Appendix

1. Initial discussion with the client transcript

Me:
Good morning. This is the first conversation with my mentor Ms [client's last name], who will be assisting me throughout the Computer Science IA. So the given scenario is that AIS economic students often create Economics graphs digitally through clunky, slow, and imperfect means. Many are forced to use drawing tools or modify pre-existing graphs. Clean and accurate economic graphs are essential for Economics IAs, where poorly produced graphs can cause students to lose marks. So Ms [client's last name], I have proposed the following solution to you that we create a program to fix this problem. I believe that the aims of this program should be to one to assist students in creating Economics graphs digitally, both for homework and for their Economics IAs, to allow for a digital playground for students to interact with graphs in limited scenarios with sliders, and movable dots, etcetera. And finally to provide a foundation for teachers to build on as in when the IBDP Economics syllabus changes, as we know this is an international syllabus, we're not in control of what graphs students need without the need for coding experience, which I believe is very important. So you agree to these aims, right?

Client:
Absolutely yes.

Me:
And of course there was a concern that was brought up with regards to ensuring that graphs are not so well completed that it creates plagiarism problems.

Client:
Yes, it would have to be the students' own work. They would need to be able to [pause], there's some basic graphs that can be drawn, demand and supply, but they would need to do that 100% themselves, but they would need to be able to change text so that they are applying their graphs to real situations which are not generic graphs. So when they're reading a news article that they can apply it to that country's currency, the quantity of that product, for example, and have an ability to do that, but you're absolutely right, they do need to be 100% accurate. They do need to be able to have the freedom to move lines and to move positions of lines depending on the situation. But we do need to be careful that they're not preset so that it's not the students' own work.

Me:
And I think we've stated before that this program is not designed primarily as a learning tool for theory. There is an expectation that students understand what they are doing before they use this graph tool because there is no, [pause] the framework that they're given is so rudimentary that you cannot plagiarise based off this. So I do actually have some, two programs that are, programs and websites that I have found on the Internet, one that is free and one that is not free and does not work with Mac. I do understand that one of the requirements that we would need a program that would work on Mac computers because our school uses Mac. So this is an Economics graph maker that I found on the internet called Econ Graph. It is more of an interactive Economics graph maker. So I want your

opinion on the UI the design do you like this design of how the user is able to modify the graphs and sliders?

Client:
Yeah, that's good. The curvy lines are definitely the hardest. I find that when I'm drawing them online as well. Yeah. They look good. Yeah. Because we do have graphs that are not straight lines. So I think that's always the hardest when you're showing a curvature, especially at Keynesian graphs and things like that. So I do think that that's I think one of the main problems sometimes the straight line graphs are okay, but it's things like adding curves. It's adding triangles, adding your rectangles put lines, all those the main problems I think that we have when we're drawing the graph, yeah.

Me:
And I'm not sure if any graphs will need this but maybe it would be possible to offer some level of guidance like if you move this graph [pause], if you move this line this other line will move along because there is specific correlation.

Client:
For example, it would be good, yeah. Because you're a Computer Science student, we do have well, it's not aligned to a Computer Science grouping so some students won't have maybe that same level of knowledge as you have.

Me:
Yeah. And then finally here's another program. This is unfortunately a Windows program but is there anything, and unfortunately this is not interactive as such because it is a Windows program, but is there anything about this design that you are particularly interested in that you would like to be integrated it into the design of the final product yeah.

Client:
So we try to avoid colour because colour, doesn't we don't use colours on Economics graphs because they're scanned in possibly black and white. So colour wouldn't be necessary. The labelling of the lettering, it looks good on this one, and the ability to put the arrows to show the shifters is important. And these like sort of dotted lines going down, they sort of [pause], I like the lettering against the dots because I do find that students hand in graphs where the letter is not close enough to the dots. So it makes it very confusing to read. So I think the labelling being close to the point of reference is important. And definitely the ability to put the arrow shifters. But I would say no need for the colour.

Me:
So because there will be no colour, would you like to, have some sort of like, shading tool?

Client:
Shading is good. Yeah, you can use shading, especially for like, welfare loss triangles and things like that. That's often good. So shading is acceptable. Probably one shade per graph, shaded area. We show revenue rectangles that they can be labelled with the lettering system and the ability you might want to put in making sure you can do percentage shifts and things like that to show the actual data from the articles on those graphs as well. So an ability to put currency symbols, make sure you have access to be able to put all of the

currencies on and things like, tax free text would be useful. But yeah, it looks clear. Room for Title. Is there a way that I don't know how possible, but it's incomplete unless it has all certain things. The basics of any graph should be a title axis labelled.

Me:
So just provide...

Client:
A checklist? Maybe?

Me:
Yeah. We can certainly integrate that kind of checklist.

Client:
That would come from feedback from me. It would be like check your graph and I can't really say what they're checking, but it should be under the assumption that a student has applied title has all lines and axes labelled, so it would show as incorrect if, say, you've drawn lines that have no labelling.

Me:
Yeah, we can certainly try to integrate that within the system and we can also integrate specific advice and guidance. Hey, we need to align this curve with this curve. The correlation should be here. We can apply that to certain curves as well. That's something we can certainly look into. Finally, I would like to present to you my design, which I Unfortunately have forgotten to bring with me today. But this is a sketch that I actually drew. So what I propose is, sorry if it's a bit small. What I propose is that we create something like this. So we would have the graph in the middle, we would have a toolbar on top. There will be an arrow tool as well, and a shading tool as well. Essentially what would happen is we would have all the settings for the graphs on here and then there would be sliders and you can also type in values if you know these values. So for example, let's say it's demand you can determine how much you want to shift the demand curve you can determine the elasticity of the demand curve for example and sometimes we might have say an advanced option for things that may be useful to students who are reading supplementary materials for example. In the middle you can see that there is a blank economics graph. This will be the baseline for all Economics graphs. Maybe with the exception of the game theory matrix matrix maybe we would create something specialised for that but that would be an exception.

Client:
This kind of axis would cover most of the course so it would be a definite basic setup for most and I think having that basic set up we can avoid the plagiarism side of things as well because then everything the student does from this point on is their own work. So it's definitely for the use of it being a tool to support learning rather than giving completed graphs out.

Me:
And then on the left I will admit I have not determined what would be the cleanest way of doing this but essentially you would have a menu of different graphs, of different curves that is available to the user.

Client:
Yeah because a lot are going to be linear, a lot of the ones we do or like the one on the screen now, the production possibilities, like a kind of curvature, so you are going to be even like we, haven't got to that in the course, yet, but there's going to be a curved Keynesian one, so the other way around that faces up but that will be another design. But all your international ones it will be straight lines, quotas, tariffs all of those and your Macro ones are very similar to the Micro style but with different labelings. In these ones is that showing different values so that you can predetermine the value for the curves? Yes.

Me:
For the curves? Yes. Yes, so that gives them the customisation and would avoid plagiarism because we are forcing them to input values on how they want their curve to be.

Client:
In your Theory of the Firm diagrams, obviously there's more like, specific ways that they like, MC curves and things look. But yeah, if you can do this, that would be amazing. I think I would use that as well. So I definitely find it tricky to use it through Google Drawings or I use Google Slides or something, like not too sensible. So I think, yeah, this is a really worthy project.

Me:
I want to quickly go over the customization because we did speak about it earlier in the conversation. So I will admit I have not determined how this would be implemented. But essentially what I'm proposing is that we create a formatting guide that allows teachers like yourself and Ms [another teacher's last name] and whoever else would need this, to add in additional curve presets as they're needed as the syllabus changes. But I will need to let you know right now that in order for this to work, in order for you to create your own graphs for this, you would need to know the mathematical equations to all of these curves. If you know them, then we can create this fairly easily.

Client:
We use linear equations in the old syllabus so we don't teach you that. But there are equations that we use to determine the slope and the sort of length of the line. So I think we spoke, sort of briefly once the demand function equations and the supply function equations, you could use that because that should predetermine the lines based on a data set. I can't tell you that with IB they do syllabus changes every ten years. My understanding is so I wouldn't worry too much about it changing because we've just changed syllabuses. This is the first year, so it's going to be ten more years before that will change. So you can actually, if you use the Tragakes graph pack that I gave you, you can see all the graphs that you need to know, and that will be a really good basis for this project.

Me:
We just want to make sure that this is something that can be sustained in the future. Certainly modularity, the ability to change the program is certainly something that I'm looking into.

Client:
Yeah, it's wonderful. It's definitely a problem that we have. There is a shame. Sometimes we don't really like hand drawn graphs because it's much clearer in an IA to have an electronic version. But I do say I do drop marks in my marking because they're not accurate graphs. And that is down to just you can see the frustration of the student trying to move lines and things to make it. So I do think it's a really worthy project.

Me:
We did speak a little bit in the past about doing a trial with this. I'm well aware that you plan to use this yourself for your presentations, and we have also discussed about the potential of next year's Year Eleven cohort using this program and trialling it. When do you think we can get them to start using it, and when do you think we can complete the trial? Because we do need this to be done within a certain period of time to align with IA due dates.

Client:
How long will it take you to set up?

Me:
Okay, so this project is to be completed... The majority of it is to be done over the holidays along with all my other IAs, and then there will be a rough draft ready in around sometime in Term One.

Client:
So you are going to have in Economics, we are going to be doing Macroeconomics IA in Term One and a Global Economy IA in Term Two. So you are welcome to trial this with your classmates, all of them, or select a few of them, offer it to them, perhaps with your current cohort, your current class in Term One or Two, or when you feel it's ready, because they will need to draw graphs as part of those projects, as part of those IAs, now Year Eleven, obviously, we'll be starting with a new class in January. We don't do the IA until Term Three, so that much that will be too late, but it could be used with homework or something at the beginning. We could definitely try something in Term One and say whilst they're learning to draw graphics, there's graphs all the way through Economics. So we could definitely, I could work with you to integrate that into the inquiry tasks we do with the new Year Eleven. So you've probably got a few, three different ways you can test it. One, I can test it for use for use on my website because I like to post the graphs and because I'm redoing the course I can be creating and doing it anytime, as I'm teaching, putting the teaching materials together. Your current class could trial it in Term One or Term Two with one of those two IAs, and that will probably be really good feedback because they understand graphs and they understand what's going on, and I could definitely roll this out to my Year Elevens should I have Year Eleven next year in Term One or Two. But it won't be for the IA because that will be Term Three.

Me:
Okay.

Client:
Which will be too late.

Me:
Yeah. I'm assuming it will be too late.

Client:
I think you've probably got a lot of people who can potentially use it. We could do it as revision as well. So people are creating revision materials as well. So if you had a sample of your classmates who could do it outside as well to create provision cards or photographs and things like that. So I think you've probably got enough people who could trial this out and I'll definitely give it a go and see if, I'm always drawn graphs for the website so I'll see if I can use it, easily. So you can have me and all the other economics teachers as well, Mr. [another teacher's last name] has taken over Ms [another teacher's last name] as well.

Me:
And just to let you know, part of the Computer Science IA is to include user documentation so there will be some sort of PDF provided to you so you're not completely in the dark and so that I don't need to sit with you or sit with other students when they're using the program. There will be a guide.

Client:
Like a reflection of how they found it, and things?

Me:
Like a manual on how to use it.

Client:
Well, I'm happy if you wanted to introduce it to the class next year. Once you feel at a point where it's ready to be used, you're welcome to take some lesson time to say, right, I created this. This is a tool you can use and people may use it for their IA. Definitely. People are just using Google Drawings right now So they might find it's really good. So you're welcome to introduce that in class. That would be fine. If it helps. All good.

Me:
Thank you, Miss. Well, I think that concludes our conversation. Thank you so much and I hope you have a wonderful day.

Client:
Thanks [My name].

2. Final discussion with the client summarised

- Relative to the initial conversation, the program is incomplete and basic
    - The presets functionality is not present, a suggestion that in a future product that it is included
    - The settings pane functionality is not present, a suggestion that it is included in a future product
    - Graph files cannot be saved, meaning that after the program is closed, all graph objects would need to be recreated
    - The recents panel is no longer present
    - The checklist of things to take into consideration when doing IA graphs is missing
    - The functionality to see whether the checklist of things to take into consideration when doing IA graphs is also missing
- Although usable, and working without bugs, many quality of life features are missing
    - Objects are not draggable
    - Objects cannot be deleted with the delete key
    - Objects cannot be copied and pasted
    - You cannot name the image file
    - Toolbar buttons do not have any indicator of if they are selected or not
    - Undo and redo functionality is not present
    - There is no ability to change the text size
    - Creating curved lines is a pain, as the control points cannot be moved
    - The pen draw function should have auto smoothing to it, to make it more usable
    - Dark mode does not exist
    - More interactivity with straight line drawing and curved line drawing would have been appreciated
    - A border around the text box that is currently being edited would have been useful
- The image output is not clear
- The text box seems to move upwards after every line added. Please fix this for next time.
- Nonetheless Economic diagrams can be created using this application that would meet the standards for IA work, provided the issues with image clarify are fixed
    - However, wider diagrams such as the business cycle diagram cannot be created
    - Diagrams that utilise a quarter of a circle cannot be created
- The design is minimal, and there is no confusion in what did what
- The program did not break whilst using it
- The program would be faster than using Google Drawings, but this may be attributed to how this program lacks the auto connect functionality of Google Drawings

3. Codes

App Delegate (the main class)

```
//
//  AppDelegate.swift
//  Economic Graph Maker Version 2
//
//

import Cocoa

// Indicates that this class is the entry point for program execution.
@main
// Identifiers after the colon are the class that AppDelegate is subclassing, followed by the protocol that it his implementing, in this case, NSApplicationDelegate, which allows AppDelegate to utilise functions to control the functionality and behaviour of the application.
class AppDelegate: NSObject, NSApplicationDelegate {

	// A function that is called once the program finishes launching
	func applicationDidFinishLaunching(_ aNotification: Notification) {
		// Insert code here to initialize your application
	}

	// A function that is called after the program has been called to terminate
	func applicationWillTerminate(_ aNotification: Notification) {
		// Insert code here to tear down your application
	}

	// A function that is called to state that this application supports the use of persistence storage to allow the app to be restored to its previous state securely when the application is terminated
	func applicationSupportsSecureRestorableState(_ app: NSApplication) -> Bool {
		return true
	}
}
```

GraphToolbar

```
//
//  graphToolbar.swift
//  test3
//
//

import Cocoa

// A class containing the methods of the graph editor toolbar
class GraphToolbar: NSToolbar {

	// A function that is linked to the pen draw toolbar item
	// "_sender" means that the sender (which can be any data type as the "Any" data type is a wrapper that all types, classes, protocols, objects, structs, enums, and other variables and structures in Swift conform to
	// The use of "_" means that this label does not need to be written when parsing in anything the programmer wants into the function
	// This parameter is standard for functions that are called, whether by an interactive UI element, or by a selector
	@IBAction func penDrawAction (_sender: Any) {
		if (GraphView.actionInt == 0) {
			GraphView.actionInt = -1
			GraphEnvironmentViewController.actionInt = -1
		} else {
			GraphView.actionInt = 0
			GraphEnvironmentViewController.actionInt = 0
		}
	}

	// A function that is linked to the straight line toolbar item
	@IBAction func straightLineAction (_sender: Any) {
		if (GraphView.actionInt == 1) {
			GraphView.actionInt = -1
			GraphEnvironmentViewController.actionInt = -1
		} else {
			GraphView.actionInt = 1
			GraphEnvironmentViewController.actionInt = 1
		}
	}

	// A function that is linked to the curved line toolbar item
	@IBAction func curvedLineAction (_sender: Any) {
		if (GraphView.actionInt == 2) {
			GraphView.actionInt = -1
			GraphEnvironmentViewController.actionInt = -1
		} else {
```

```
                GraphView.actionInt = 2
                GraphEnvironmentViewController.actionInt = 2
        }
    }


    // A function that is linked to text toolbar item
    @IBAction func textItemAction (_sender: Any) {
        if (GraphView.actionInt == 3) {
            GraphView.actionInt = -1
            GraphEnvironmentViewController.actionInt = -1
        } else {
            GraphView.actionInt = 3
            GraphEnvironmentViewController.actionInt = 3
        }
    }

    // A function that is linked to the delete toolbar item
    @IBAction func deleteItemAction(_sender: Any) {
        if (GraphView.actionInt == 7) {
            GraphView.actionInt = -1
            GraphEnvironmentViewController.actionInt = -1
        } else {
            GraphView.actionInt = 7
            GraphEnvironmentViewController.actionInt = 7
        }
    }

    // Data refers to a data type of raw bytes in memory
    // As such, it is important to keep track of what type of data is being stored when the Data data type is used
    // In this scenario, PDF data is being stored to the variable
    // This variable is an optional as the variable may not always have something stored inside of it, as the graph view may not have been initialised
    static var graphViewToSave : Data?

    // A function that is linked to the save toolbar item
    @IBAction func saveAction (_sender: Any) {

        // The panel is first saved to a variable, to allow for changes in attributes, and later invocation
        let savePanel = NSOpenPanel()
        savePanel.canCreateDirectories = true
        savePanel.allowsMultipleSelection = false
        savePanel.canChooseDirectories = true
        savePanel.canChooseFiles = false
```

```
                // The panel's function is invoked as a closure, whereby "result" is the variable that holds the results of the user interactions. As functions are meant to have return types, the use of -> symbolises that this closure returns nothing after finished
                savePanel.begin { (result) -> Void in
                        // An if statement that is true if the user clicks on the OK button within the panel
                        if result.rawValue == NSApplication.ModalResponse.OK.rawValue {
                                // The directory that the user plans to save in
                                let saveLocationURL = savePanel.directoryURL

                                // The directory and the file name combined. The line below will yield the directory + "Untitled.tiff", as the user has not and cannot enter a name for the file into the name field, as Open Panels do not have that
                                // ".tiff" is appended to the end of the file name to ensure that no matter happens, the file will always save as a TIFF file image
                                let fileName = saveLocationURL?.appendingPathComponent(savePanel.nameFieldStringValue + ".tiff")

                                // The byte data is created based on PDF data from "graphViewToSave", and then converted to a TIFF image file representation, before it is converted back to bytes for later writing
                                let TIFFImage = Data((NSImage.init(data: GraphToolbar.graphViewToSave!)?.tiffRepresentation)!)

                                // A structure that can handle errors thrown from statements that can throw errors
                                do {
                                        // Attempts to write the TIFF image file to the desired directory
                                        try TIFFImage.write(to: fileName!)
                                } catch {
                                        // For internal debugging. It is not seen by the end user
                                        print(error)

                                }
                                return
                        // An if statement that is true if the user clicks on the Cancel button within the panel
                        } else if result.rawValue == NSApplication.ModalResponse.cancel.rawValue {
                                return
                        }
                }
        }

}
```

GraphEnvironmentViewController

```
//
//  ViewController.swift
//  test3
//
//

import Cocoa

// The view controller of the graph drawing environment. It is subclassed from
NSViewController
class GraphEnvironmentViewController: NSViewController {

	// The toolbar variable, which holds the toolbar within the graph drawing environment
	@IBOutlet var windowToolbar : GraphToolbar!

	// A static variable that holds an 8-bit integer representative of the action to be taken
	// -1 in this program means that not action is to take place
	static var actionInt : Int8 = -1

	// A variable that holds the actual graph view, where drawing takes place
	@IBOutlet weak var graphView : GraphView!

	// A variable that indicates whether the graph view should redraw its view
	var shouldDraw = false

	// A variable that is representative of the area to track cursor interactions
	var trackingArea : NSTrackingArea = NSTrackingArea.init()

	// A function that runs of the primary view of the view controller successfully loads
	override func viewDidLoad() {
		super.viewDidLoad()

		// States that the view needs to redraw
		self.view.needsDisplay = true
		// Initialises the closures that hold the drawing logic
		graphView.varInit()
		// Initialises the tracking area, stating that
		// - The area to track is the area of the graph view
		// - The area should be checking for mouse movements, and should always
be active
		// - The tracking area is owned by the view controller and has zero user info
		trackingArea = NSTrackingArea.init(rect: graphView.bounds, options:
NSTrackingArea.Options(rawValue:
NSTrackingArea.Options.mouseMoved.rawValue|NSTrackingArea.Options.activeAlways.raw
Value), owner: self, userInfo: nil)
		// The tracking area is added to the graph view
```

```
                graphView.addTrackingArea(trackingArea)
        }

        // A constant dictionary that converts graph object IDs to integers
        let objectIDToActionInt : [String : Int8] = [
                GraphObjectsClass.penDrawObjectID : 0,
                GraphObjectsClass.straightLineObjectID : 1,
                GraphObjectsClass.curvedLineObjectID : 2,
                GraphObjectsClass.textObjectID : 3
        ]
}
```

GraphEnvironmentWindowController

```
//
//  WindowController.swift
//  test3
//
//

import Cocoa

// Controls the window of the graph editing environment
class WindowController: NSWindowController, NSToolbarDelegate {
        // Creates the graph toolbar for the entire view
        @IBOutlet var windowToolbar : GraphToolbar!

        // Sets the window title, by returning a String literal of the window title when the
function is called
        override func windowTitle(forDocumentDisplayName displayName : String) -> String
{
                return "Economics Graph Maker"
        }

        // Runs if the window successfully loaded
    override func windowDidLoad() {
        super.windowDidLoad()
    }
}
```

GraphView

```
//
//  graphView.swift
//  test3
//
//

import Cocoa

// A class that describes how the graph view should behave
// It is subclassed from NSView so that it gains the properties of a view, thus allowing focus to be placed on drawing logic
// It is important to recognise the difference between bounds and frame
// Bounds is a rectangle of a view, with the origin of the rectangle set with respect to the coordinate system of the rectangle itself
// Frame however is also a rectangle of a view, but the origin is set with respect to the view's origin within the coordinate system of the superview that it resides in as a subview
class GraphView : NSView {

	// A copy of the action integer
	static var actionInt : Int8 = -1

	// A computed variable that returns a rectangle object of the border to draw
	var borderRect : CGRect {
		// insertBy is a function that inserts a rectangle that is based off the rectangle that is calling it
		return bounds.insetBy(
			dx: borderLineWidth * 0.5,
			dy: borderLineWidth * 0.5
		)
	}

	// A computed variable that returns a rectangle object of the thickness of the border to draw
	var borderLineWidth : CGFloat {
		return min(bounds.width, bounds.height) * 0.005
	}

	// A function that draws the view
	func drawBoarder() {
		// Tells the CGContext to begin drawing
		cgContext?.beginPath()
		cgContext?.addRect(borderRect)
		cgContext?.setStrokeColor(NSColor.black.cgColor)
		cgContext?.setLineWidth(CGFloat(5))
		// Tells the CGContext to draw the border of the rectangle added
		cgContext?.strokePath()
```

```
                // Tells the CGContext to stop drawing
                cgContext?.closePath()
        }

        // An array of graph objects for storage
        var GraphObjects = [GraphObject]()

        // The current graph object that is being drawn, edited, or manipulated
        var currentGraphObject : GraphObject = GraphObject.init(typeInp: "")

        // The current graph object that has been clicked on. It is used mainly to tell the
program when to send a new copy of the current graph view
        var selectedGraphObject : GraphObject = GraphObject.init(typeInp: "") {
                // A structure that holds code to run when the variable has been written to
                didSet {
                        // Ensures that an empty graph view does not even make it to sending
                        if (!self.GraphObjects.isEmpty) {
                                // Sends the graph view to the toolbar class, where the saving
logic is stored
                                // It is sent as bytes of PDF data for later conversion to an
image file
                                // Bounds returns the rectangle of the graph view, with the
origin set with respect to the view's own coordinate system
                                GraphToolbar.graphViewToSave = self.dataWithPDF(inside:
self.bounds)
                        }

                }
        }

        // The Core Graphics context of the view, where low-level drawing occurs
        var cgContext = NSGraphicsContext.current?.cgContext

        // Runs when the view is required to redraw
        // dirtyRect is the graph view represented as a rectangle object
        override func draw(_ dirtyRect: NSRect) {
                // Initialises the view to draw
                super.draw(dirtyRect)
                // Sets the fill of the view to white
                NSColor(red: 1, green: 1, blue: 1, alpha: 1).setFill()
                // Fills the view with white
                self.bounds.fill()

                // Calls the most recent CGContext
                cgContext = NSGraphicsContext.current?.cgContext

                // Invokes the function to draw the border
                drawBoarder()
```

```
                cgContext?.beginPath()
                // Invokes the function to draw all graph objects
                redrawView()
                // Ensures that the border is always on top of the graph objects
                drawBoarder()
                cgContext?.closePath()
        }

        // An identifier that represents a type, which will be used to create a dictionary of
functions to draw the graph objects
        typealias VoidFunction = () -> Void

        // A dictionary of functions to draw the graph objects
        var funcDict = [String : VoidFunction]()

        // A closure that returns nothing. The use of the exclamation mark means that if the
closure is nil, or in other words not initialised, it will invoke a runtime error
        var penDraw : VoidFunction!

        var straightLineDraw : VoidFunction!

        var curvedLineDraw : VoidFunction!

        var textDraw : VoidFunction!

}
```

GraphViewObjectDrawing

```
//
//  GraphViewObjectDrawing.swift
//  Economic Graph Maker Version 2
//
//

import Foundation
import Cocoa

// The extension keyword implies that this code body extends the functionality of the
GraphEnvironmentViewController class
// In this scenario, it is extending the amount of functions that the graph view has within a
seperate file for organisation purposes
// This file only contains functions that pertain to object drawing
extension GraphView {
```

```
        // A function that initialises the closures for usage
        func varInit() {
                // The closure that when called does pen drawing
                // "[self] in" means that the closure is using variables that are from the
GraphView object
                // Without it, all variables encapsulated within the GraphView object would
need to have "self."
                // Attached to the start to signify that the variable is from the GraphView object
                penDraw = { [self] in
                        // Adds all the pen draw points to the CGContext
                        // A forced downcast (as seen through "as!") forces the program to
attempt to use "currentGraphObject" as a "PenDrawObject", in order to access the
"penDrawPoints" array, and if it cannot be downcasted, then the program invokes a runtime
error, and the program stops
                        cgContext?.addLines(between: (currentGraphObject as!
PenDrawObject).penDrawPoints)
                        // Tells the CGContext to draw a path through all of the pen draw
points
                        cgContext?.strokePath()
                }

                // The closure that when called does straight line drawing
                straightLineDraw = { [self] in
                        // A check to see if the end coordinate has been written by the user, as
the point (-1, -1) cannot be written to the variable by the user
                        // A forced downcast (as seen through "as!") forces the program to
attempt to use "currentGraphObject" as a "StraightLineObject", in order to access the
specified variables that a "StraightLineObject" would have, and if it cannot be downcasted,
then the program invokes a runtime error, and the program stops
                        if ((currentGraphObject as! StraightLineObject).endCoord !=
CGPoint(x: -1, y: -1)) {
                                // Moves the start of the drawing to the starting coordinate
                                cgContext?.move(to: (currentGraphObject as!
StraightLineObject).startCoord)
                                // Adds a line from the starting coordinate, to the end
coordinate
                                cgContext?.addLine(to: (currentGraphObject as!
StraightLineObject).endCoord)
                                cgContext?.strokePath()
                        } else {
                                cgContext?.move(to: (currentGraphObject as!
StraightLineObject).startCoord)
                                // Draws a circle to assist users in understanding where the
start of their line is
                                cgContext?.addEllipse(in: CGRect(x: (currentGraphObject as!
StraightLineObject).startCoord.x, y: (currentGraphObject as!
StraightLineObject).startCoord.y, width: 5, height: 5))
                                // In this scenario, the circle is drawn
```

```
                        cgContext?.strokePath()
                    }
                }

                // The closure that when called does curved line drawing
                curvedLineDraw = { [self] in
                    // Determines if the curved line is a quadratic bezier curve
                    // A forced downcast (as seen through "as!") forces the program to attempt to use "currentGraphObject" as a "CurvedLineObject", in order to access the specific variables that a "CurvedLineObject" would have, and if it cannot be downcasted, then the program invokes a runtime error, and the program stops
                    if ((currentGraphObject as! CurvedLineObject).bezierControlPoints.count == 3) {
                        cgContext?.move(to: (currentGraphObject as! CurvedLineObject).bezierControlPoints[0])
                        // Creates a quadratic bezier curve
                        cgContext?.addQuadCurve(to: (currentGraphObject as! CurvedLineObject).bezierControlPoints[2], control: (currentGraphObject as! CurvedLineObject).bezierControlPoints[1] )
                        cgContext?.strokePath()

                    // Determines if the curved line is a cubic bezier curve
                    } else if ((currentGraphObject as! CurvedLineObject).bezierControlPoints.count == 4) {
                        cgContext?.move(to: (currentGraphObject as! CurvedLineObject).bezierControlPoints[0])
                        // Creates a cubic bezier curve
                        cgContext?.addCurve(to: (currentGraphObject as! CurvedLineObject).bezierControlPoints[3], control1: (currentGraphObject as! CurvedLineObject).bezierControlPoints[1], control2: (currentGraphObject as! CurvedLineObject).bezierControlPoints[2])
                        cgContext?.strokePath()
                    }
                }

                // A group of declarations that associate an object ID to its respective drawing closure, allowing for the closures to be called based on the object type alone
                funcDict[GraphObjectsClass.penDrawObjectID] = penDraw
                funcDict[GraphObjectsClass.straightLineObjectID] = straightLineDraw
                funcDict[GraphObjectsClass.curvedLineObjectID] = curvedLineDraw
                funcDict[GraphObjectsClass.textObjectID] = textDraw
        }

        // A function that redraws every graph object after it is added or removed
        func redrawView() {
            // Iterates through all of the objects within a closure, with the variable "object" as the variable that changes after each loop iteration
            GraphObjects.forEach { (object) in
```

```
                        // Sets the current graph object to draw as the object that is currently
                        currentGraphObject = object
                        // Ensures that the lines are draw as solid lines
                        cgContext?.setLineDash(phase: 0, lengths: [])
                        cgContext?.setLineWidth(5)
                        // A switch-case statement that determines the current graph object
type, and invokes the drawing closure for that specific graph object type
                        switch (object.objectType) {
                                case GraphObjectsClass.penDrawObjectID:
                                        penDraw()
                                case GraphObjectsClass.straightLineObjectID:
                                        straightLineDraw()
                                case GraphObjectsClass.curvedLineObjectID:
                                        curvedLineDraw()
                                case GraphObjectsClass.textObjectID:
                                        // As the text objects have their own drawing logic, this
code exists primarily to show the user where the text box is being drawn as a square. It will
disappear after the user starts typing
                                        // A forced downcast (as seen through "as!") forces the
program to attempt to use "object" as a "TextObject", in order to access the specific variables
that a "TextObject" would have, and if it cannot be downcasted, then the program invokes a
runtime error, and the program stops
                                        if ((object as! TextObject).textView.string.isEmpty) {
                                                cgContext?.move(to: (object as!
TextObject).drawLocation)
                                                cgContext?.setLineWidth(5)
                                                cgContext?.addRect(CGRect.init(origin: (object
as! TextObject).drawLocation, size: CGSize(width: 5, height: 5)))
                                                cgContext?.strokePath()
                                                cgContext?.setFillColor(CGColor.black)
                                                cgContext?.fill(CGRect.init(origin: (object as!
TextObject).drawLocation, size: CGSize(width: 5, height: 5)))
                                                cgContext?.strokePath()
                                        }
                                default: break
                        }
                }
        }
}
```

GraphEnvironmentViewControllerTextViewSizeHandling

```
//
//  GraphViewTextViewSizeHandling.swift
//  Economic Graph Maker Version 2
//
//

import Foundation
import AppKit

// The extension keyword implies that this code body extends the functionality of the GraphEnvironmentViewController class
// In this scenario, it is inheriting the NSTextViewDelegate protocol, and thus its specific functions and characteristics
extension GraphEnvironmentViewController : NSTextViewDelegate {
	// A function that runs when the text within a text box changes, so that the size of the text box changes accordingly
	func textDidChange(_ notification: Notification) {
		// In essence this statement first filters out all TextObject graph objects, and then casts the returned array of text objects, as a TextObject array, so that the program only iterates through TextObject objects, and is already forced downcast as a TextObject, so that forced downcasting is not repeated.
		// The use of $0 means that we are using the 0th variable that is parsed into the closure, which in this case are the graph objects
		// This function checks to see if all of the text boxes have had changes in their contents, and resizes all changed text boxes
		for textObject in (self.graphView.GraphObjects.filter({ $0.objectType == GraphObjectsClass.textObjectID }) as! [TextObject]) {
			// Code that removes the text box if the user removes all the text within it, thus auto-deleting the text box.
			// As this function only runs when the text changes, the text box is able to exist as an empty text box when first created, as the text contents have not yet changed
			if (textObject.textView.string.isEmpty) {
				// Removes the text box from the graph view
				textObject.textView.removeFromSuperview()
				// Rewrites the graph objects array so that the deleted text box is taken (or rather filtered) out
				self.graphView.GraphObjects = self.graphView.GraphObjects.filter({ $0 != textObject as GraphObject })
				// As the text box is now removed, we can now continue
			}
			// NSSize is a struct that contains a width and height variable
			// It holds the longest length of a given line
			var sizeToDraw : NSSize = NSSize.init(width: -1, height: -1)
			// The last character index
			var lastCharIdx : Int = 0
```

```
                // The number of lines within the text box
                var height : Int = 0

                // Enumerated means that the program is iterating through both the
index of the characters within the text box, and the characters within the text box themselves
                for (idx, character) in (textObject.textView.string).enumerated() {
                        // Runs if the character is a newline character
                        // The index check is a sanity check so that if the user
accidentally adds a newline character at the beginning, the NSRange does not state that the
lower bound variable (in this case lastCharIdx) is greater than the upper bound variable (in
this case idx-1)
                        if (character.isNewline && character != " " && idx > 0) {
                                // An if statement that compares a given line to the
largest line width found
                                // NSRange is an object that represents range, from a
lower bound to an upper bound
                                // It is the way that range literals are stored
                                // The attributed string is used at it will take into
consideration the width and the height of the text line with respect to the glyph (a character
with specific characteristics, such as font, kerning, size) that will be drawn
                                if
(NSAttributedString.size(textObject.textView.attributedString().attributedSubstring(from:
NSRange.init(lastCharIdx...idx-1)))().width
                                        > sizeToDraw.width) {
                                        // Updates the new longest width of a line
                                        sizeToDraw = NSAttributedString.size(
                                                textObject
                                                        .textView
                                                        .attributedString()
                                                        .attributedSubstring(
                                                                from:
NSRange.init(lastCharIdx...idx-1)
                                                        )
                                        )()

                                        // As the current character is a newline
character, the next character of the new line should be the next character index
                                        lastCharIdx = idx + 1
                                }
                                // Height is incremented by one, as a newline
represents a new line being created, thus increasing the height of the text box
                                height += 1
                        }
                }

                // A sanity check is done at the beginning, so that when the user adds
a newline character, the lower bound of the NSRange variable does not exceed the length of
the string
```

```
                        if (lastCharIdx <= textObject.textView.attributedString().length-1 &&
NSAttributedString.size(textObject.textView.attributedString().attributedSubstring(
                                from:
NSRange.init(lastCharIdx...textObject.textView.attributedString().length-1)))().width >
sizeToDraw.width) {
                                sizeToDraw = NSAttributedString.size(
                                        textObject.textView
                                                .attributedString()
                                                .attributedSubstring(
                                                        from:
NSRange.init(lastCharIdx...textObject.textView.attributedString().length-1)
                                                )
                                )()
                        }
                        // Height is incremented one final time, as the 0th line is not accounted
for
                        height += 1

//                      textObject.textView.layoutManager?.ensureLayout(for:
textObject.textView.textContainer!)
                        // Changes the frame of the text box to its new size
                        // Frame, is unlike bounds, as its origin is with respect to the
superview, which in this case is the graph view
                        textObject.textView.frame = .init(
                                origin: textObject.drawLocation,
                                size: NSSize.init(
                                        width: ceil(sizeToDraw.width + 10),
                                        height: ceil(sizeToDraw.height * CGFloat(height))
                                )
                        )

                        // Tells the text object that its associated text view must redraw
                        textObject.textView.viewWillDraw()
                }
        }
}
```

PenDrawObject

```
//
//  penDrawObject.swift
//  test3
//
//

import Foundation

// The extension keyword implies that this code body extends the functionality of the PenDrawObject class
// In this scenario, it is adding functions that allow the PenDrawObject class to conform to Equatable through overloading the equality and inequality operators, which it must as PenDrawObject inherits this from its superclass, GraphObject
extension PenDrawObject {
	// Operator overloading the equality operator so that PenDrawObjects can be compared using the equality operator
	// "left" represents the object on the left side of the operator, and "right" represents the object on the right side of the operator
	// "-> Bool" means that this function will return a boolean variable
	static func ==(left : PenDrawObject, right : PenDrawObject) -> Bool {
		return (
			left.height == right.height &&
			left.width == right.width &&
			left.objectType == right.objectType &&
			left.parameters == right.parameters &&
			left.useAdvanced == right.useAdvanced &&
			left.penDrawPoints == right.penDrawPoints
		)
	}

	// Operator overloading the inequality operator so that PenDrawObjects can be compared using the inequality operator
	static func !=(left : PenDrawObject, right : PenDrawObject) -> Bool {
		return (
			left.height != right.height ||
			left.width != right.width ||
			left.objectType != right.objectType ||
			left.parameters != right.parameters ||
			left.useAdvanced != right.useAdvanced ||
			left.penDrawPoints != right.penDrawPoints
		)
	}
}

// A class that describes objects representing pen drawing
class PenDrawObject : GraphObject {
```

```
	// An array of points
	// CGPoint is effectively a struct holding an x-coordinate and a y-coordinate
	var penDrawPoints : [CGPoint]

	// The initialiser of the PenDrawObject
	init() {
		// Writes an empty CGPoint array to the variable
		penDrawPoints =  [CGPoint]()
		// Invokes the initialiser of the superclass, thus initialising the variables of the superclass
		// As this is a parameterised constructor, the type of the object is parsed into the superclass initializer
		// So that the type of the object is known from initialisation
		super.init(typeInp: GraphObjectsClass.penDrawObjectID)
	}
}
```

CurvedLineObject

```
//
//  curvedLineObject.swift
//  test3
//
//

import Foundation

// The extension keyword implies that this code body extends the functionality of the CurvedLineObject class
// In this scenario, it is adding functions that allow the CurvedLineObject class to conform to Equatable through overloading the equality and inequality operators, which it must as CurvedLineObject inherits this from its superclass, GraphObject
extension CurvedLineObject {
	// Operator overloading the equality operator so that CurvedLineObjects can be compared using the equality operator
	// "left" represents the object on the left side of the operator, and "right" represents the object on the right side of the operator
	// "-> Bool" means that this function will return a boolean variable
	static func ==(left : CurvedLineObject, right : CurvedLineObject) -> Bool {
		return (
			left.height == right.height &&
			left.width == right.width &&
			left.objectType == right.objectType &&
			left.parameters == right.parameters &&
			left.useAdvanced == right.useAdvanced &&
			left.bezierControlPoints == right.bezierControlPoints &&
			left.isCubic == right.isCubic
```

```
            )
        }

        // Operator overloading the inequality operator so that CurvedLineObjects can be
compared using the equality operator
        static func !=(left : CurvedLineObject, right : CurvedLineObject) -> Bool {
            return (
                left.height != right.height ||
                left.width != right.width ||
                left.objectType != right.objectType ||
                left.parameters != right.parameters ||
                left.useAdvanced != right.useAdvanced ||
                left.bezierControlPoints != right.bezierControlPoints ||
                left.isCubic != right.isCubic
            )
        }
}

// A class that describes objects representing curved lines
class CurvedLineObject : GraphObject {
        // An array of bezier curve control points
        var bezierControlPoints : [CGPoint]
        // A variable to state whether the bezier curve is a cubic (thus true) or a quadratic
(thus false)
        var isCubic : Bool

        init() {
            bezierControlPoints = [CGPoint]()
            isCubic = false
            super.init(typeInp: GraphObjectsClass.curvedLineObjectID)
        }
}
```

StraightLineObject

```
//
//  straightLineObject.swift
//  test3
//
//

import Foundation

// The extension keyword implies that this code body extends the functionality of the
StraightLineObject class
```

```
// In this scenario, it is adding functions that allow the StraightLineObject class to conform to Equatable through overloading the equality and inequality operators, which it must as StraightLineObject inherits this from its superclass, GraphObject
extension StraightLineObject {
	// Operator overloading the equality operator so that StraightLineObjects can be compared using the equality operator
	// "left" represents the object on the left side of the operator, and "right" represents the object on the right side of the operator
	// "-> Bool" means that this function will return a boolean variable
	static func ==(left : StraightLineObject, right : StraightLineObject) -> Bool {
		return (
			left.height == right.height &&
			left.width == right.width &&
			left.objectType == right.objectType &&
			left.parameters == right.parameters &&
			left.useAdvanced == right.useAdvanced &&
			left.startCoord == right.startCoord &&
			left.endCoord == right.endCoord &&
			left.equation == right.equation &&
			left.gradient == right.gradient &&
			left.yIntercept == right.yIntercept
		)
	}

	// Operator overloading the inequality operator so that StraightLineObjects can be compared using the equality operator
	static func !=(left : StraightLineObject, right : StraightLineObject) -> Bool {
		return (
			left.height != right.height ||
			left.width != right.width ||
			left.objectType != right.objectType ||
			left.parameters != right.parameters ||
			left.useAdvanced != right.useAdvanced ||
			left.startCoord != right.startCoord ||
			left.endCoord != right.endCoord ||
			left.equation != right.equation ||
			left.gradient != right.gradient ||
			left.yIntercept != right.yIntercept
		)
	}
}

// A class that describes objects representing straight lines
class StraightLineObject : GraphObject {
	var startCoord : CGPoint
	var endCoord : CGPoint
	var equation : String
```

```
        // A computed property, effectively a closure that can return values based on other variables during the point of call and be written data
        var gradient : CGFloat {
                // Runs when the variable is used
                get {
                        if (self.startCoord != CGPoint(x: -1, y: -1) && self.endCoord != CGPoint(x: -1, y: -1) && (self.startCoord.x - self.endCoord.x) != 0) {
                                return (self.startCoord.y - self.endCoord.y) / (self.startCoord.x - self.endCoord.x)
                        } else { return CGFloat.infinity }
                // Sets the value of the property
                } set(newGradient) {
                        self.gradient = newGradient
                }
        }
        // Another computed property
        var yIntercept : CGFloat {
                get {
                        self.startCoord.y - (self.gradient * self.startCoord.x)
                } set (newYIntercept) {
                        self.yIntercept = newYIntercept
                }
        }

        init() {
                // CGPoint(x: -1, y: -1) is written as the user cannot select those points when using the application
                startCoord = CGPoint(x: -1, y: -1)
                endCoord = CGPoint(x: -1, y: -1)
                equation = ""
                super.init(typeInp: GraphObjectsClass.straightLineObjectID)
//              yIntercept = CGFloat.infinity
//              gradient = 0
        }
}
```

TextObject

```
//
//  textObject.swift
//  test3
//
//

import Foundation
import AppKit
```

```
// The extension keyword implies that this code body extends the functionality of the TextObject class
// In this scenario, it is adding functions that allow the TextObject class to conform to Equatable through overloading the equality and inequality operators, which it must as TextObject inherits this from its superclass, GraphObject
extension TextObject {
	// Operator overloading the equality operator so that TextObjects can be compared using the equality operator
	// "left" represents the object on the left side of the operator, and "right" represents the object on the right side of the operator
	// "-> Bool" means that this function will return a boolean variable
	static func ==(left : TextObject, right : TextObject) -> Bool {
		return (
			left.height == right.height &&
			left.width == right.width &&
			left.objectType == right.objectType &&
			left.parameters == right.parameters &&
			left.useAdvanced == right.useAdvanced &&
			left.textView == right.textView &&
			left.drawLocation == right.drawLocation
		)
	}

	// Operator overloading the inequality operator so that TextObjects can be compared using the equality operator
	static func !=(left : TextObject, right : TextObject) -> Bool {
		return (
			left.height != right.height ||
			left.width != right.width ||
			left.objectType != right.objectType ||
			left.parameters != right.parameters ||
			left.useAdvanced != right.useAdvanced ||
			left.textView != right.textView ||
			left.drawLocation != right.drawLocation
		)
	}
}

// A class that describes objects representing text objects
class TextObject : GraphObject {
	// NSTextView is a class that describes a text box object
	// It was used as it has predefined methods for drawing
	var textView : NSTextView
	var drawLocation : CGPoint

	init() {
		textView = NSTextView.init()
		drawLocation = CGPoint(x: -1, y: -1)
```

```
                super.init(typeInp: GraphObjectsClass.textObjectID)
        }
}
```

GraphObject

```
//
//  GraphObject.swift
//  test3
//
//

import Foundation

// The extension keyword implies that this code body extends the functionality of the GraphObject class
// This extension states that GraphObjects conforms to the Equatable protocol, allowing it to use the equality and inequality operator. This protocol inheritance is passed down to subclasses of GraphObject, forcing them to conform to the Equatable protocol as well
extension GraphObject : Equatable {
        // Operator overloading the equality operator so that GraphObjects can be compared using the equality operator
        // "left" represents the object on the left side of the operator, and "right" represents the object on the right side of the operator
        // "-> Bool" means that this function will return a boolean variable
        static func ==(left : GraphObject, right : GraphObject) -> Bool {
                // These if statements are meant to call equality functions that do further equality checks on specific attributes of two given graph objects, which is important when the graph objects being compared have not been downcasted to their specific graph types
                if (left.objectType == right.objectType && left.objectType == GraphObjectsClass.penDrawObjectID) {
                        // As the type of the two graph objects used with the equality operator are the same, they are both force downcasted to the same graph object class so that further equality check can occur with the specific variables found within the different graph object, as operator overloading has occurred for those graph object types as well, and by downcasting both of the objects to the same graph object class, the operator operations associated with the equality operator for that specific graph object can occur
                        return ((left as! PenDrawObject) == (right as! PenDrawObject))

                } else if (left.objectType == right.objectType && left.objectType == GraphObjectsClass.straightLineObjectID) {
                        return ((left as! StraightLineObject) == (right as! StraightLineObject))

                } else if (left.objectType == right.objectType && left.objectType == GraphObjectsClass.curvedLineObjectID) {
                        return ((left as! CurvedLineObject) == (right as! CurvedLineObject))
```

```
            } else if (left.objectType == right.objectType && left.objectType ==
GraphObjectsClass.textObjectID) {
                return ((left as! TextObject) == (right as! TextObject))

            } else {
                return (
                    left.height == right.height &&
                    left.width == right.width &&
                    left.objectType == right.objectType &&
                    left.parameters == right.parameters &&
                    left.useAdvanced == right.useAdvanced
                )
            }
        }

        // Operator overloading the inequality operator so that GraphObjects can be
compared using the equality operator
        static func !=(left : GraphObject, right : GraphObject) -> Bool {
            // These if statements are meant to call inequality functions that do further
inequality checks on specific attributes of two given graph objects, which is important when
the graph objects being compared have not been downcasted to their specific graph types
            if (left.objectType == right.objectType && left.objectType ==
GraphObjectsClass.penDrawObjectID) {
                // As the type of the two graph objects used with the inequality
operator are the same, they are both force downcasted to the same graph object class so
that further inequality check can occur with the specific variables found within the different
graph object, as operator overloading has occurred for those graph object types as well, and
by downcasting both of the objects to the same graph object class, the operator operations
associated with the inequality operator for that specific graph object can occur
                return ((left as! PenDrawObject) != (right as! PenDrawObject))

            } else if (left.objectType == right.objectType && left.objectType ==
GraphObjectsClass.straightLineObjectID) {
                return ((left as! StraightLineObject) != (right as! StraightLineObject))

            } else if (left.objectType == right.objectType && left.objectType ==
GraphObjectsClass.curvedLineObjectID) {
                return ((left as! CurvedLineObject) != (right as! CurvedLineObject))

            } else if (left.objectType == right.objectType && left.objectType ==
GraphObjectsClass.textObjectID) {
                return ((left as! TextObject) != (right as! TextObject))

            } else {
                return (
                    left.height != right.height ||
                    left.width != right.width ||
```

```
                                left.objectType != right.objectType ||
                                left.parameters != right.parameters ||
                                left.useAdvanced != right.useAdvanced
                        )
                }
        }
}

// A class that serves as a blueprint for graph objects that are to be drawn
class GraphObject {
        // CGFloat is a Core Graphics Float that changes in byte size based on whether the code runs on a 64 bit device or a 32 bit device. It is a legacy data type that is used so that during runtime, when used by Core Graphics functions, the variables do not need to be converted to CGFloat, unlike other floating-point data types
        // This is because CGFloat does not support toll-free bridging, meaning that it is not a typealias or identifier for a different data type, and as such when another data type is substituted when CGFloat is expected, conversion from that floating-point data type to CGFloat must occur
        var height : CGFloat
        var width : CGFloat
        var objectType : String
        // A dictionary of parameters, where key values are Strings, and value values can be any data type or object, so long as the type or object conforms to the Hashable protocol, which is required in order to use Swift dictionaries, which internally are akin to Java HashMaps
        // The postfix "Any" implies that any data type can be used as a value in a key-value pair, so long as it is hashable
        var parameters : [String : AnyHashable]
        var useAdvanced : Bool

        // As all graph objects are to have a type associated with it, a non-parameterised constructor is the only constructor present
        // If a placeholder object is needed, the type is set to ""
        // The left side of the colon is the label that must be used when the function is called, to make it explicitly clear to programmers that they are parsing in the type of the graph object
        // The right side of the colon is the data type that must be parsed
        init(typeInp : String) {
                height = 0;
                width = 0;
                objectType = typeInp
                parameters = [String : AnyHashable]()
                useAdvanced = false
        }
}
```

GraphObjectsClass

```
//
//  GraphObjectsClass.swift
//  test3
//
//

import Foundation


class GraphObjectsClass {
	// A group of static constants that contain Strings, serving as a wrapper for String literals
	static let penDrawObjectID : String = "penDrawObject"
	static let straightLineObjectID : String = "straightLineObject"
	static let curvedLineObjectID : String = "curvedLineObject"
	static let textObjectID : String = "textObject"

}
```

ViewController

```
//
//  ViewController.swift
//  Economic Graph Maker Version 2
//
//

import Cocoa

// The view controller of the welcome screen
// As the usage of storyboard connectivity has been used, this file is empty, and exists mainly as a file for the welcome screen (which is contained under Main.storyboard) to point at.
class ViewController: NSViewController {

	override func viewDidLoad() {
		super.viewDidLoad()
	}

}
```

DrawingMouseDown

```
//
//  drawingMouseDown.swift
//  test3
//
//

import Foundation
import Cocoa

// The extension keyword implies that this code body extends the functionality of the GraphEnvironmentViewController class
// In this scenario, it is extending the amount of functions that the graph view has within a seperate file for organisation purposes
// This file only contains functions that pertain to how the program should react when the user does a primary click when having chosen their given action to do
extension GraphEnvironmentViewController {
	// A function that is called when the user does a primary click when it has selected to do pen drawing
	func penDrawMouseDown() {
		// Ensures that the current graph object that is being operated on is a pen draw object, and that the program is currently not expecting to draw anything
		// If this is true, then "shouldDraw "is set to true to imply that the view should expect to draw, and "needsDisplay" is set to true to tell the graph view to call its draw function, thus refreshing the view as often as possible, in order to create the illusion of simultaneous drawing
		if (graphView.currentGraphObject.objectType == GraphObjectsClass.penDrawObjectID && self.shouldDraw == false) {
			self.shouldDraw = true
			graphView.needsDisplay = true
		// This else body exists so that when the user clicks whilst the program is expect to draw, it will stop drawing functionality
		} else {
			self.shouldDraw = false
			graphView.needsDisplay = false
		}
	}

	// A function that is called when the user does a primary click when it has selected to do create a straight line
	func straightLineMouseDown() {
		// Checks to see if the view is currently expecting to draw, and if it is not, the if body is executed to tell the view to expect to draw and to refresh the view
		if (self.shouldDraw == false) {
			self.shouldDraw = true
			graphView.needsDisplay = true
		}
```

```
                // Ensures that the current graph object that is being operated on is a straight line object
                if (graphView.currentGraphObject.objectType == GraphObjectsClass.straightLineObjectID) {

                    // CGPoint(x: -1, y: -1) is a point that users cannot select. This variable exists to check whether the variable has just been initialised
                    if ((graphView.currentGraphObject as! StraightLineObject).startCoord.equalTo(CGPoint(x: -1, y: -1)) && (graphView.currentGraphObject as! StraightLineObject).endCoord.equalTo(CGPoint(x: -1, y: -1))) {

                        // A check to see if the mouse location variable has been initialised
                        // "mouseLocationOutsideOfEventStream" is an inherited variable, that returns the mouse location with respect to the coordinate system of the window's primary view, which itself contains "graphView"
                        if (graphView.window?.mouseLocationOutsideOfEventStream != nil) {
                            // As graph objects are drawn within "graphView"'s rectangle, with respect to "graphView"'s coordinate system, the mouse location that is taken with respect to the coordinate system of the window's primary view, is converted to coordinates with respect to the coordinate system of the target view, which in this case is "graphView"'s rectangle
                            // As "graphView.window?.mouseLocationOutsideOfEventStream" may be nil, the nil coalescing operator is used so that if there exists no mouse location recorded, despite a mouse click, then the literal "CGPoint(x: 0, y: 0)" is used so that the program doesn't crash when attempting to operate on a nil value
                            let pointInTargetView = self.graphView.convert(graphView.window?.mouseLocationOutsideOfEventStream ?? CGPoint(x: 0, y: 0), from: self.view)

                            // If the variable has just been initialised, as both the starting coordinate and the ending coordinate are equal to CGPoint(x: -1, y: -1), and has thus entered the the current code body, then the start coordinate is now assigned to be the value of the current cursor location that was click on with respect to the coordinate system of "graphView"'s rectangle
                            (graphView.GraphObjects.last as! StraightLineObject).startCoord = pointInTargetView
                        }

                    // If one of the coordinates has been assigned to a coordinate that can be selected by the user, this means that the next click will determine the position of the next coordinate, meaning that all the coordinate data will have been provided to draw
                    } else if (!(graphView.currentGraphObject as! StraightLineObject).startCoord.equalTo(CGPoint(x: -1, y: -1)) &&
```

```
(graphView.currentGraphObject as! StraightLineObject).endCoord.equalTo(CGPoint(x: -1, y: -1))) {

                                    if (graphView.window?.mouseLocationOutsideOfEventStream != nil) {
                                        let pointInTargetView = self.graphView.convert(graphView.window?.mouseLocationOutsideOfEventStream ?? CGPoint(x: 0, y: 0), from: self.view)

                                        // As this code body is entered if the starting coordinate is initialised, but the ending coordinate is not, the ending coordinate is now assigned a coordinate that exists within the bounds of "graphView"
                                        (graphView.GraphObjects.last as! StraightLineObject).endCoord = pointInTargetView
                                        // This function tells "graphView" that it must now redraw its view, ensuring that the line will be drawn and seen by the user
                                        graphView.viewWillDraw()


                                    }

                                    // Tells the program to stop expecting to draw, and "graphView" to stop refreshing its view
                                    self.shouldDraw = false
                                    graphView.needsDisplay = false

                                // If one of the coordinates has been assigned to a coordinate that can be selected by the user, this means that the next click will determine the position of the next coordinate, meaning that all the coordinate data will have been provided to draw
                                } else if ((graphView.currentGraphObject as! StraightLineObject).startCoord.equalTo(CGPoint(x: -1, y: -1)) && !(graphView.currentGraphObject as! StraightLineObject).endCoord.equalTo(CGPoint(x: -1, y: -1))) {

                                    if (graphView.window?.mouseLocationOutsideOfEventStream != nil) {
                                        let pointInTargetView = self.graphView.convert(graphView.window?.mouseLocationOutsideOfEventStream ?? CGPoint(x: 0, y: 0), from: self.view)

                                        (graphView.GraphObjects.last as! StraightLineObject).startCoord = pointInTargetView
                                        graphView.viewWillDraw()
                                    }

                                    // Tells the program to stop expecting to draw, and "graphView" to stop refreshing its view
                                    self.shouldDraw = false
```

```
                        graphView.needsDisplay = false

                    } else {
                        // Tells the program to stop expecting to draw, and "graphView"
to stop refreshing its view
                        self.shouldDraw = false
                        graphView.needsDisplay = false
                    }
                }
            }

            // A function that is called when the user does a primary click when it has selected to
create a curved line
            func curvedLineMouseDown() {
                if (self.shouldDraw == false) {
                    self.shouldDraw = true
                    graphView.needsDisplay = true
                }

                if (graphView.currentGraphObject.objectType ==
GraphObjectsClass.curvedLineObjectID) {
                    if ((graphView.currentGraphObject as! CurvedLineObject).isCubic) {
                        if ((graphView.currentGraphObject as!
CurvedLineObject).bezierControlPoints.count < 4) {

                            if
(graphView.window?.mouseLocationOutsideOfEventStream != nil) {
                                let pointInTargetView =
self.graphView.convert(graphView.window?.mouseLocationOutsideOfEventStream ??
CGPoint(x: 0, y: 0), from: self.view)

                                (graphView.currentGraphObject as!
CurvedLineObject).bezierControlPoints.append(pointInTargetView)
                                graphView.viewWillDraw()
                            }
                        }

                        if ((graphView.currentGraphObject as!
CurvedLineObject).bezierControlPoints.count == 4) {
                            graphView.viewWillDraw()
                            // Tells the program to stop expecting to draw, and
"graphView" to stop refreshing its view
                            self.shouldDraw = false
                            graphView.needsDisplay = false
                        }

                    } else {
```

```
                                if ((graphView.currentGraphObject as!
CurvedLineObject).bezierControlPoints.count < 3) {

                                        if
(graphView.window?.mouseLocationOutsideOfEventStream != nil) {
                                                let pointInTargetView =
self.graphView.convert(graphView.window?.mouseLocationOutsideOfEventStream ??
CGPoint(x: 0, y: 0), from: self.view)

                                                (graphView.currentGraphObject as!
CurvedLineObject).bezierControlPoints.append(pointInTargetView)

                                                graphView.viewWillDraw()
                                        }
                                }

                                if ((graphView.currentGraphObject as!
CurvedLineObject).bezierControlPoints.count == 3) {
                                        graphView.viewWillDraw()
                                        // Tells the program to stop expecting to draw, and
"graphView" to stop refreshing its view
                                        self.shouldDraw = false
                                        graphView.needsDisplay = false
                                }
                        }
                }
        }

        // A function that is called when the user does a primary click when it has selected to
create a text box
        func textObjectMouseDown() {
                // As graph objects are drawn within "graphView"'s rectangle, with respect to
"graphView"'s coordinate system, the mouse location that is taken with respect to the
coordinate system of the window's primary view, is converted to coordinates with respect to
the coordinate system of the target view, which in this case is "graphView"'s rectangle
                // As "graphView.window?.mouseLocationOutsideOfEventStream" may be nil,
the nil coalescing operator is used so that if there exists no mouse location recorded, despite
a mouse click, then the literal "CGPoint(x: 0, y: 0)" is used so that the program doesn't crash
when attempting to operate on a nil value
                let pointInTargetView =
self.graphView.convert(self.view.window?.mouseLocationOutsideOfEventStream ??
CGPoint(x: 0, y: 0), from: self.view)

                // Creates a empty text object
                let textObject : TextObject = TextObject.init()

                // Assigns the text object with a text box, with a height that acts as a rectangle
to click to start editing
```

```
                textObject.textView = NSTextView.init(frame: NSRect.init(origin:
pointInTargetView, size: CGSize.init(width: 10, height: 20)))

                textObject.drawLocation = pointInTargetView

                // Tells the text box that it can draw its rectangle asynchronously to its
superview via a thread that handles drawing
                textObject.textView.canDrawConcurrently = true



                // Sets the background colour to clear
                textObject.textView.backgroundColor = .clear
                textObject.textView.textColor = .black
                // Assigns the delegate of this text box to be
"GraphEnvironmentViewController", which will now handle changes and the user interactions
that occur on the text box
                textObject.textView.delegate = self
                // As the text box will be resized manually, the text box is told that it will not
automatically resize itself horizontally or vertically
                textObject.textView.isHorizontallyResizable = false
                textObject.textView.isVerticallyResizable = false

                // The container size is the maximum size that the text view can grow to,
which in this case is this case is the size of the graph view
                textObject.textView.textContainer!.containerSize = CGSize(width:
self.graphView.bounds.width, height: self.graphView.bounds.height)
                // As the text box will be resized manually, the text box is told to not track for
width changes and change accordingly
                textObject.textView.textContainer!.widthTracksTextView = false

                // The text object is now added to the array of graph objects
                graphView.GraphObjects.append(textObject as GraphObject)
                // The text box is now added to the graph view as a subview, meaning that the
graph view will now be displayed on the graph view
                self.graphView.addSubview(textObject.textView)

        }

        // A function that is called when the user does a primary click when it has selected to
do delete a graph object
        func deleteMouseDown() {
                // As graph objects are drawn within "graphView"'s rectangle, with respect to
"graphView"'s coordinate system, the mouse location that is taken with respect to the
coordinate system of the window's primary view, is converted to coordinates with respect to
the coordinate system of the target view, which in this case is "graphView"'s rectangle
                // As "graphView.window?.mouseLocationOutsideOfEventStream" may be nil,
the nil coalescing operator is used so that if there exists no mouse location recorded, despite
```

```
a mouse click, then the literal "CGPoint(x: 0, y: 0)" is used so that the program doesn't crash when attempting to operate on a nil value
                let pointInTargetView = self.graphView.convert(graphView.window?.mouseLocationOutsideOfEventStream ?? CGPoint(x: 0, y: 0), from: self.view)

                // Iterates through the graph objects within the "GraphObjects" array in reverse, looking at the most recent objects first, so that if multiple objects are clicked, only the most recent object is deleted
                // A break label is created so that once the most recently placed object that intersects with the user click is is deleted, the loop is exited
                objectDeleted: for (GraphObject) in (graphView.GraphObjects.reversed()) {
                        // Determines the type of the object, in order to utilise the correct detection method
                        switch (objectIDToActionInt[GraphObject.objectType]) {
                                // Runs if the object is a pen draw object
                                case (0):
                                        // Creates a mutable CGPath object that is effectively an array of points with specialised functions
                                        let tempVar : CGMutablePath = CGMutablePath.init()
                                        // Tells the path object to add lines between the points of the pen draw curve
                                        tempVar.addLines(between: (GraphObject as! PenDrawObject).penDrawPoints)
                                        // An if statement that checks to see if the point of the user click intersects with the pen draw curve
                                        if (tempVar.contains(pointInTargetView)) {
                                                // Filters out the graph object to be deleted, and then returns the rest of the graph objects in a new array that has been automatically resized with all elements shifted
                                                graphView.GraphObjects = graphView.GraphObjects.filter{ $0 != GraphObject }
                                                // A break statement that exists the loop so that only one object is deleted
                                                break objectDeleted
                                        }
                                        // A break statement that exists the case's code body
                                        break
                                // Runs if the object is a straight line object
                                case (1):
                                        // If the straight line object has not been given coordinates by the user, then the program does not attempt to detect if intersection has occurred
                                        if ((GraphObject as! StraightLineObject).startCoord != CGPoint(x: -1, y: -1) && (GraphObject as! StraightLineObject).endCoord != CGPoint(x: -1, y: -1)) {
```

```
                                                    // An array of points, which will then be checked against the point of where the user clicked to see if intersection has occurred
                                                    var pointsArray : [CGPoint] = [CGPoint]()

                                                    // If the starting coordinate has an x-coordinate that is less than the ending coordinate, the loop will iterate from the starting coordinate to the ending coordinate, and if the starting coordinate has an x-coordinate that is greater than the ending coordinate, the loop will iterate from the ending coordinate to the starting coordinate
                                                    if ((GraphObject as! StraightLineObject).startCoord.x < (GraphObject as! StraightLineObject).endCoord.x) {

                                                        // stride() is a function that returns a sequence from the lower bound to the upper bound, with increments from the lower bound to the upper bound being based on the incremental value
                                                        for xValue in stride(from: (GraphObject as! StraightLineObject).startCoord.x, to: (GraphObject as! StraightLineObject).endCoord.x, by: 0.1) {
                                                            // As the graph object is a straight line, the y-coordinate can thus be calculated via the equation "y = mx + c", where m is the gradient, and c is the y-intercept
                                                            // Thus, the points that will be utilised for detecting if the point of the user click intersects with the straight line object will be calculated based on the equation above
                                                            pointsArray.append(CGPoint(x: xValue, y: ((GraphObject as! StraightLineObject).gradient * xValue) + (GraphObject as! StraightLineObject).yIntercept))
                                                        }
                                                    } else {
                                                        // As the starting coordinate has an x-coordinate that is greater than the ending coordinate, the sequence created from the stride function will be created in reverse
                                                        for xValue in stride(from: (GraphObject as! StraightLineObject).startCoord.x, to: (GraphObject as! StraightLineObject).endCoord.x, by: -0.1) {

                                                            pointsArray.append(CGPoint(x: xValue, y: ((GraphObject as! StraightLineObject).gradient * xValue) + (GraphObject as! StraightLineObject).yIntercept))
                                                        }
                                                    }

                                                    // Iterate through the points array to extract the individual points
                                                    for point in pointsArray {
                                                        // Creates a square at a given point
                                                        let tempRect : CGRect = CGRect(origin: point, size: CGSize.init(width: 4, height: 4))
```

```
                                                    // Translates the square to the centre of
the point of the line
                                                    // This works as a square has the same
height and width, thus the middle coordinate of the square is half the height and width of the
square, which then can be subtracted or added from the point of origin created, to then
translate the square to the centre of the point on the line
                                                    let intersectionRect : CGRect =
tempRect.offsetBy(dx: -(tempRect.midX - point.x), dy: (point.y - tempRect.midY))

                                                    // If the intersection rectangle intersects
with the point of the cursor click, then the user has chosen to delete the most recently
created object that intersects with the point of the user's cursor click
                                                    if
(intersectionRect.contains(pointInTargetView)) {
                                                        // Filters out the graph object to
be deleted, and then returns the rest of the graph objects in a new array that has been
automatically resized with all elements shifted
                                                        graphView.GraphObjects =
graphView.GraphObjects.filter{ $0 != GraphObject }
                                                        // A break statement that exists
the loop so that only one object is deleted
                                                        break objectDeleted
                                                    }
                                                }
                                            }
                                            // A break statement that exists the case's code body
                                            break
                                        // Runs if the object is a curved line object
                                        case (2):
                                            // This code operates on similar logic to the pen draw
objects, through creating a path objects and then utilising predefined functions for
intersection detection
                                            let tempVar : CGMutablePath = CGMutablePath.init()
                                            // If "bezierControlPoints.count == 3", this means that
the curve is a quadratic bezier curve
                                            if ((GraphObject as!
CurvedLineObject).bezierControlPoints.count == 3) {
                                                tempVar.move(to: (GraphObject as!
CurvedLineObject).bezierControlPoints[0])
                                                tempVar.addQuadCurve(to: (GraphObject as!
CurvedLineObject).bezierControlPoints[2], control: (GraphObject as!
CurvedLineObject).bezierControlPoints[1])
                                            // If "bezierControlPoints.count == 4", this means that
the curve is a cubic bezier curve
                                            } else if ((GraphObject as!
CurvedLineObject).bezierControlPoints.count == 4) {
                                                tempVar.move(to: (GraphObject as!
CurvedLineObject).bezierControlPoints[0])
```

```
                                                tempVar.addCurve(to: (GraphObject as!
CurvedLineObject).bezierControlPoints[3], control1: (GraphObject as!
CurvedLineObject).bezierControlPoints[1], control2: (GraphObject as!
CurvedLineObject).bezierControlPoints[2])
                                        }
                                        // If "bezierControlPoints.count" does not equal 3 or 4,
this means that the curve has not been properly created, and is thus ignored
                                        if (tempVar.contains(pointInTargetView)) {
                                                // Filters out the graph object to be deleted, and
then returns the rest of the graph objects in a new array that has been automatically resized
with all
                                                graphView.GraphObjects =
graphView.GraphObjects.filter{ $0 != GraphObject }
                                                // A break statement that exists the loop so that
only one object is deleted
                                                break objectDeleted
                                        }
                                        // A break statement that exists the case's code body
                                        break
                                // Runs if the object is a text object
                                case (3):
                                        // As the text object exists within the rectangle of
"graphView", frame is used so that the text box's area and location is taken with respect to
the coordinate system of "graphView"'s rectangle
                                        if ((GraphObject as!
TextObject).textView.frame.contains(pointInTargetView)) {
                                                // Filters out the graph object to be deleted, and
then returns the rest of the graph objects in a new array that has been automatically resized
with all
                                                graphView.GraphObjects =
graphView.GraphObjects.filter{ $0 != GraphObject }
                                                // A break statement that exists the loop so that
only one object is deleted
                                                break objectDeleted
                                        }
                                        // Runs if the object is a text object
                                        break
                                default:
                                        // This should never be reached, as only four types of
graph objects with four different IDs should be populated within the array
                                        break
                        }
                }

                // Checks to see if the view is currently expecting to draw, and if it is not, the if
body is executed to tell the view to expect to draw and to refresh the view
                if (self.shouldDraw == false) {
                        self.shouldDraw = true
```

```
                        graphView.needsDisplay = true
                }
                // Forces "graphView" to redraw its view
                graphView.viewWillDraw()
                // Tells the program to stop expecting to draw, and "graphView" to stop refreshing its view
                self.shouldDraw = false
                graphView.needsDisplay = false

                // Sets "selectedGraphObject" to an empty, generic graph object so that a new copy of the "graphView" rectangle is sent to the "GraphToolbar" object associated with the current view controller, so that the object is ready to create an image file when asked
                graphView.selectedGraphObject = GraphObject.init(typeInp: "")
        }
}
```

UserInteraction

```
//
//  userInteraction.swift
//  test3
//
//

import Foundation
import Cocoa

extension GraphEnvironmentViewController {
        // Overrides the mouseDown function that is called when the "trackingArea" variable that belongs to "GraphEnvironmentViewController" detects a mouse click within its tracking area
        override func mouseDown(with event: NSEvent) {
                super.mouseDown(with: event)

                // As graph objects are drawn within "graphView"'s rectangle, with respect to "graphView"'s coordinate system, the mouse location that is taken with respect to the coordinate system of the window's primary view, is converted to coordinates with respect to the coordinate system of the target view, which in this case is "graphView"'s rectangle
                // As "graphView.window?.mouseLocationOutsideOfEventStream" may be nil, the nil coalescing operator is used so that if there exists no mouse location recorded, despite a mouse click, then the literal "CGPoint(x: 0, y: 0)" is used so that the program doesn't crash when attempting to operate on a nil value
                let pointInTargetView = self.graphView.convert(graphView.window?.mouseLocationOutsideOfEventStream ?? CGPoint(x: 0, y: 0), from: self.view)
```

```
                // If the mouse click has occurred within the graph view, then the statement is
true
                if (graphView.bounds.contains(pointInTargetView)) {

                    // If "actionInt" is equal to 7, this means that a delete operation has
been attempted, which means that it does not matter if the click intersects with any text
boxes
                    // If the click is contained within a text box, and the user is attempting
to draw something, the program stops that from happening, and allows the user to instead
edit the text box
                    // This is done by first creating an array that filters out only text
objects, and then another function that checks to ensure that all text objects satisfy the
condition that their text box does not contain the point that the user has clicked on
                    if (GraphEnvironmentViewController.actionInt == 7 ||
graphView.GraphObjects
                            .filter { $0.objectType ==
GraphObjectsClass.textObjectID }
                            .allSatisfy{ !($0 as!
TextObject).textView.frame.contains(pointInTargetView) }
                    ) {

                        // Removes focus from the text boxes, stopping editing, and
bringing focus to the "graphView" again
                        // This is achieved by removing first responder responsibilities
from the text view, which is the text box, meaning that user interactions are sent to
"graphView" instead of the text objects
                        (graphView.GraphObjects.filter{ $0.objectType ==
GraphObjectsClass.textObjectID } as! [TextObject]).forEach{
$0.textView.window?.makeFirstResponder(nil) }

                        // If the "GraphObjects" is empty, or if the object last inserted is
not the same as the new object that the user is trying to create, then the condition below is
true
                        if (graphView.GraphObjects.isEmpty ||
(objectIDToActionInt[graphView.GraphObjects.last!.objectType] !=
GraphEnvironmentViewController.actionInt)) {
                            // The switch structure ensures that the function
branches off to the correct code body to execute based on what the user is trying to do
                            switch (GraphEnvironmentViewController.actionInt) {
                                case (0):
                                    // Adds a pen draw object to the
"GraphObjects" array

graphView.GraphObjects.append(PenDrawObject.init())
                                    // Tells the "graphView" object that the
object to now take action on is the most recently added graph object
                                    graphView.currentGraphObject =
graphView.GraphObjects.last!
```

```
                                                    // Calls a function to handle pen drawing
                                                    penDrawMouseDown()
                                                    // States that the new selected graph
object is the current graph object, so that "graphView" sends a new copy of its view to the
associated "GraphToolbar" object for the view controller for saving when needed
                                                    graphView.selectedGraphObject =
graphView.currentGraphObject
                                                    break
                                                case (1):
                                                    // Adds a straight line object to the
"GraphObjects" array

graphView.GraphObjects.append(StraightLineObject.init())
                                                    graphView.currentGraphObject =
graphView.GraphObjects.last!
                                                    // Calls a function to handle the creation
of a straight line object
                                                    straightLineMouseDown()
                                                    graphView.selectedGraphObject =
graphView.currentGraphObject
                                                    break
                                                case (2):
                                                    // Adds a curved line object to the
"GraphObjects" array

graphView.GraphObjects.append(CurvedLineObject.init())
                                                    graphView.currentGraphObject =
graphView.GraphObjects.last!
                                                    // Calls a function to handle the creation
of a curved line object
                                                    curvedLineMouseDown()
                                                    graphView.selectedGraphObject =
graphView.currentGraphObject
                                                    break
                                                case (3):
                                                    // Calls the text object creation function
                                                    textObjectMouseDown()
                                                    break
                                                case (7):
                                                    // Calls the object deletion function
                                                    deleteMouseDown()
                                                default: break
                                            }
                                        // The condition below is true if the previous graph object type
added correlates to the action integer associated with the graph object type
                                        } else if
(objectIDToActionInt[graphView.GraphObjects.last!.objectType] ==
GraphEnvironmentViewController.actionInt) {
```

```
                                    // The switch structure ensures that the function
branches off to the correct code body to execute based on what the user is trying to do
                                    switch (GraphEnvironmentViewController.actionInt) {
                                        case (0):
                                            // When the user clicks down after
initialising a previous pen draw object, this leads "shouldDraw" to be false, as the pen draw
object is now complete
                                            // Another click after that means that
another pen draw object should be added to the "GraphObjects" array and the process
should start again
                                            if (!shouldDraw) {

graphView.GraphObjects.append(PenDrawObject.init())
                                                graphView.currentGraphObject =
graphView.GraphObjects.last!
                                            }
                                            // Calls a function to handle pen drawing
                                            penDrawMouseDown()
                                            graphView.selectedGraphObject =
graphView.currentGraphObject
                                            break
                                        case (1):
                                            // When the user clicks down after
initialising a previous straight line object, this leads "shouldDraw" to be false, as the straight
line object is now complete
                                            // Another click after that means that
another straight line object should be added to the "GraphObjects" array and the process
should start again
                                            if (!shouldDraw) {

graphView.GraphObjects.append(StraightLineObject.init())
                                                graphView.currentGraphObject =
graphView.GraphObjects.last!
                                            }
                                            // Calls a function to handle the creation
of a straight line object
                                            straightLineMouseDown()
                                            graphView.selectedGraphObject =
graphView.currentGraphObject
                                            break
                                        case (2):
                                            // When the user finishes creating a
curved line object, this leads "shouldDraw" to be false, as the curved line object is now
complete
                                            // Another click after that means that
curved line object object should be added to the "GraphObjects" array and the process
should start again
                                            if (!shouldDraw) {
```

```
graphView.GraphObjects.append(CurvedLineObject.init())
                                                graphView.currentGraphObject =
graphView.GraphObjects.last!
                                                }
                                                // Calls a function to handle the creation
of a curved line object
                                                curvedLineMouseDown()
                                                graphView.selectedGraphObject =
graphView.currentGraphObject
                                                break
                                        case (3):
                                                // Calls the text object creation function
                                                textObjectMouseDown()
                                        default:
                                                break
                                }
                        }
                }
        } else {
                // Sets "selectedGraphObject" to an empty, generic graph object so
that a new copy of the "graphView" rectangle is sent to the "GraphToolbar" object associated
with the current view controller, so that the object is ready to create an image file when
asked
                graphView.selectedGraphObject = GraphObject.init(typeInp: "")
        }
}

// Overrides the mouseMoved function that is called when the "trackingArea" variable
that belongs to "GraphEnvironmentViewController" detects mouse movement within its
tracking area
override func mouseMoved(with event: NSEvent) {
        super.mouseMoved(with: event)

        // If the graph view is expecting drawing to occur, and the last graph objected
appended to "GraphObjects" is a pen draw object, then the code block inside of the if
statement below runs
        // A check to see if "GraphObjects" is empty occurs so that if the array is
empty, short-circuit evaluation will stop the program from then checking if the last element
added to an empty array contains a graph object that is a pen draw object, which is
impossible as the array is empty
        if (self.shouldDraw && !graphView.GraphObjects.isEmpty &&
graphView.GraphObjects.last?.objectType == GraphObjectsClass.penDrawObjectID) {


                // A check to see if the mouse location variable has been initialised
```

```
                        // "mouseLocationOutsideOfEventStream" is an inherited variable, that returns the mouse location with respect to the coordinate system of the window's primary view, which itself contains "graphView"
                        if (graphView.window?.mouseLocationOutsideOfEventStream != nil) {
                                // As graph objects are drawn within "graphView"'s rectangle, with respect to "graphView"'s coordinate system, the mouse location that is taken with respect to the coordinate system of the window's primary view, is converted to coordinates with respect to the coordinate system of the target view, which in this case is "graphView"'s rectangle
                                // As "graphView.window?.mouseLocationOutsideOfEventStream" may be nil, the nil coalescing operator is used so that if there exists no mouse location recorded, despite a mouse click, then the literal "CGPoint(x: 0, y: 0)" is used so that the program doesn't crash when attempting to operate on a nil value
                                let pointInTargetView = self.graphView.convert(graphView.window?.mouseLocationOutsideOfEventStream ?? CGPoint(x: 0, y: 0), from: self.view)

                                // Appends the current cursor location to the array of pen draw points, so that when the view redraws, the new point is included in the redraw
                                (graphView.GraphObjects.last as! PenDrawObject).penDrawPoints.append(pointInTargetView)

                        }
                }

                // If the action integer is equal to -1, this means that no drawing should be occurring, and if the action integer is equal to 3, this means that text box operations are occuring, which are asynchronous to the drawing of the "graphView" rectangle, and thus means that "graphView" does not need to redraw its rectangle
                // However if the action integer is equal to something else, then redrawing is required, and "graphView" should redraw its rectangle
                if (GraphView.actionInt != -1 || GraphView.actionInt != 3) {
                        // As redrawing is still required, "needsDisplay" is set to true
                        graphView.needsDisplay = true
                } else {
                        // As redrawing is not needed, "needsDisplay" is set to false
                        graphView.needsDisplay = false
                }
        }
}
```

4. Criterion C Terminology

Term	Description
AppKit	A framework that allows developers to create GUI components in macOS. It is included in Apple's Cocoa (see appendix definition). It utilises the NS prefix, which is derived from macOS' origin as an OS built on NextStep's OSes.
Closures	Closures can be thought of as functions, or bodies of code that are associated with variables. This allows for effectively arrays of functions. They may also be used as anonymous or lambda functions
Cocoa	Apple's Cocoa is a set of development and runtime tools for application development and running on macOS.
Computed Properties	A computed property is a variable that when read will compute a set of instructions, before returning the result of said instructions. Thus, the data of the variable is determined at and during the point of call. These properties are effectively closures associated with variables, but also have the ability to be written to.
Core Graphics	A low-level graphics handling library that operates below AppKit. It utilises the CG prefix to denote whether a class or function is from the core graphics library
Data Persistence	Apple's term for allowing data to be retained whether the application is opened or closed. This can be done through the data models that Apple provides, or through file handling, which is what the application created uses.
Delegation	Giving "parental responsibility" to other objects, allowing the delegate (the object that has been given responsibility) to observe and make changes to the instance it is responsible for.
Encoding and Decoding	Apple's terms for converting objects to and from textual representations respectively. The Swift method of encoding and decoding is done through protocol conformance (see appendix definition) to the Codable protocol, allowing for textual representations of graph objects.

Term	Description
Forced Downcasting	In Swift, forced downcasting is the concept of attempting to use an object as if it was a subclass of the superclass that the object is currently defined as. This is so that objects can access specific methods and attributes of the subclass that is being attempted to be downcasted to. Simply put, forced downcasting is a form of typecasting where an object is treated as if it was an object of a class that inherits the properties of its current class.
Functional Programming (FP)	FP is a programming paradigm whereby functions are given first class citizenship, meaning that they can be treated as variables and operated on as such. FP is used in my program through combining functions to extract or create desired and repeatable outcomes, particularly in array manipulation.
Graphical User Interface (GUI)	All GUI components except for the Economics Graph Editor View are implemented with AppKit components. Designing was done in Xcode's Interface Builder (IB), a drag and drop GUI builder, and then instantiated and manipulated with code using AppKit, a library that handles GUI creation and usage in MacOS.
IB Annotations	In order to connect the storyboard elements to code, IB Annotations allow for Xcode (the IDE used) to determine what variables are connected to the UI elements within the storyboard via the @IBOutlet annotation, and what interactive elements (such as buttons) via the @IBAction annotation. This is so UI elements can be further manipulated via code, and interactive elements know what function to call when the user interacts with them. Although the Swift terminology IB Attributes is the proper terminology, to assist developers more familiar with Java, the term annotations was used instead
Interface Builder (IB)	IB is one of Xcode's drag and drop GUI builders that create Storyboard files that describe GUI elements and the relationships between these items. IB GUI elements associate themselves with code via IB Annotations, allowing for them to be extended and further described via code.
Observers	Observation in Swift refers to the observation for

Term	Description
	changes of a variable. This is achieved through the didSet code structure, one of Swift's Key-Value Observation (KVO) structures.
Optionals	Many Swift data types may be optionals, meaning that they lack a value and hence are nil, Swift's equivalence of null. As nils cannot be used, optional handling must be done to ensure that when the program "unwraps" (revealing the true value of the optional) a nil value, the branch of execution is changed so that the program does not attempt to operate on nil values.
Protocol Oriented Programming (POP)	Protocol-oriented programming (POP) is a programming paradigm based on protocols that differ from classes in that protocols allow for the optional or mandatory inheritance of behaviours (such as the ability to be hashed by Swift's hash function, used internally for dictionaries) and functions by classes and data types from one or more protocols. Although delegation is an OOP concept, in Swift a class gains the properties required to delegate a type of object through inheriting delegation protocols.
Protocol Conformance	When a class or protocol in Swift has inherited a protocol, it is said that the class or protocol now conforms to the protocol it has inherited, thus gaining the functions and expected behaviours that originate from the protocol inherited. For example, classes that conform to the Equatable protocol allows for objects of the class or an object that conforms to a protocol that also conforms to Equatable to compare with each other via the equality operator (==), effectively overloading the equality operator in the recommended Swift way.
Selectors	Selectors are variables that can call functions, and can be thought of as a wrapper for function calls via a function pointer
Storyboard	A storyboard is a GUI file that represents GUI elements. Internally, they are XMLs that represent the relationships between GUI elements.
Swift	Swift is Apple's preferred language for macOS

Term	Description
	development. It is a multi-paradigm language, utilising elements of Object Oriented Programming, Protocol Oriented Programming, Functional Programming, and Procedural Programming.
Swift Standard Library	A library of data types, data structures, functions, protocols, classes, collections, Input/Output handling, and more
Xcode	Xcode is Apple's IDE, made exclusively for development on Apple platforms (such as iOS, macOS, and so on).

Sources: https://www.swift.org/, https://developer.apple.com/documentation/, https://developer.apple.com/library/archive/navigation/, http://www.hackingwithswift.com, https://www.raywenderlich.com/

[Words not included in word count]