

# A Graphical Programming Language Editor

Candidate No. 198719

Interim Report

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Aims . . . . .	2
1.2	Objectives . . . . .	3
1.2.1	To investigate software requirements to produce a requirements specification. . . . .	3
1.2.2	To select and justify an appropriate simple design for the project. . .	3
1.2.3	Implement a simple interface for users which is easy to understand.	4
1.2.4	Functional and working loops, conditionals and variables that can be interacted with. . . . .	4
1.3	Problem Area . . . . .	4
1.4	Expected Outcomes . . . . .	5
1.5	Relevance . . . . .	5
<b>2</b>	<b>Professional and Ethical Considerations</b>	<b>6</b>
2.1	BCS Code of Conduct . . . . .	6
<b>3</b>	<b>Related Work</b>	<b>7</b>
3.1	Scratch . . . . .	7
3.2	MIT App Inventor . . . . .	7
3.3	Blockly . . . . .	8
3.4	Graphical Programming . . . . .	10
<b>4</b>	<b>Requirements Analysis</b>	<b>11</b>
4.1	Mandatory Requirements . . . . .	11
4.2	Desirable Requirements . . . . .	12
<b>5</b>	<b>Project Plan</b>	<b>13</b>
5.1	Completed Work . . . . .	13
5.2	What's to come . . . . .	13
5.3	Other Tools . . . . .	14
<b>6</b>	<b>Main Report</b>	<b>15</b>
6.1	Designing a rough GUI . . . . .	15
6.2	Copy Constructor . . . . .	18
6.3	DragLayout & ComponentMover . . . . .	19
6.4	Deleting blocks . . . . .	21
6.5	Maintaining order of the program . . . . .	24
6.6	Interpreter . . . . .	27

<b>7</b>	<b>Appendix</b>	<b>28</b>
7.1	Interim Log . . . . .	28
7.2	Project Log . . . . .	29
7.3	Project Proposal . . . . .	30

# 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 [6].

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!

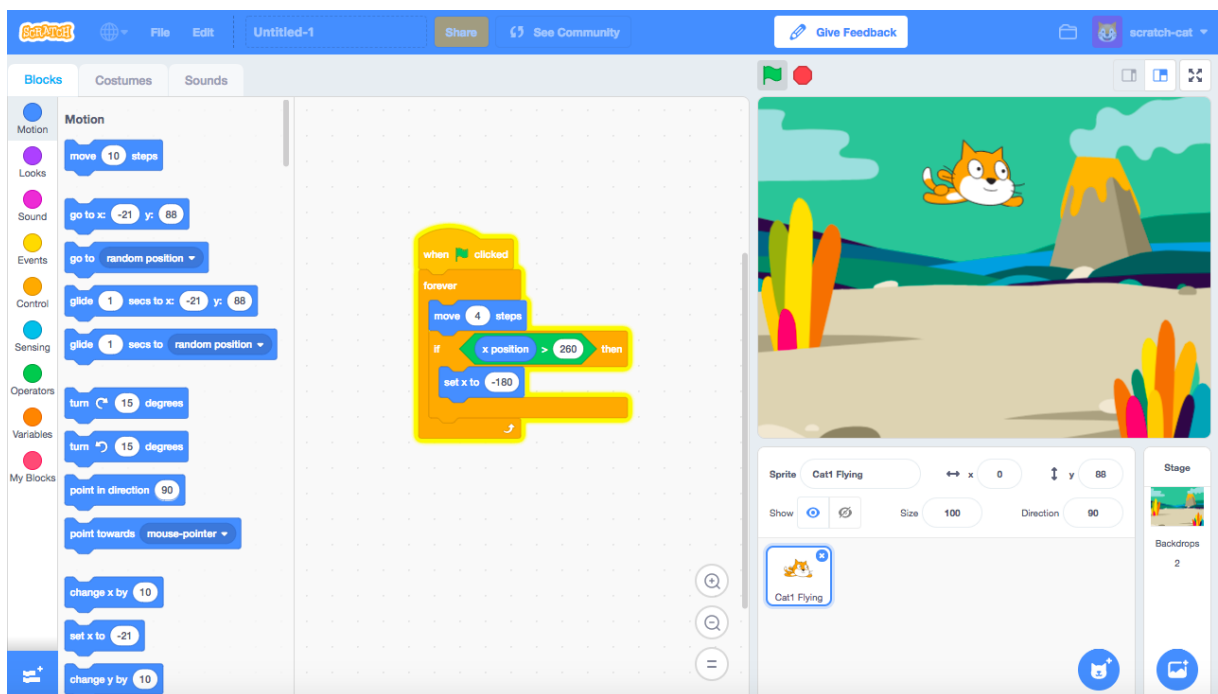


Figure 1: Scratch's visual scripting over a white canvas [13].

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 in 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 software requirements to produce a requirements specification.

It is important to clarify what the software requirements are for this project as they define whether this project will work or not. Without the mandatory requirements, the game will not be properly functional. I investigated what requirements are mandatory by doing research on existing softwares such as App Inventor and Blockly. For instance, all softwares I found had a **run** button in order to be able to run the graphical entities. Without this, how else is the canvas run? Of course it is not a big deal, and there would probably be another solution, but it's a simple feature that also provides simple UI for the user, which is self explanatory.

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

This objective is to research other designs of similar projects and to understand why their interface works well from a new user's perspective, but also why it doesn't work very well. My knowledge in Human-Computer Interaction will help me to focus on the design, and what defines a *"user-friendly"* design. The goal here isn't implementing the design, it's creating a design, on pen paper, attempting to make it as self-explanatory as possible. But that also begs the question, *"what determines a self-explanatory design?"* It's important to realise that everyone's judgement of what a simple design looks like is different. It's a subjective topic but I must ensure I use the principles learnt from HCI about simple design and user-friendly interaction to make a design at the best of my ability.

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

An important objective, which builds off of 1.2.2. I have limited GUI experience, where the first and last GUI building I engaged with was in 2nd year, Further Programming module, with Dr. Ian Wakeman. I spent a lot of time during that assignment at the time focusing on a simple interface for the Futoshiki puzzle.

This objective will benefit me greatly. Through research and problems, I will get through this objective and implement an interface, whether it be outstanding, or just about finished, and I will gain a lot of knowledge and experience on the way.

### **1.2.4 Functional and working loops, conditionals and variables that can be interacted with.**

What's the point of having a pretty canvas if you can't do anything with it? This is the meat of the project, what's going to make the project be useful. The main objective here is this one. My main priority is the ability to get this project off its feet.

Variables for user's to be able to create and store data with, as well as interact with within the project. Loops to allow the user to work with logic. Without this objective being met, it's just a pretty GUI with no functionality.

## **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.
- 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.

What do we expect it to behave/look like? A blank canvas should appear upon starting, with a heading on the left hand side, presenting the user with a selection of choice such as "*Loops, Procedures*" and so on, for each category of block. They will be able to click on said heading, which will expand and show the different blocks they can choose from. So if they chose the *loops* heading (a draft name for now, I know), this will expand and show for & while loops for them to choose from.

An important point to address, is what will happen when the user runs the program? This is a key concept which will be an ongoing investigation throughout the project. But the current idea is simple: an output screen showing the terminal. Upon running, a heading on the right hand side, or perhaps underneath, will pop out with the terminal showing you the output of your program.

## 1.5 Relevance

This project involves modules across my degree, as well as Java; the most extensively used language across my university education thus far. This project will use skills learnt in Human Computer Interaction; test my ability to understand the user's perspective and struggles, involving the evaluation of the software, rather than my own when designing the interface for them to use.

This project also incorporates key concepts understood from Software Engineering. The ability to criticise my own work, collect requirements, manage time and manage an agile approach to the project. Therefore, this project will test my ability to make use of several skills in such a way that they will be used outside of my university education.



## 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 [9]. Scratch also couldn't be any more clearer to use, with a heading of categories for you to choose and select the different blocks that provide different functionality.

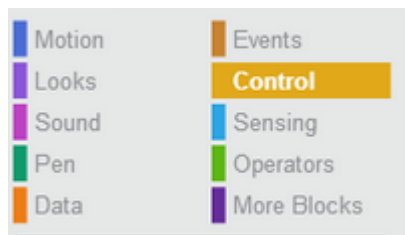


Figure 3: Scratch's categories of blocks that you can select from.

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 [16].

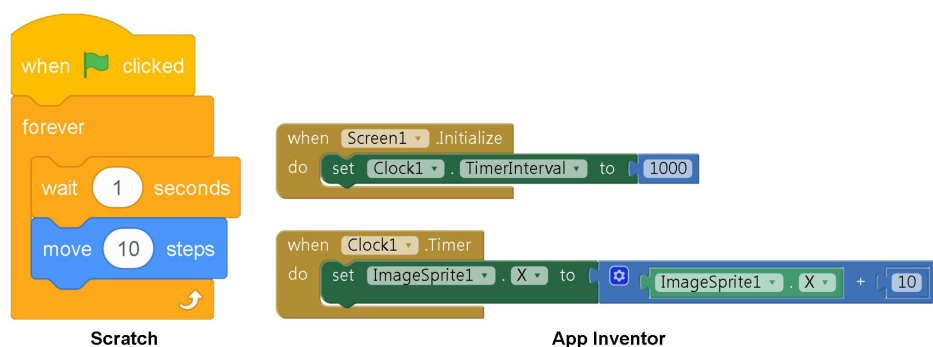


Figure 4: Graphical style of Scratch vs App Inventor [10]. App Inventor is slightly more sophisticated.

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 [10].

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 [8]:

- 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 [7]. 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.

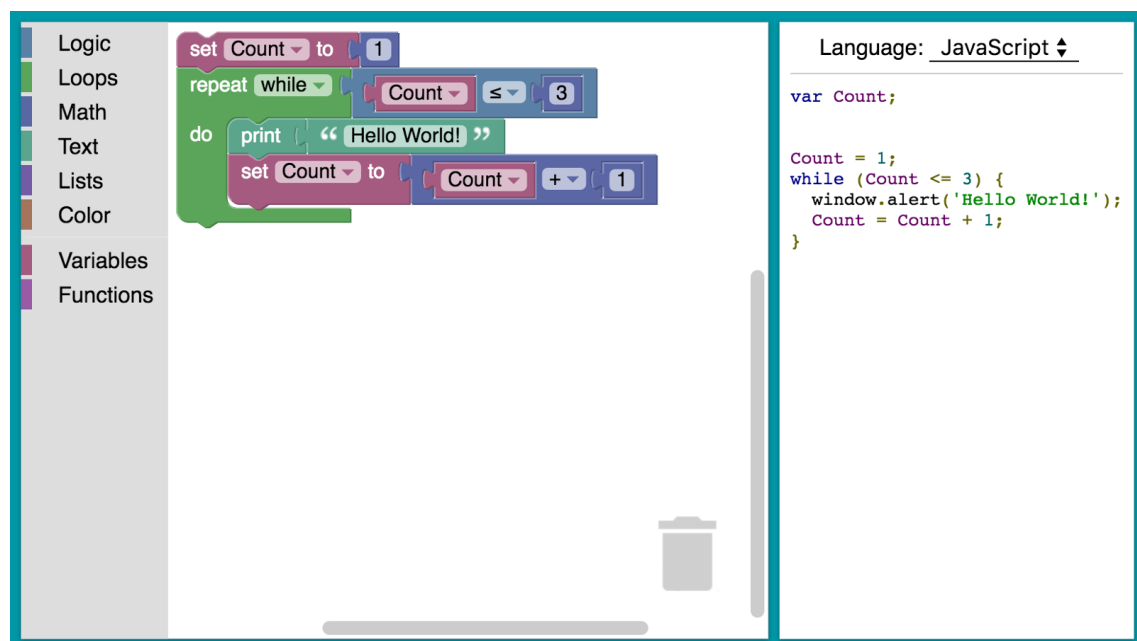


Figure 5: Blockly’s modern and simple block programming design, as well as the option to choose the programming language that it will syntactically output as [14].

### 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"*.

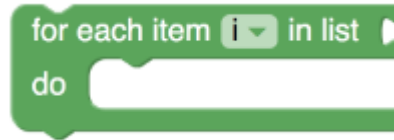


Figure 6: The lowercase convention, as followed by Blockly as an example [11].

Graphical programming is not a permanent solution to programming, but rather an introduction [3]. Users are being taught the convention of the key syntax that is used across all languages. I plan to follow this convention too. A picture is worth a thousand words [12], and reading code isn't learnt nearly as fast and reading a picture. The strength of graphical representations is that they complement perceptually something also expressed textually in this case.

## 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

Below in figure 7 is a gantt chart outlining my process to progress through this task.

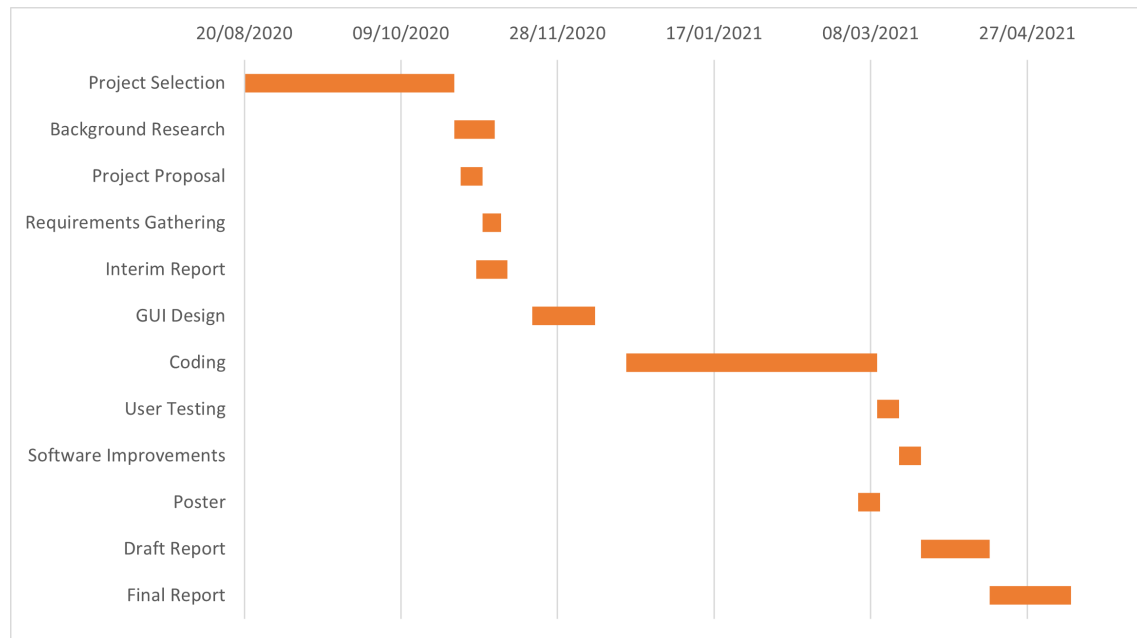


Figure 7: My gantt chart displaying the process I intend to follow throughout the project.

### 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 - outlining my objectives, aims and motivations for the project.
- Background research - related work to the project I am working on.
- Requirements gathering - software requirements that must be met for the project to work, as well as some desirable objectives for quality of life improvements!
- 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.

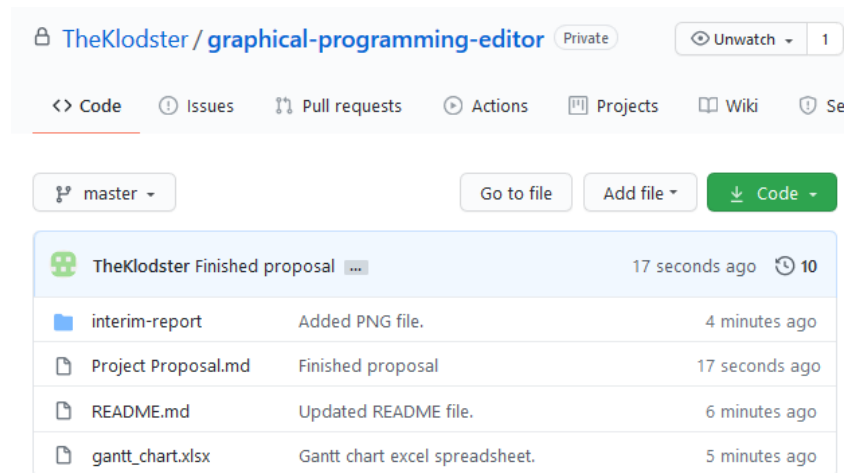


Figure 8: My Github private repository for the project.

## 6 Main Report

### 6.1 Designing a rough GUI

What I do at the beginning of this project is crucial to what will come up ahead since this will be the foundation of the entire project. How I will present the work to the user. Before I can begin getting into the deep fundamentals of this project, I must first, but most importantly of all, design the basic canvas GUI (graphical user interface) for the user to be able to see! This initial design of this interface will be a blank canvas, hopefully with a left-hand side panel where the blocks will be.

To begin, I will start with the blank canvas, then leading onto dividing the canvas into two so there is a left side panel, and in this panel, a print block that the user will be able to drag and drop. The functionality will not matter currently at this stage, but it's essential I research how I will implement the blocks, as this is the commitment I will need to take for the rest of the project. I will start with making the blocks out of buttons. I should note that the reason the start of this project is the most important and requires careful thought and consideration is simply that this is not a small project. I must ensure that whatever I do now will work for the remainder of the project, and that it will be compatible with any functionality I will implement later. If I don't consider this carefully, there may be a chance that I might hit a dead end and I will have to come back to this stage, at a time where I shouldn't be.

Upon building basic GUI upon running the program, it would be nice if it was centered so it can be visible on any resolution that the user will be using. Thinking it would be a simple case of researching the Java JFrame documentation, I did not find what I was looking for and branched out. *Jack* from this thread about setting up JFrames to the centre of the screen regardless of resolution provided an excellent function which takes the resolution of the screen and mathematically applies divisions so that it will be centred of any screen resolution we use [2]. Figure 9 shows a rough sketch about the idea I have for what the interface will roughly look like. This may change in the near future when I begin implementing the JPanels, but for now this sort of design would look appropriate and make sense for the user. This type of formation is adopted too by other applications such as Scratch and App Inventor - both use a design similar to this where the control panel is on the left hand side with all the different blocks the user can use, followed by the big canvas in the centre.

Upon implementing the buttons I had them hugging the sides of the button panel, which didn't look appealing at all, so I had a read through the BorderLayout documentation which revealed that I could enter some padding to sort out the issue.

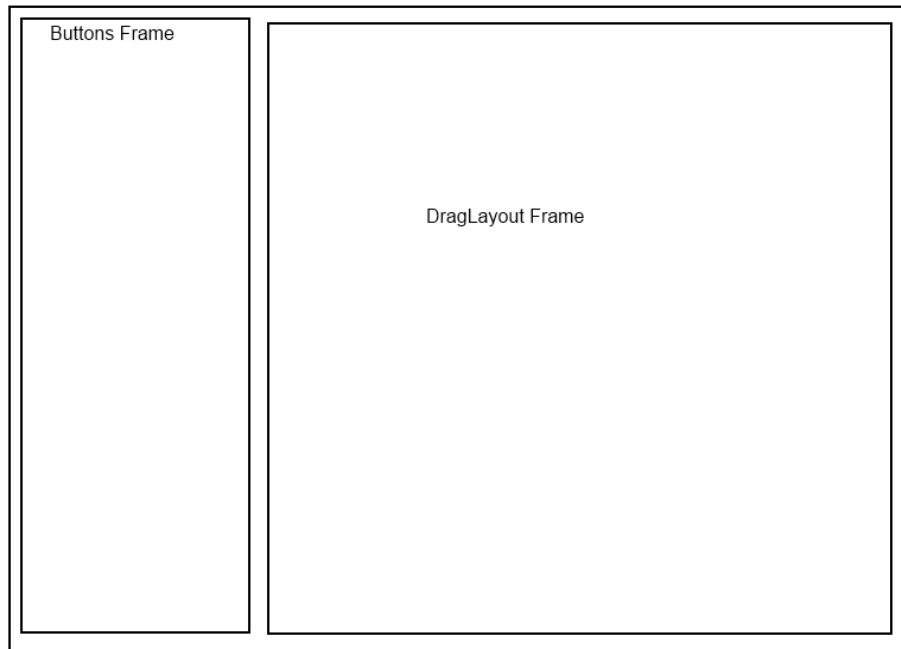


Figure 9: Drawn out plan on essentially what the interface may look like, subject to change.

In regards to the console output for the user, I could do this several ways. A particular way I had in mind specifically was to research whether Java Swing could be able to open another window showing the output as a result of running the user's program. This is not the focus for the time being, so more on this later. Furthermore, I didn't want the user to be able to resize the program as this could potentially lead to issues in the GUI not showing the user all the information they need to work, or not giving them enough space - a simple `frame.setResizable(false);` worked like a charm.

I ran into a problem whereby in Java Swing, when I added padding to the border edges that are touching the edges of the JFrame, the background colour filled outside the border.

Firstly, the background of the panel is painted. Then the border is painted afterwards - whether a compound border function is used where the function allows you to specify the border objects for the outside and inside edges. Secondly, then the border is painted on top of the panel. In the case of my panels, only the line is painted on top of the background. I will be using a compound border to adjust the margins for the borders around the panels. I will run into this issue where the white background colour I choose will overflow.



Figure 10: The white background overflowing above the border in the top image; my solution to the current problem, and the gray background now just appearing as a white background. No one liked a gray background anyway.

## 6.2 Copy Constructor

The procedure forward is to design the application only using a print statement. Upon succeeding with said functionality, the follow up for the remainder of the implementations would function more or less the same, and it should be an easy job of copy and paste. I created a button named **print statement** that would be the button used to initialise a print statement. The next important step was reproducing this in the main canvas panel. I implemented functionality that listens to the mouse clicks, thus once the button is pressed, it clones it on the canvas to be used. Initially I was going to implement the **Cloneable** library but I came across an article that suggested otherwise. **Cloneable** is broken and shouldn't be used as the architecture was essentially mistaken, and it's only there for backward compatibility reasons [15]. So I followed up by implementing the copy constructor suggestion that takes a component as an argument and returns the said component as a new entity.

```
private Component cloneSwingComponent(Component c) {  
    try {  
        ByteArrayOutputStream baos = new ByteArrayOutputStream();  
        ObjectOutputStream oos = new ObjectOutputStream(baos);  
        oos.writeObject(c);  
        ByteArrayInputStream bais = new ByteArrayInputStream(baos.toByteArray());  
        ObjectInputStream ois = new ObjectInputStream(bais);  
        return (Component) ois.readObject();  
    } catch (IOException | ClassNotFoundException ex) {  
        ex.printStackTrace();  
        return null;  
    }  
}
```

Figure 11: Some caption of the constructor.

Upon completion of cloning the button blocks available to the user on the button panel, the user will need the ability to edit what's inside the print button to their choosing. I implemented a simple window to popup to the user, whereby the user can enter what they wish to be printed.

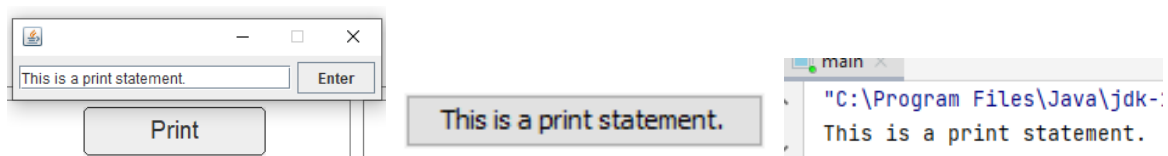


Figure 12: Ability to create a print statement block, with a text of the users' choosing, and successfully print it out to the console.

### 6.3 DragLayout & ComponentMover

It is a key objective, although it may not be explicitly noted in this report, but it is implied, that when designing a project such as this that the user has, not only to be able to interact with the features the program provides, but be able to edit these features - one way being through movement. The user will need access to drag components around to their desirable positions. *Rob Camick's* blog on this specific feature provides a `DragLayout.java` class file that does exactly that, and it is paired up with another class file `ComponentMover.java`. Both classes are dependent on each other to work [4]. The decision was between using a null layout or a drag layout manager. The answer is drawn from the 3 different functions a manager performs:

- sets the locations of the components in the container.
- sets the size of the components in the container.
- calculates the preferred size of the container.

A null layout is a layout that technically it's a legitimate layout manager. This means that no layout manager is assigned, meaning the components can be put at specific ( $x, y, z$ ) coordinates. It can be particularly useful for making quick prototypes but that is essentially it. When dragging components inside a `JPanel` container, it should be clear that the manager should not replace the location of the component. That being said, there's no reason a layout manager could *not* set the size of the object to calculate the preferred size for the container. Essentially, the `DragLayout` manager was designed to replace a null layout.

The manager uses the `ComponentMove.java` class to drag the objects. `DragLayout.java` is just the foundation - it is not responsible for the moving of any components. One could use a `MouseListener` to handle the `mousePressed` event to track the original location of the component, followed by a `MouseMotionListener` to handle the `mouseDragged` event so the component can allow itself to be moved via each drag event. However once tested, I notice this was producing flickering results on the components that I was trying to move on the drag layout. This wasn't appealing and rather distracting instead, the user's experience would decline as a result. `ComponentMover.java` gives us the flexibility for controlling which component is responsible for moving a window [5].

In regards to my action listener events, when a button on the side panel is clicked, it's going to do the same process every time, no matter what button is being clicked. It would be lazy to create action listener events for every single button to do the exact same process. Thinking ahead for when I add future buttons to the side panel, I decided to create an action listener variable so that all the buttons can call this single event instead creating the same event for every button.

```
private ActionListener spawn = new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        JButton button = (JButton) e.getSource();  
        Component clone = cloneSwingComponent(button);  
        dragPanel.add(clone);  
        cm.registerComponent(clone);  
  
        dragPanel.repaint();  
        dragPanel.revalidate();  
    }  
};
```

Figure 13: Creating the action listener one time and one time only to be used for all the buttons.

## 6.4 Deleting blocks

An important objective for the user is the ability to edit their programs. This includes deleting blocks they have made by mistake/no longer want anymore. There isn't any sense in having the user close and re-open the application each time they need to delete a block as this would also clear all their progress. This isn't smart.

There are several concepts to think about when implementing this. Key questions to ask include *what will I use to delete a block* and *how will I use this to go about deleting the block*. There are many ways to go about this, one being that there may be a recycling bin in the corner, where the user would be able to drag their blocks over there and drop them, causing the blocks to be deleted in the process. App Inventor uses this method, and Scratch uses a method where you drag the blocks back into the block palette on the left hand side - where the blocks are to create your program - and this would cause those blocks to be delete too. Both works with stacks of blocks too if they are interconnected.

Both current methods work and sound great - however I want to be a bit more simple than that. My plan is to setup a new java class that deals specifically with deleting blocks. This class will extend the `MouseAdapter` library - this library will be the answer as it holds the functions that allow me to read mouse clicks - such as two specific functions that I will override: `mousePressed` and `mouseReleased`.

Using these two functions together, I can detect the mouse clicks on the button and use these to delete a button. How so? By use of the right click button. Currently, the user has no reason to use the right click mouse button as it serves to purpose - this is about to change. The user will be able to right click any block, resulting in deletion. This will work dynamically at runtime, so the user will not need to restart the application. `mousePressed` will become true when a mouse click has been pressed, followed by the function `mouseReleased` which will confirm the press and a new condition if that press is a right click - if so then it will proceed to remove the button and update the panel. Finally, all that's left is for me to call this class on the clones that are created, a simple one line.

Figures 14-17 illustrate my point in practice, note this this instance is performed all at runtime, hence the proof via console screenshots showing each program being run within the overall instance.





Further in Figure 16, I delete the variable and continue to run the program.

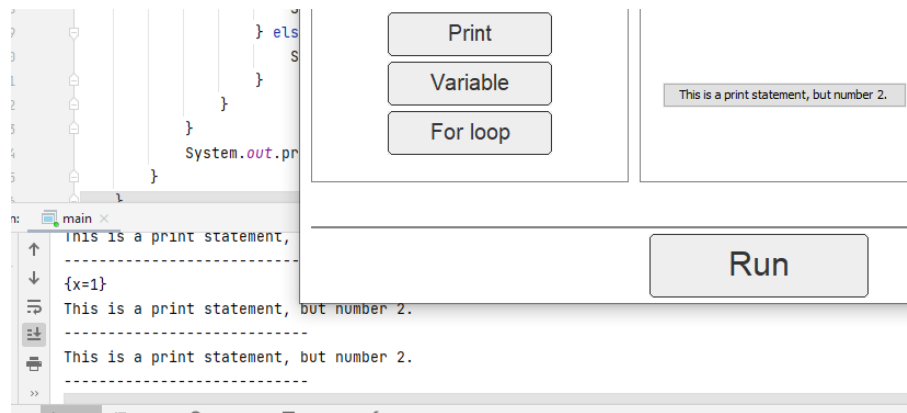


Figure 16: Deleting the variable and running the program, all working as intended.

Finally, I delete the final print statement. Programs cannot be run if there are no contents in the panel, thus I create a new statement, and run this too.

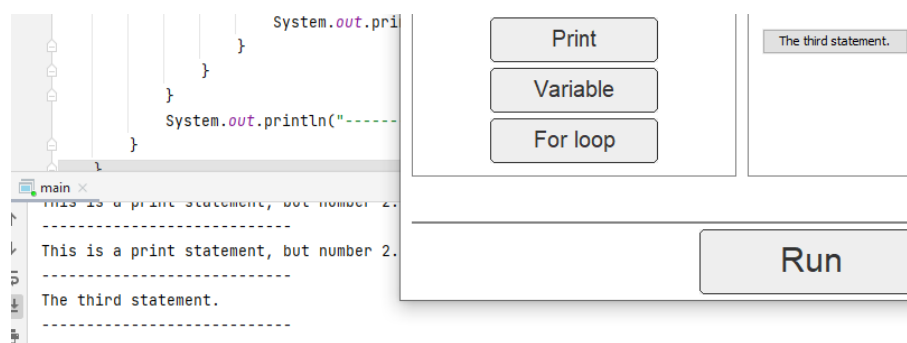


Figure 17: Deleting the final print statement, and reintroducing a new one!

I now have a fully functional deleting mechanism allowing users to edit and change their work as they go.

## 6.5 Maintaining order of the program

A crucial objective to be fulfilled in this project is the flow of the program and ensuring that where the user places their blocks determines its position in their program and how they are interacted at compile time. It is key that if the user declares an assignment variable, followed by a print statement in their program printing the variable, that when they proceed with the **Run** button, there is a mechanism behind the scenes that will maintain a block structure. So when the print statement is met at run time, it understands what it is trying to print out, and where to get it from. Maintaining this block structure enables the user to create high level programs and begin thinking about their programs syntactically, which will help them develop skills for writing code in text-based languages.

I used the **Collections** and the **Comparator** library to achieve this. Using these libraries together, I was able to interact with the data structure instantiated by collecting the components of the panel in an arraylist of type **Component**. Using the latter library, I compared the Y coordinate locations of all components in the arraylist and sorted them based on that rule - however if it is apparent that there are several components with the same Y coordinate, then they are further sorted on their X coordinates.

Below in Figures 18-21, we can see how the methods I have discussed work in practice. The console in Figure 18 console reveals to us the block structure working as intended, sorting by their Y coordinate.

I would also like to point out that on my end as the developer, this entire program is being run in one go live at runtime! This is impressive and useful; if I wanted to I could, after full completion, export the program as an executable **.exe** file that users can use by opening it and letting the program run. Users are able to run their own program via the execution button **Run** and this would work live for them as each time they run their own programs, it is being updated and runs with any changes they have made.

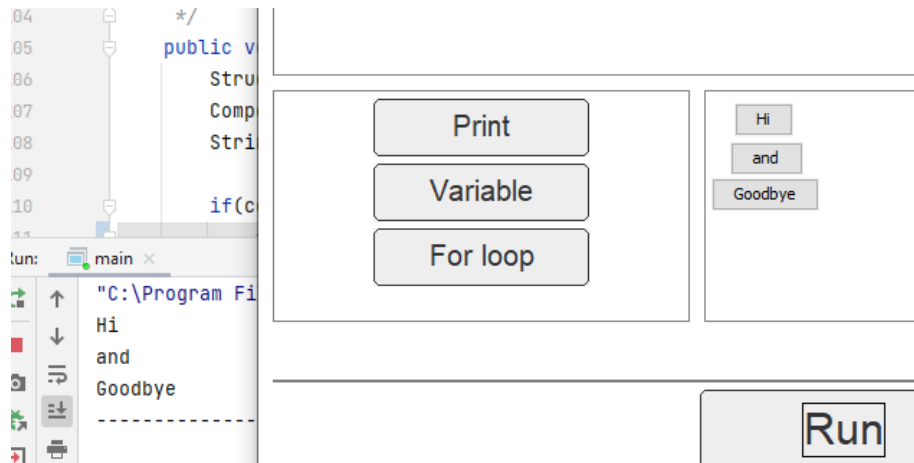


Figure 18: Normal block structure being sorted by their Y coordinates.

In Figure 19, I reajust the program, moving one block further to the right but above the **Goodbye** print block. Running the program again also updates the block structure accordingly and correctly, sorted by their Y coordinate as main priority.

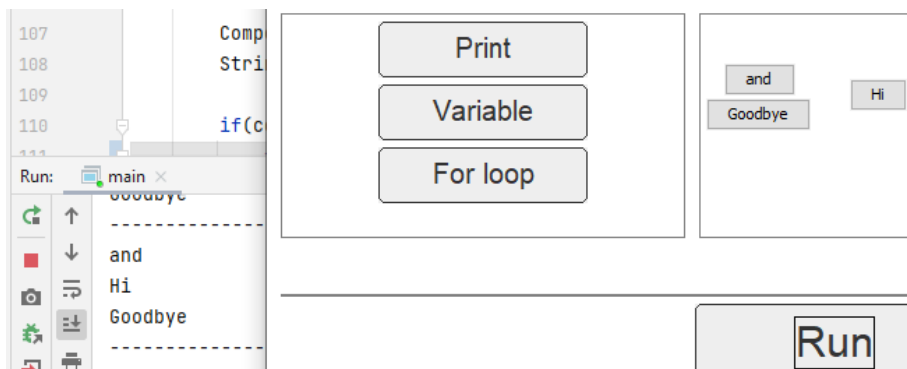


Figure 19: **and** print statement is classed as the first element in this program.

Figure 20 proves how when aligning all of the blocks by their Y coordinates, that their X coordinates are then learnt and used to determine their knew positions. Note that if their X & Y coordinates are identical, then the block that was instantiated first will take priority. Although it would not be ideal as a programmer to have blocks hidden on top of one another.

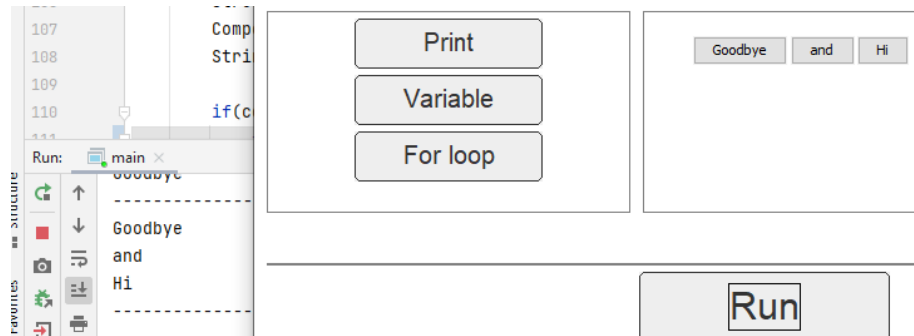


Figure 20: All Y coordinates are the same so they are sorted by their X coordinate.

Furthermore in Figure 21, continuing from Figure 16, if I were to move just a few of the blocks up immediately, the block structure updates again and works as intended.

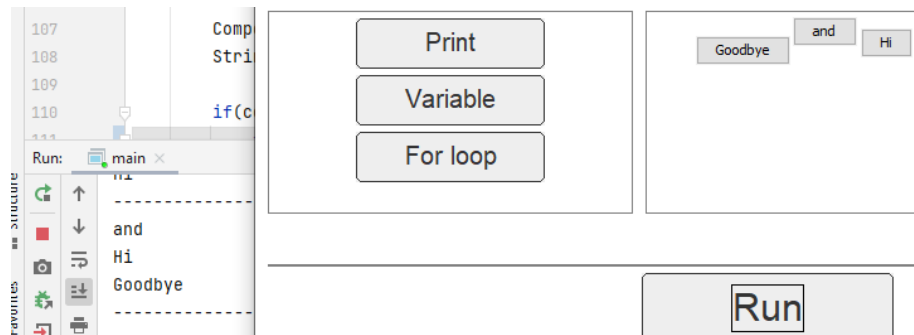


Figure 21: In relation to Figure 20, if we adjust their Y coordinate, it will update its block structure.

Now I have a fully working block structure, capable of allowing blocks to be placed in a syntactical manner that holds true as with almost any other programming languages' structure.

## 6.6 Interpreter

Interpreter comparative programming

## 7 Appendix

### 7.1 Interim Log

#### Friday 23<sup>rd</sup> October

- Discussing what my key interests are.
- Deciding which projects are suitable for me.
- Showing me some of my supervisor's projects.
- Going through the introduction of the project proposal.

#### Monday 9<sup>th</sup> November

- Focusing on time management between project and university studies.
- Strong introduction, that explains what I'm doing!!
- Project proposal:
  - deeper detailed objectives.
  - be more specific if I can.
- grouping objectives into one group, not several.

## 7.2 Project Log



## 7.3 Project Proposal

### Motivations

The motivation behind this project is that it will push my Java programming skills to limit, and experience designing software for other people rather than just for my own benefit. This will help me put my focus on the user's goals rather than my own. Additionally, I can put this project on my portfolio to show future employers!

This project will further put content learnt from modules in my degree to practice. Time management techniques from Software Engineering will help me manage when and what I should be focusing on. Furthermore, concepts learnt from Human Computer Interaction will aid me in focussing on user design for the user's goals; implementing a user interface that everyone will be able to use effectively.

Finally, the ability to use Java this extensively will most definitely improve my proficiency of the skill!

### Aims

The aim of this project is to design and implement a user friendly 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 in 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.

### Objectives

The main objectives are:

1. To investigate system requirements and produce a requirements specification.

2. To select and justify an appropriate research design for the project.
3. Implement a simple user-friendly interface which is easy to use.
4. Functional and working loops, conditionals and data structures that can be interacted with.

### **Extra Objectives**

1. Colour coordinate the different functions.
  - for instance, the loops and conditionals can be coloured differently to show more clearly what they represent.
2. Implement an undo button.
3. Implement a redo button.
4. Testing by a regular user, of the software.

### **Relevance**











This project involves modules across my degree, as well as Java; the most extensively used language across my university education thus far. This project will use skills learnt in Human Computer Interaction; test my ability to understand the user's perspective and struggles, involving the evaluation of the software, rather than my own when designing the interface for them to use.

This project also incorporates key concepts understood from Software Engineering. The ability to criticise my own work, collect requirements, manage time and manage an agile approach to the project. Therefore, this project will test my ability to make use of several skills in such a way that they will be used outside of my university education.

### **Resources Required**

None.

## Personal Timetable

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
9am	9:00am–10:00am  Human-Computer Interaction Seminar	9:00am–6:00pm  Computer Science Project		9:00am–10:00am  Introduction to Computer Secu... Lecture	9:00am–6:00pm  Computer Science Project	9:00am–6:00pm  Computer Science Project
10am						
11:00am–1:00pm  Comparative Programming Lecture			11:00am–12:00pm  Human-Computer Interaction Lecture			
1pm						
2pm						
3pm				3:00pm–5:00pm  Introduction to Computer Secu... Lab		
4pm	4:00pm–5:00pm  Introduction to Computer Secu... Lecture					
5pm				5:00pm–6:00pm  Comparative Programming Lab		

## References

- [1] Richard Albritton. “What is App Inventor?” In: <https://learn.adafruit.com/mit-app-inventor-and-particle-io> (2018).
- [2] AmateurProgrammer. “How to set JFrame to appear centered, regardless of monitor resolution?” In: <https://stackoverflow.com/a/2442610/4552257> (2010).
- [3] Gregory Barkans. “Visual Programming and Version Control: Future Considerations”. In: <https://medium.com/vapurrrmaid/visual-programming-version-control-future-considerations-b18c9a49c187> (2018).
- [4] Rob Camick. “Drag Layout”. In: <https://tips4java.wordpress.com/2011/10/23/drag-layout/> (October 23, 2011).
- [5] Rob Camick. “Moving Windows”. In: <https://tips4java.wordpress.com/2009/06/14/moving-windows/> (June 14, 2009).
- [6] REMI Dehouck. “The maturity of visual programming”. In: <http://www.craft.ai/blog/the-maturity-of-visualprogramming> (2015).
- [7] Neil Fraser. “Ten things we’ve learned from Blockly”. In: *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. IEEE. 2015, pp. 49–50.
- [8] “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).
- [9] John Maloney et al. “The scratch programming language and environment”. In: *ACM Transactions on Computing Education (TOCE)* 10.4 (2010), pp. 1–15.
- [10] 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.
- [11] 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.
- [12] Marian Petre. “Why looking isn’t always seeing: readership skills and graphical programming”. In: *Communications of the ACM* 38.6 (1995), pp. 33–44.
- [13] 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).
- [14] “Try Blockly”. In: <https://developers.google.com/blockly> (2020).
- [15] Bill Venners. “A Conversation with Effective Java Author, Josh Bloch”. In: <https://www.artima.com/bloch-on-design> (January 4, 2002).
- [16] 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.