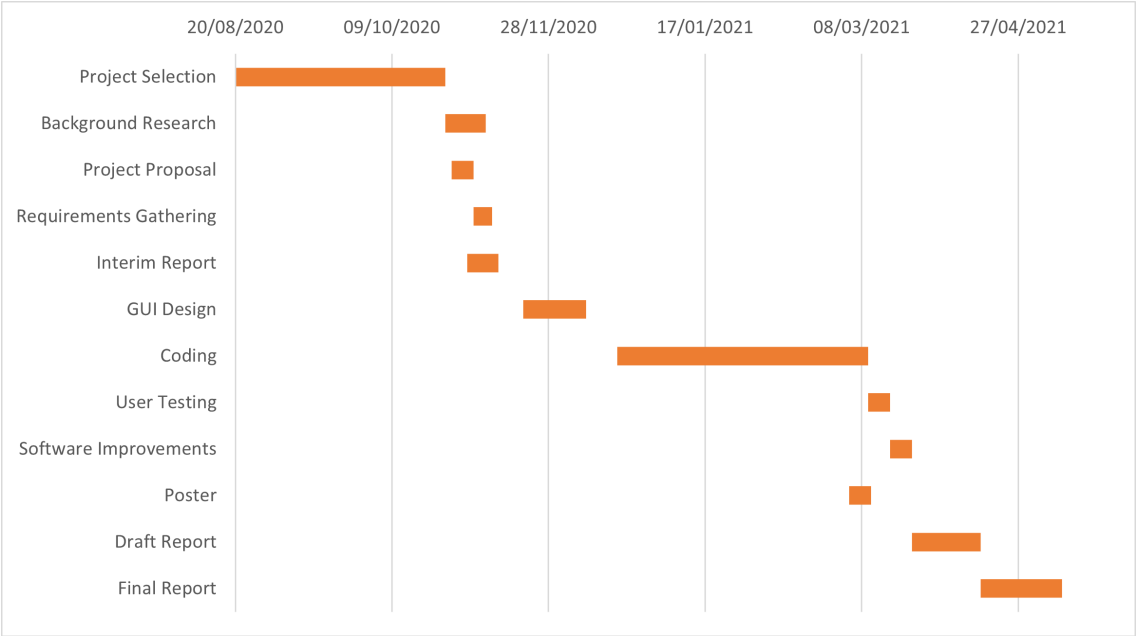

Figure 7: My gantt chart displaying the process I intend to follow throughout the project.

5.1 Completed Work

As of the time of this report being handed in, all the work I have completed thus far are the following:

- Project proposal.
- Background research.
- Requirements gathering.
- Interim report.

5.2 What's to come

After completing this, I plan to have a meeting with my supervisor soon to discuss design ideas for the project, and what the GUI might potentially look like.

Of course, throughout the project my draft report will be the main task. Although my gantt chart shows a period in March-April where I will be hard focusing on it, throughout the project, I do plan to add bits here and there to it, to get started.

5.3 Other Tools

I have utilised a private Github repository for version control.

6 Interim Log

Log of the meetings with your supervisor.
Should cover discussions of every important project stage.
Record the prupose and the outcome of every meetings.
Note how these meetings helped you follow your plan.
Include this log as an appendix to your report.

7 Appendices

7.1 Project Proposal

References

[1] Richard Albritton. "What is App Inventor?" In: *https://learn.adafruit.com/mit-app-inventor-and-particle-io* (2018).

[2] Gregory Barkans. "Visual Programming and Version Control: Future Considerations". In: *https://medium.com/vapurrmaid/visual-programming-version-control-future-considerations-b18c9a49c187* (2018).

[3] REMI Dehouck. "The maturity of visual programming". In: *http://www.craft.ai/blog/the-maturity-of-visualprogramming* (2015).

[4] Neil Fraser. "Ten things we've learned from Blockly". In: *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. IEEE. 2015, pp. 49–50.

[5] "Google's Blockly Teaches You to Create Apps". In: *https://www.nbcbayarea.com/news/national-international/googles-blockly-teaches-you-to-create-apps/1918242/* (2012).

[6] John Maloney et al. "The scratch programming language and environment". In: *ACM Transactions on Computing Education (TOCE)* 10.4 (2010), pp. 1–15.

[7] Youngki Park and Youhyun Shin. "Comparing the Effectiveness of Scratch and App Inventor with Regard to Learning Computational Thinking Concepts". In: *Electronics* 8.11 (2019), p. 1269.

[8] Erik Pasternak, Rachel Fenichel, and Andrew N Marshall. "Tips for creating a block language with blockly". In: *2017 IEEE Blocks and Beyond Workshop (B&B)*. IEEE. 2017, pp. 21–24.

[9] The Scratch Team. "3 Things To Know About Scratch 3.0". In: *https://www.medium.com/scratcht blog/3-things-to-know-about-scratch-3-0-18ee2f564278* (2018).

[10] "Try Blockly". In: *https://developers.google.com/blockly* (2020).

[11] Benjamin Xie and Hal Abelson. "Skill progression in MIT app inventor". In: *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. 2016, pp. 213–217.