Interactive Maths Graphing Tool

A-level project

Tyler carter-Bland

2018

Contents

[Analysis 3](#_Toc5969411)

[Description of the project 3](#_Toc5969412)

[Stakeholders 3](#_Toc5969413)

[Computational Methods 3](#_Toc5969414)

[Why is this problem suitable for computational methods? 3](#_Toc5969415)

[Abstraction 3](#_Toc5969416)

[Problem Decomposition 3](#_Toc5969417)

[Output 4](#_Toc5969418)

[Research 4](#_Toc5969419)

[Features of the proposed solution 5](#_Toc5969420)

[Software and Hardware Requirements 7](#_Toc5969421)

[Success Criteria 7](#_Toc5969422)

[Performance Criteria 8](#_Toc5969423)

[Design 9](#_Toc5969424)

[Decomposing the Problem 9](#_Toc5969425)

[Interactive Grid 9](#_Toc5969426)

[User Interface 9](#_Toc5969427)

[User Input/Output 10](#_Toc5969428)

[Defining Structure 11](#_Toc5969429)

[Algorithms 12](#_Toc5969430)

[Usability Features 14](#_Toc5969431)

[Navigation 14](#_Toc5969432)

[Ease of Use 14](#_Toc5969433)

[Error Prevention 15](#_Toc5969434)

[Variables, Data Structures, Validation 15](#_Toc5969435)

[Key Variables 15](#_Toc5969436)

[Data Structures 15](#_Toc5969437)

[Validation 16](#_Toc5969438)

[Iterative Development Test Data 16](#_Toc5969439)

[Post Development Test Data 17](#_Toc5969440)

[Development 20](#_Toc5969441)

[Development Evidence 20](#_Toc5969442)

[GUI 20](#_Toc5969443)

[Test Table 1 27](#_Toc5969444)

[Test Table 2 28](#_Toc5969445)

[Test Table 3 33](#_Toc5969446)

[Test Table 4 42](#_Toc5969447)

[Testing: Evaluative 50](#_Toc5969448)

[Final Testing Evidence 50](#_Toc5969449)

[Usability Testing 55](#_Toc5969450)

[Survey Analysis 58](#_Toc5969451)

[Evaluation 58](#_Toc5969452)

[Evaluating Success Criteria 58](#_Toc5969453)

[Evaluating Usability Features 61](#_Toc5969454)

[Limitations 61](#_Toc5969455)

[Maintenance 62](#_Toc5969456)

[Further Development 62](#_Toc5969457)

[Code Maintenance 62](#_Toc5969458)

[Appendix 62](#_Toc5969459)

[Code 62](#_Toc5969460)

# Analysis

## Description of the project

In mathematics, graphs are used often to visualise and solve problems in many different areas. It can be a lengthy process to plot a graph of an equation, especially if it is non-linear. My project would solve this, as it would allow the user to input an equation and instantly see the output on an interactive grid. It would provide many helpful features to assist the user, such as showing intersection points, stationary points, being able to view the gradient etc.

## Stakeholders

The stakeholders in this project will be maths students and teachers, who could use the application for an educational purpose to demonstrate certain graphs, and mathematicians who could use the application to help solve problems. Teachers would be able to use the application during a lesson, to demonstrate a graph or concept, and students should be able to use it on their home or school computers, without the use of the internet. The program will be suitable for Year 7-9 Maths, GCSE and A-Level Maths and Further Maths students. My solution will be appropriate for my stakeholders as it will be free, which is helpful for young students, it will be lightweight, so it will function well on a low-spec school computer, and it will not require the internet, which is useful to my stakeholders as many school computers either do not have internet, have heavily restricted access or if the internet goes down then the program is still useable.

## Computational Methods

### Why is this problem suitable for computational methods?

This problem is very suitable for a computer program as modern computers can generate graphics and do large amounts of maths very quickly. Plotting polynomials by hand often involves repeatedly doing large, difficult sums, which could be time-consuming without a calculator; this is easily done in a programmed solution as computers could complete the math far faster, without any chance of error. Drawing the graph is also suitable to a programmed solution as a computer provides a much higher degree of accuracy and precision than if it were drawn by hand, which would provide the user with a better output.

### Abstraction

Abstraction will be important to my solution; this is because a truly accurate graph has an infinite number of points across it (an example of this is y=x2 : x can accept an infinite number of values). Therefore, I will only plot a certain number of points and will join them using straight lines. By plotting enough points it will be indistinguishable from the actual graph and will reduce the amount of time and resources needed to plot the graph. Additionally, I will only plot the points that the user can see, rather than plotting points that wouldn’t be rendered. This would further reduce the computational resources taken by the application. This is good for my stakeholders as it reduces the hardware requirements for the program to run.

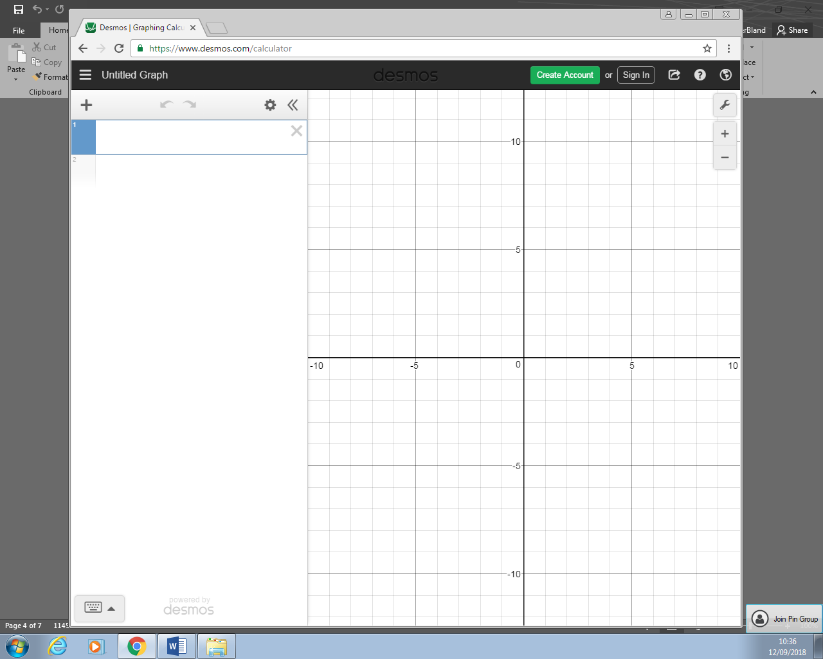
### Problem Decomposition

My solution can be decomposed into smaller problems that can be worked on individually. To decompose the problem, I am going to look at the different aspects of the application and create encapsulated objects that handle their aspect of the application and nothing else. For example, I should create a module which takes the input from the user, processes it and passes the input to other modules. This would allow me to implement each part of the solution individually and make the development stage much easier.

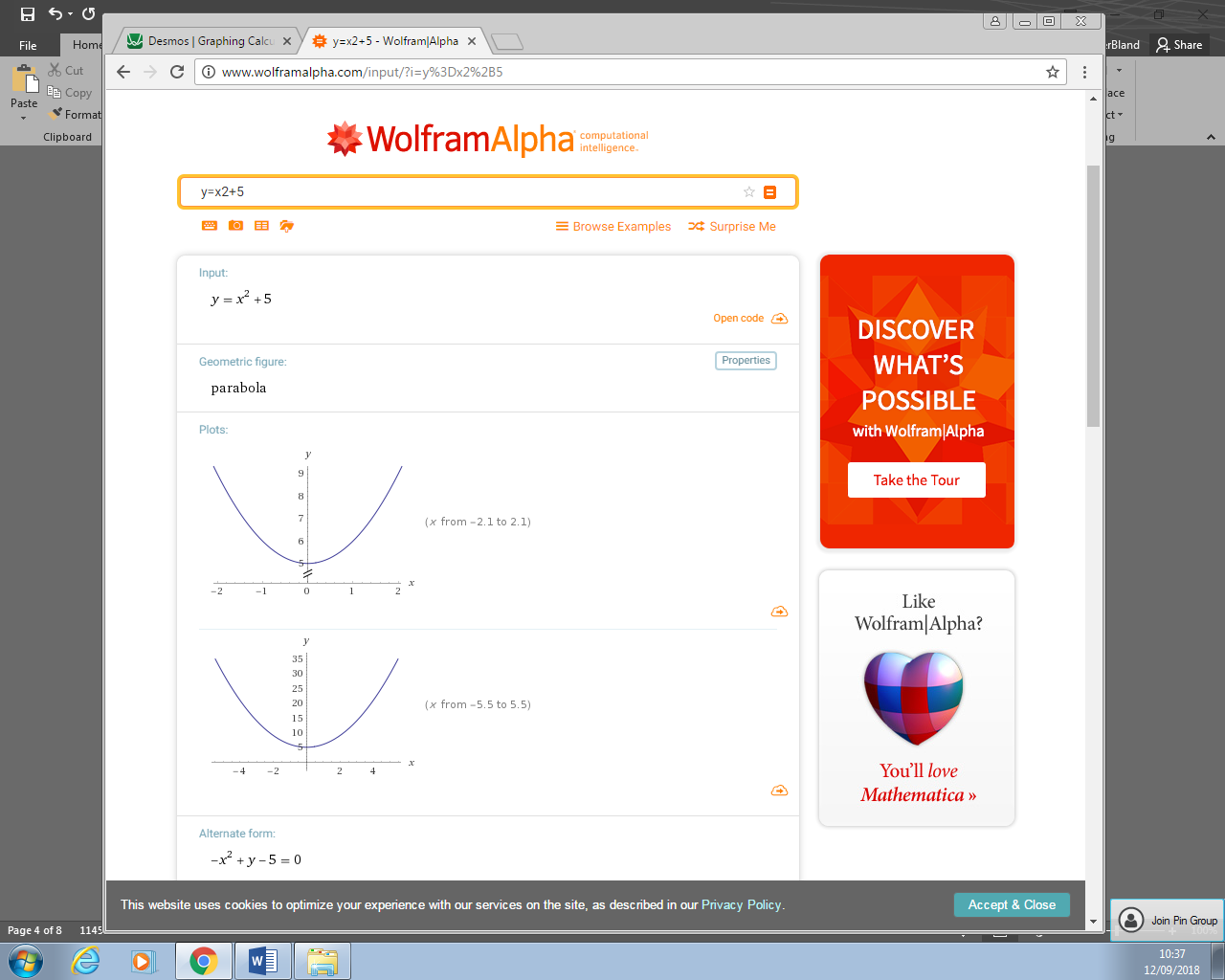
### Output

The output of the program would be in the form of an interactive Cartesian grid that would allow the user to pan and zoom to view the entire graph. This would be valuable to my stakeholders, as it would allow user to see graphs with a higher detail and range when compared to a hand-drawn graph. The output would be useful in an educational purpose, as it would allow students to interact with the graph and grasp a better understanding of the equation show.

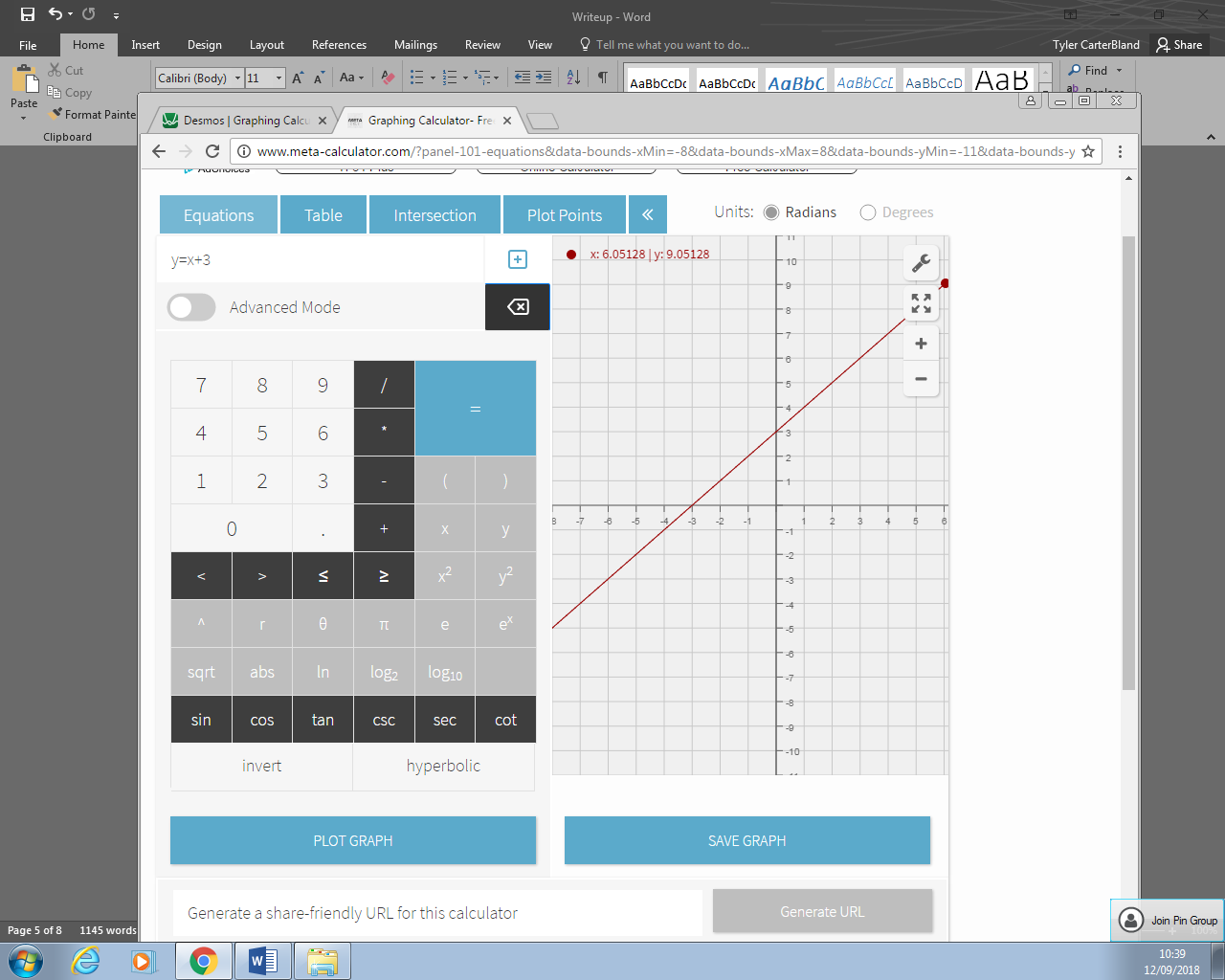
## Research

<https://www.desmos.com/>

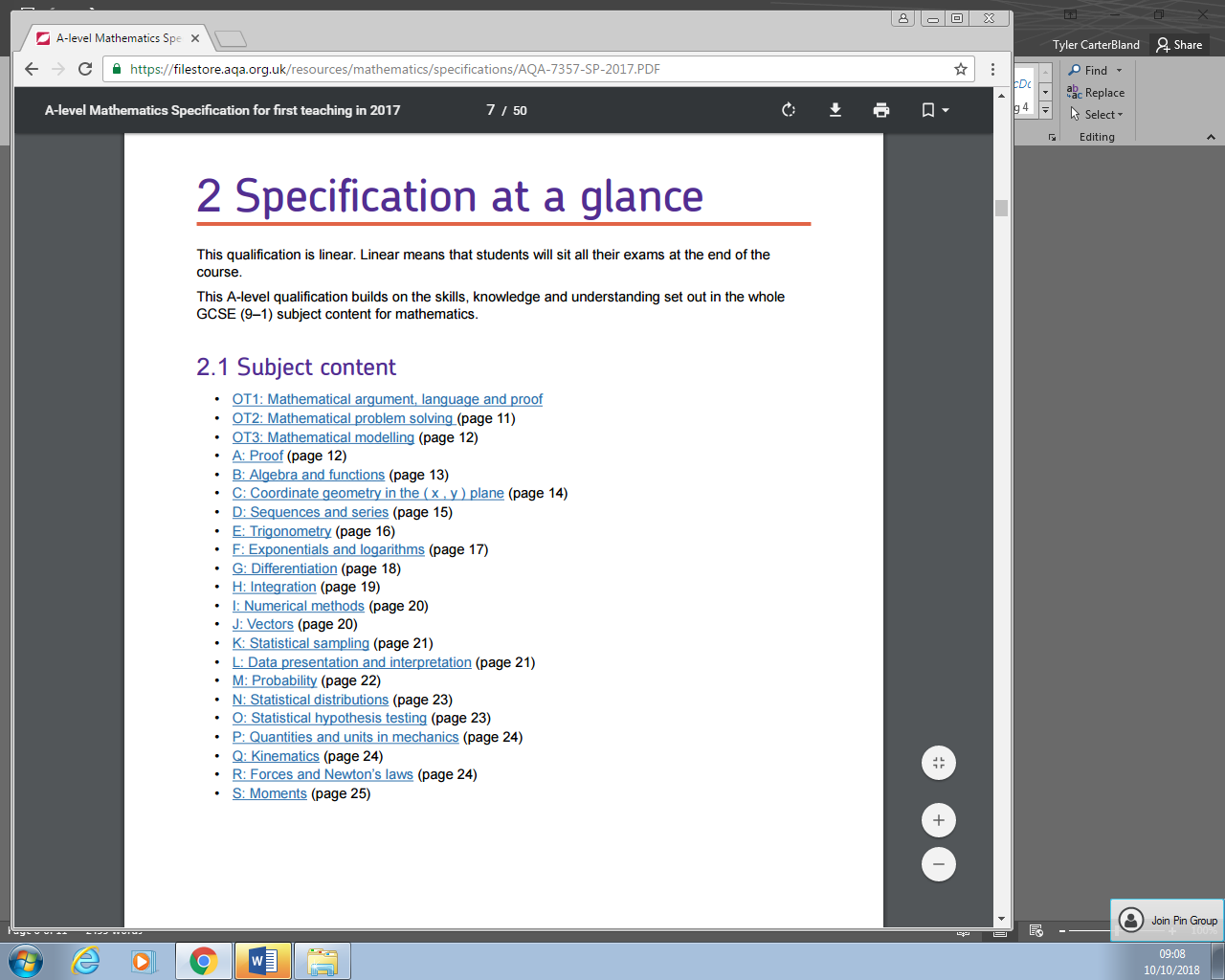
Desmos is an online web application that can graph equations. It provides an interactive grid, and allows the user to type their equation into the GUI to be displayed. Desmos provides many useful features that I believe could be adapted into my solution, such as the ability to save graphs. The most important feature that I could adopt is the pre-generated educational demonstrations that Desmos provides. I believe this feature would be highly valuable to my stakeholders as it allows the user to gain an insight into the parts of many different graphs. Another feature I want to adopt is the ability to include variables in their graph (for example y=mx, where m is variable) with the ability to change the variable with a slider and view the effect on the graph. The strengths of Desmos are that it is completely free, and it works with little delay. The main disadvantage of Desmos is that it requires the internet to use, which is not always available and the website may be blocked by a school’s website filter.

<http://www.wolframalpha.com/>

Wolfram Alpha is an answer engine that allows the user to enter a query and receive a detailed answer, and can plot graphs for the user if an equation is entered. Wolfram Alpha is limited by the fact that the graph is static and only allows the user to view a small portion around the origin. However, it does provide the user with many details about the graph, such as the graphs geometric figure and the graphs roots, domain and range. These could be adapted into my solution that would be valuable to the stakeholders, as it provides many details so that students can gain a deeper understanding. The weaknesses of Wolfram Alpha are that many features require the user to pay monthly, which is not ideal. Furthermore, Wolfram Alpha does not provide much guidance, as its target users seem to be A-Level and university students.

<http://www.meta-calculator.com/>

Meta-Calculator provides an online graphing calculator and an interactive grid. Meta-Calculator is limited as the interactive grid is slow and unresponsive, and the GUI is not intuitive enough for a younger student to be able to use easily. Additionally, the website has many advertisements on it, which causes the school computers to slow down a lot. The main advantage is that it is free, but overall Meta-Calculator is not very useful. Meta-Calculator does have the ability to graph single points however, which can be valuable when demonstrating shapes on a Cartesian grid.

*A-Level and GCSE Maths Specifications*

I have read the specifications for GCSE and A-Level Maths from Edexcel and AQA, two exams boards that schools often follow. The specifications describe the topics needed for the maths exam, and I should ensure that my solution covers as many of these topics as possible so that it is a useful educational tool for my stakeholders.

Some topics I have identified from the specifications are:

* (GCSE) Identifying roots, turning points and intercepts on graphs so my solution should include a feature to highlight and describe these points.
* (GCSE) Plotting graphs using the form y = mx + c and sketching quadratic functions
* (A-Level) Understanding and sketching y = a/x and y = a/x2 (reciprocal) graphs.
* (A-Level) Understanding the effects the transformations y = af(x), y = f(x) + a, y = f(x + a) and y = f(ax).
* (A-Level) Understanding the equation of a circle in the form (x-a)2 + (y-b)2 = r2

## Stakeholder Questionnaire

I have created a questionnaire and sent it to 10 of my stakeholders in order to gain a better understanding of what my stakeholders want in the program. Here are the results of each question:

The majority of my stakeholders find graphs helpful as a visual aid so the ability to graph a wide variety of graphs accurately and quickly will be included in my proposed solution.

My stakeholders rated their understanding of graphs at a 2.6/5 on mean average so a tutorial and educational examples will be included in my proposed solution.

All of my stakeholders want the program to include educational examples, so this will be included in my proposed solution and will be a priority in development.

A majority of my stakeholders want the program to include saving and loading of graphs, so this feature will be included in the proposed solution

Most of my stakeholders want the program to be customizable, so an extensive options menu will be included in the proposed solution.

Most of the stakeholders do not study further maths, therefore features such as polar graphs and hyperbolic functions will be a lower priority in development as they will not be useful for the majority of my stakeholders.

My questionnaire has been very useful as it will allow me to derive many features for my proposed solution.

## Features of the proposed solution

Features that I believe should be in the proposed solution are:

* Panning and zooming about the interactive grid using the mouse
  + This is essential so that users can view any part of the grid, as many graphs will not be near the origin of the grid and many will require zooming in to view in finer detail, or zooming out to view the entire graph.
* A GUI that is simple and easy to use with minimal instructions
  + Younger pupils will use the application, so it is essential that it is easy to use, even for somebody who only has a basic understanding of maths.
* The ability to plot linear, polynomial and exponential graphs
  + These are the types of graphs often used in GCSE and A-Level maths; hence, it is essential that the program can handle these properly.
* The ability to plot trigonometric functions
  + Trigonometric Functions are also used in GCSE and A-Level maths, so it is required that the application can detect certain keywords like ‘sin’ so that the graphs are handled properly.
* The ability to delete graphs.
  + The user should be able to clear the grid to a default state without restarting the program. This is needed for when users make a typo or mistake when creating a graph, and need to change it.
* Pre-generated demonstrations of various common functions and figures.
  + This feature is very important to my stakeholders as students will be able to learn from the demonstrations and the teachers can use them in a classroom setting to demonstrate a certain graph.
* Saving and loading of user-created graphs onto local storage.
  + This feature is needed to store the pre-generated graphs, and this feature will be very helpful to my stakeholders, as teachers will often create demonstrations prior to lessons and will need to save them for later. The graphs will be stored in a JSON or XML file, and be stored in a sub-directory of the program’s current folder.
* Plotting points onto the grid
  + Transforming single points on a graph is a subject included in GCSE and A-Level maths, so plotting a single point rather than a line would be helpful for my stakeholders.
* Points of interest on a graph highlighted, such as turning points and roots.
  + This feature will be great for my stakeholders, as A-Level maths requires the examination of these points of interest, students will find this useful.
* A tutorial on how to use the application
  + My stakeholders include younger students, so the tutorial should be suitable for anybody and should be simple enough so that all my stakeholders can use the program.
* The grid should adapt to the level of zoom
  + This is needed so that the markers and labels on the grid axis adjust and allow for easier viewing of the grid

The limitations of the proposed solutions are that it would not be able to plot polar graphs and very complex equations may be impossible or resource-intensive to graph. The solution also would not be able to plot equations with imaginary numbers or be able to plot 3-Dimensional graphs. These limitations are not of concern to most of my stakeholders; however, Further Maths students will be impacted by the limitations considerably.

Another limitation I must consider is the time I need to create my solution; if my solution becomes too complex or has too many features I may not have enough time to create it. This may mean some less important features may not be implemented, and I will have to future-proof my solution so that more features could be implemented easily.

There are some desirable features that I may revisit if time allows:

* The ability to export graphs to a pdf file
  + This will be very useful for my users as they will easily be able to print off the pdf file and use a paper copy of the graph. My stakeholders would enjoy this feature as teachers may wish to hand out a paper example of a graph, and students would want to use paper copies in their notes.
* The ability to use variables (y = mx) and the ability to use a slider to change the variable.
  + This feature would be useful for my stakeholders as students could use it to view the effect of changing the variable on the graph, and teachers would be able to use it in a classroom lesson to demonstrate the effect of changing a coefficient.

## Software and Hardware Requirements

I am aiming for hardware requirements to be easily achievable, so that school computers would be able to run the application. The hardware requirements are:

* 1 GHz or faster processor
  + This is required to run the application with minimal to no delay. The better the processor, the smoother the program will run.
* 1 GB of RAM
  + This is required for the operating system and program to run smoothly.
* 1 GB of available hard disk space.
  + Required to store the program and any user-created graphs.
* A keyboard or other typing device
  + A keyboard is necessary so that the user can input information into the application.
* A mouse or other similar devices.
  + A mouse would be required to navigate the GUI
* A monitor or other display device.
  + Required to view the application.

The software requirements to run the application are:

* Python v3.6 or newer
  + This will be the programming language I will use to create the application, so it will be required to run the application
* Tkinter for Python
  + Tkinter will be used to create the GUI for the application
* PyGame for Python
  + The interactive grid will use PyGame to function.
* NumPy for Python
  + This is a library for Python that has many maths-related functions.
* Windows 7 or newer
  + The program will be designed for Windows OS and be designed for the newer versions. Older versions of Windows may work but have reduced usability.

## Success Criteria

These criteria **MUST** be met**:**

|  |  |
| --- | --- |
| Criteria | Evidence |
| Grid can be panned across. | Screenshot(s) showing the grid being panned |
| Grid can be zoomed in and out | Screenshots of the grid zoomed out, and grid zoomed in |
| Coordinate Axis appears and can be seen easily | Screenshot of the coordinate axis in the interactive grid |
| Coordinate Axis have labels on them | Screenshot of the labels on the grid axis |
| Labels dynamically adjust to the level of zoom | Screenshot(s) of how the labels change according to the zoom level. |
| A entry field to input equations | Screenshot of the entry field |
| Equations inputted are drawn accurately with little to no delay | Screenshot of the drawn equation and a measurement of the time taken to drawn |
| Intuitive, easy to use interface | Screenshot of the interface and user feedback survey |
| A tutorial that explains how the program is used. | Screenshot of the contents of the tutorial |
| The ability to save and load equations to local storage | Screenshot of the save/load buttons and screenshot of the saved file |
| Many pre-created educational demonstrations to teach users about graphs | Screenshot of the demonstrations |
| A settings menu to adjust certain aspects of the application | Screenshot of the menu and evidence of the effects of different settings |
| Invalid Inputs entered into entry fields do not crash the program | Screenshot of “Invalid Input” error messages. |

These criteria **SHOULD** be met:

|  |  |
| --- | --- |
| Criteria | Evidence |
| Points of Interest highlighted on the grid. | Screenshot of the point of interest being highlighted. |
| The ability to plot trigonometric, logarithmic and exponential functions | Screenshot of the graphs being plotted |
| The ability to plot circles in the form (x-a)2 + (y-b)2 = r2 | Screenshot the plot of the circle |

These criteria **COULD** be met:

|  |  |
| --- | --- |
| Criteria | Evidence |
| Export graphs into a pdf document | Screenshot of the exporting process and the resulting pdf file |
| The ability to plot polar graphs | Screenshot of the application plotting a polar graph |
| When using trigonometric functions, the ability to use degrees instead of radians. | Screenshot of the application being set into radians mode |
| The ability to plot hyperbolic functions (sinh etc.) | Screenshot of the hyperbolic function being plotted. |

## Performance Criteria

I will also be aiming to complete performance criteria for the program, shown in the table:

|  |  |
| --- | --- |
| Criteria | Evidence |
| Program does not use more than 1gb of RAM | Screenshot of the RAM usage of the program |
| Program does not use over 20% of the CPU during use. | Screenshot of the CPU usage of the program |
| Program does not use more than 100mb of hard disk space | Screenshot of program size. |

# Design

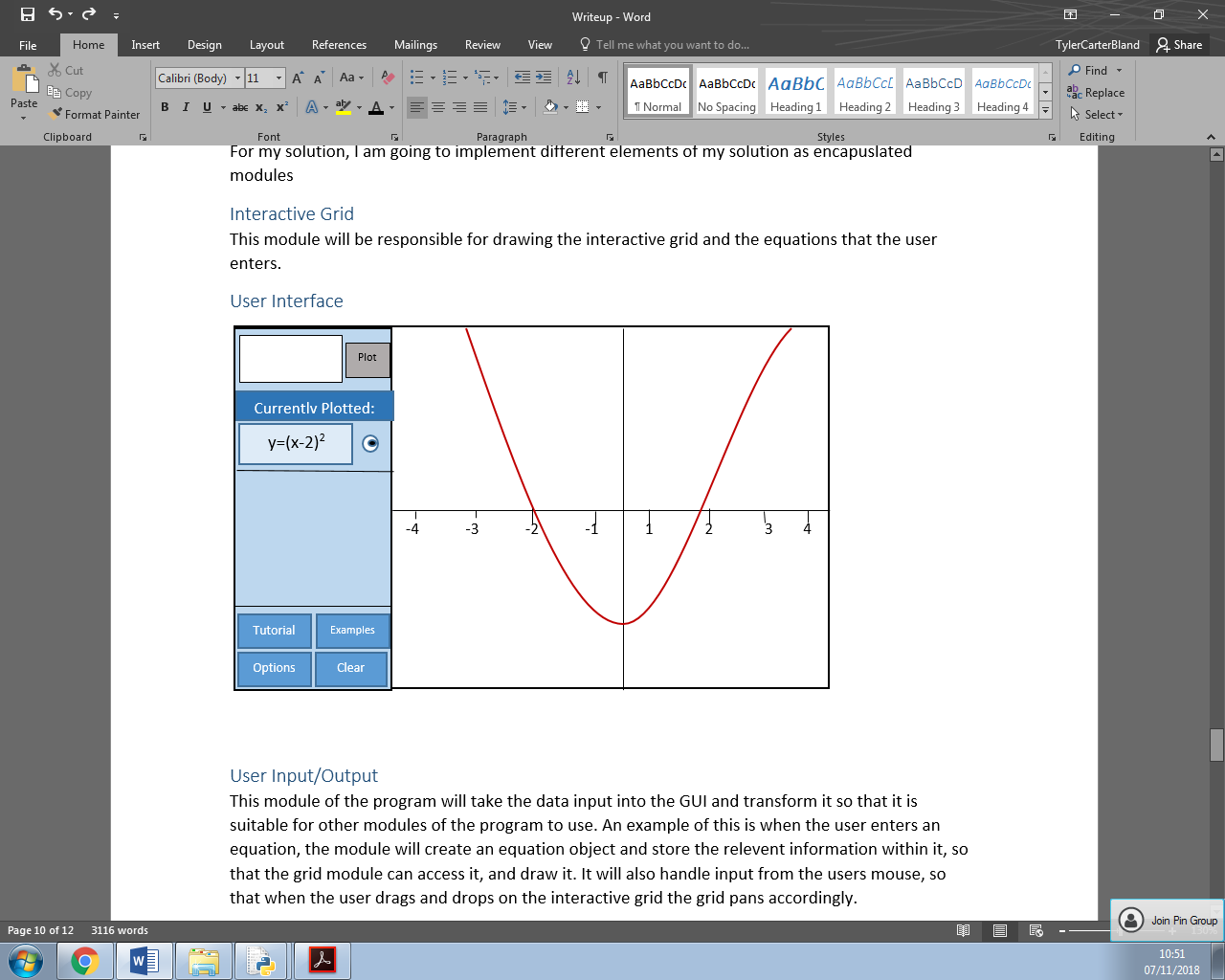
## Decomposing the Problem

For my solution, I am going to implement different elements of my solution as encapuslated modules, as listed below. The benefit of this is that it allows for modular development, allowing for each part of the program to be developed and tested seperately, which will make development easier and testing faster.

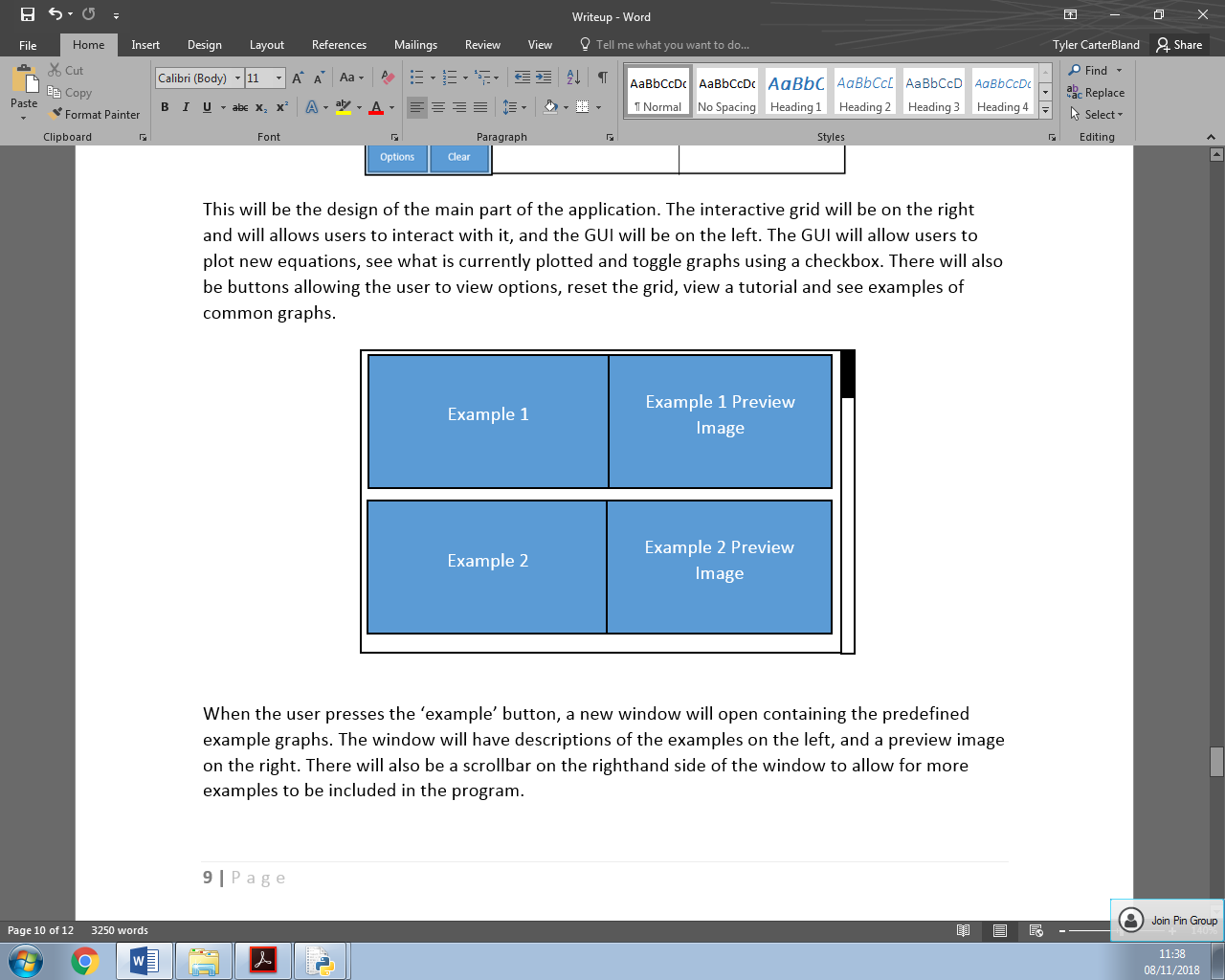
### Interactive Grid

This module will be responsible for drawing the interactive grid and the equations that the user enters. It will allow that user to click and drag on the grid to pan around, and use the mouse wheel to zoom in and out of the grid.

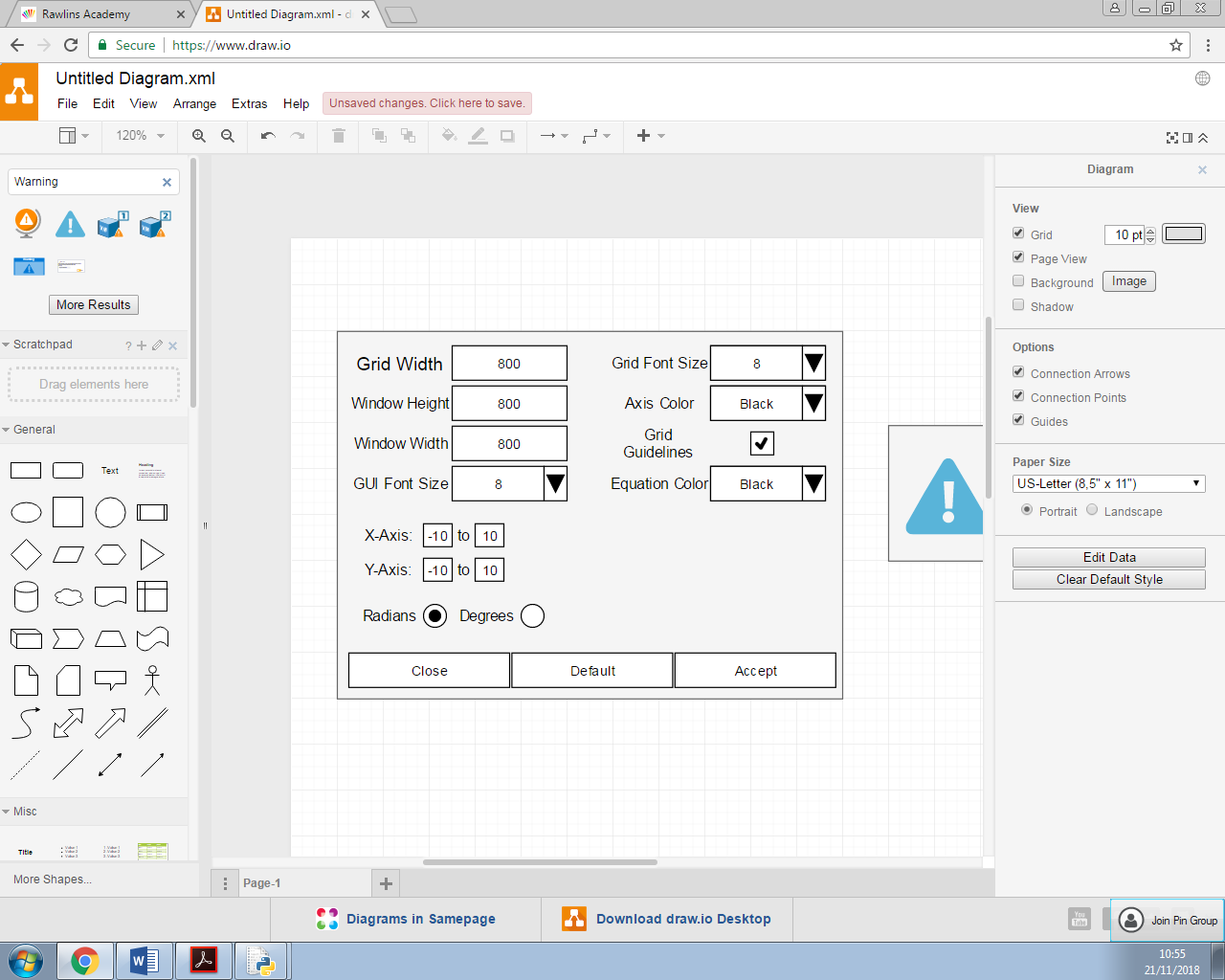
### User Interface



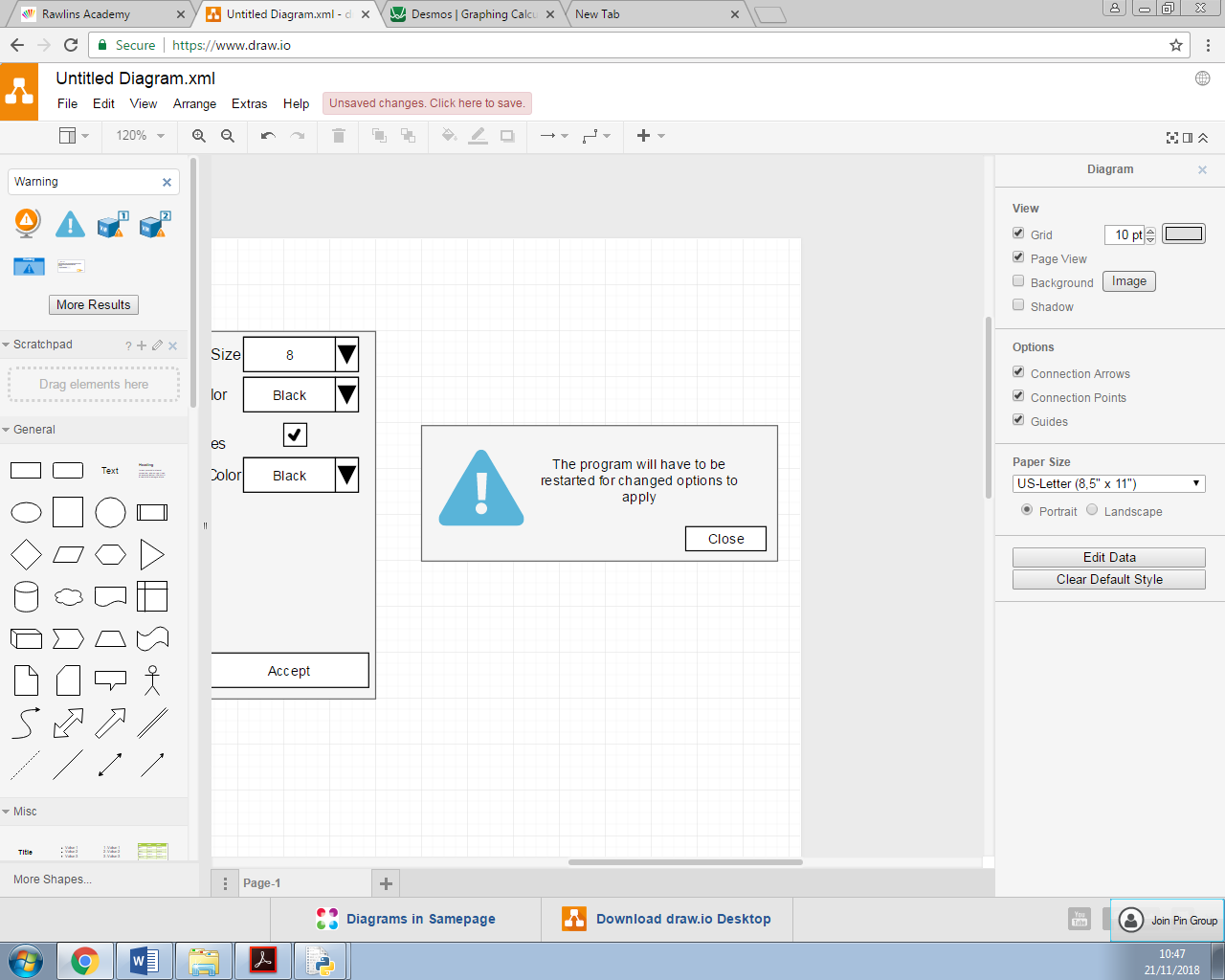
This will be the design of the main part of the application. The interactive grid will be on the right and will allow users to interact with it by clicking and dragging within the grid area. On the left hand side of the screen there will be a panel that allows users to see currently plotted graphs, and click buttons to access the tutorial, example graphs, options and to clear the graph. The ‘currently plotted’ panel will allow users to scroll up and down if the list goes off-screen and will allow users to toggle each graph using a checkbox.



When the user presses the ‘example’ button, a new window will open containing the predefined example graphs. The window will have descriptions of the examples on the left, and a preview image on the right. There will also be a scrollbar on the righthand side of the window to allow for more examples to be included in the program. When the user selects an example, a prompt will open asking them if they are sure as loading the example will clear any existing graphs. When the user selects ‘yes’, the example window and the prompt will close, and the example will be loaded into the interactive grid.



This is my initial design for the options menu. It allows the user to change many aspects of the program, such as font sizes and colors. These options are important so that the program is more accessible, and also so that teachers can change aspects so the program is easier to be seen when projected onto an interactive whiteboard. When the user presses default, the options will change to the default values that the program uses. The options will be saved in an ini file. When the user selects ‘accept’, the changed options will be written to the ini file and a prompt will appear informing that they have to restart the program:

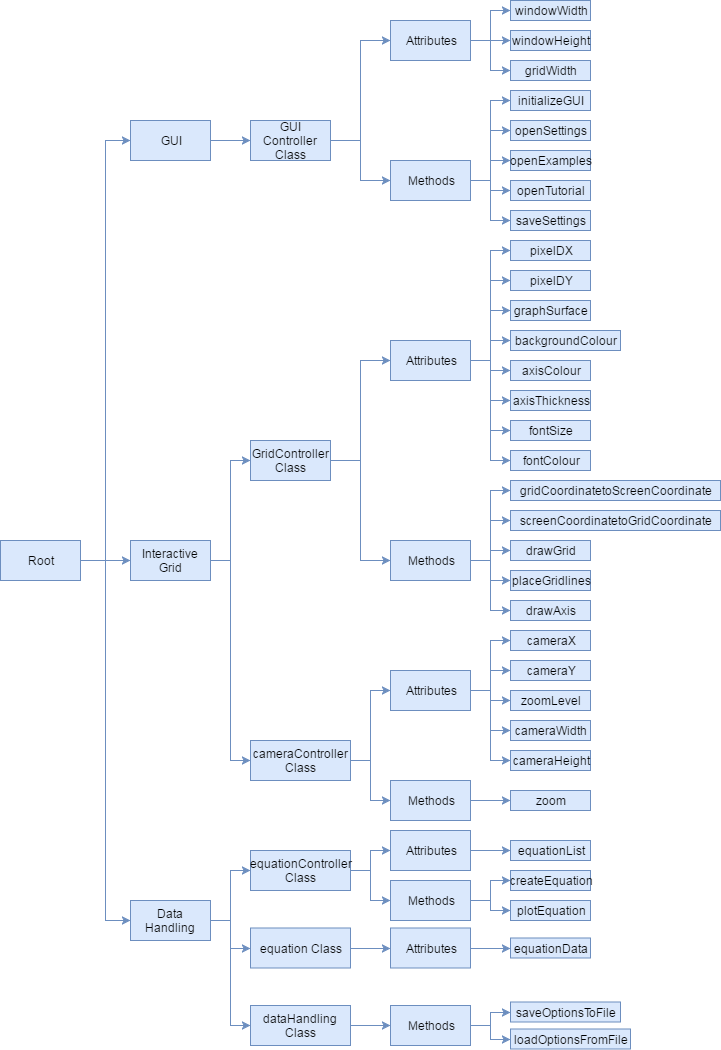


### User Input/Output

This module of the program will take the data input into the GUI and transform it so that it is suitable for other modules of the program to use. An example of this is when the user enters an equation, the module will create an equation object and store the relevent information within it, so that the grid module can access it, and draw it. It will also handle input from the users mouse, so that when the user drags and drops on the interactive grid the grid pans accordingly. This module will also be responsible for accessing json files within the program directory to load and save settings and graphs.

## Defining Structure

This is the structure tree of my design, showing the attributes and methods of each class.



## Algorithms

Algorithms I have identified are:

* gridCoordToScreenCoord
  + This function will take a tuple of two float values as input, x and y, and will return a tuple of two float values. The function takes two coordinates on the interactive grid and returns the coordinates of where they are on the user’s screen.
  + Two variables called ‘pixelDX’ and ‘pixelDY’ will be used, which will define how much grid space a single pixel uses.
  + This is needed so that, for example, if we need to plot the point (2, 2) we can find the point on screen where it will be.

def gridCoordToScreenCoord(coordsTuple):

x, y = coordsTuple

screenX = ((x – cameraX)/pixelDX) + GRID\_WIDTH/2

screenY = ((y – cameraY)/pixelDY) + GRID\_HEIGHT/2

return ( screenX, screenY )

* screenCoordToGridCoord
  + This function will be the inverse function of the above, so it will take two screen coordinates and return the corresponding coordinates on the interactive grid.

def screenCoordToGridCoords(coordsTuple):

x, y = coordsTuple

screenX = pixelDX \* ( x – GRID\_WIDTH/2 ) + cameraX

screenY = pixelDY \* ( y – GRID\_HEIGHT/2 ) - cameraY

return ( screenX, screenY )

* drawGrid
  + This function will be responsible for taking the relevent information needed to draw the grid, and use PyGame’s graphic features to draw the axis, axis numbers and equations onto the grid. First, it fills the graph surface with white to remove the previous frame. Then if grid lines are turned on, the drawGridLines function is called which draws on grid guidelines. The function then finds where 0, 0 is on the screen and draws the axis onto the screen. The graphs are then drawn and the displays are updated.

def drawGrid():

graphSurface.fill(white) #resetting the graph surface to white

if settings.gridLines == True:

drawGridLines()

#Get where 0,0 is on the screen so we can draw the axis

screenXZero, screenYZero = gridCoordToScreenCoord( (0, 0) )

#Draw the grid axis

draw.line(graphSurface, black, (screenXZero, 0), (screenXZero, GRID\_HEIGHT), 2)

draw.line(graphSurface, black, (0, screenYZero), (GRID\_WIDTH, screenYZero), 2)

for graph in graphList:

graph.draw()

#Update the displays

pygame.display.flip()

guiController.update()

* zoomIn/Out
  + This function will be responsible for zooming in and out of the grid, and will take input from the user input module. The amount of zoom created by one input will be proportional to the current level of zoom. CameraWidth and CameraHeight are two variables which define how far the camera can see, horizontally and vertically. As the change in cameraWidth and cameraHeight is 20% of the current value, the cameraWidth and Height cannot be changed below zero, which would produce an error.

def zoom(direction):

if direction == “in”:

cameraWidth = cameraWidth – ( 0.2\*cameraWidth )

cameraHeight = cameraHeight – ( 0.2\*cameraHeight )

else

cameraWidth = cameraWidth + ( 0.2 \* cameraWidth )

cameraHeight = cameraHeight + ( 0.2 \* cameraHeight )

pixelDX = cameraWidth / GRID\_WIDTH

pixelDY = cameraHeight / GRID\_HEIGHT

* plotEquation
  + This function will take an equation object as input and attempt to plot the corrosponding equation to the graph. It will create a list of evenly spaced x-coordinates between the highest and lowest x-coordinates that are visible on screen, and calculate the corrosponding y-value for each one. Then it will use pygame’s drawing functions to connect each pair of x,y values to form a graph. The list of evenly spaced x-coordinates must be large enough so that the graph is accurate, but not so large that it slows down the program.

def plotEquation(equation):

xValues = numpy.linspace(cameraX-cameraWidth//2, cameraX+cameraWidth//2, noOfPoints).tolist()

yValues = []

for x in xValues:

yValues.append( equation.calculate(x) )

for i in range( xValues.length-1 ):

x, y = gridCoordToScreenCoord( (xValues[i],yValues[i] ) )

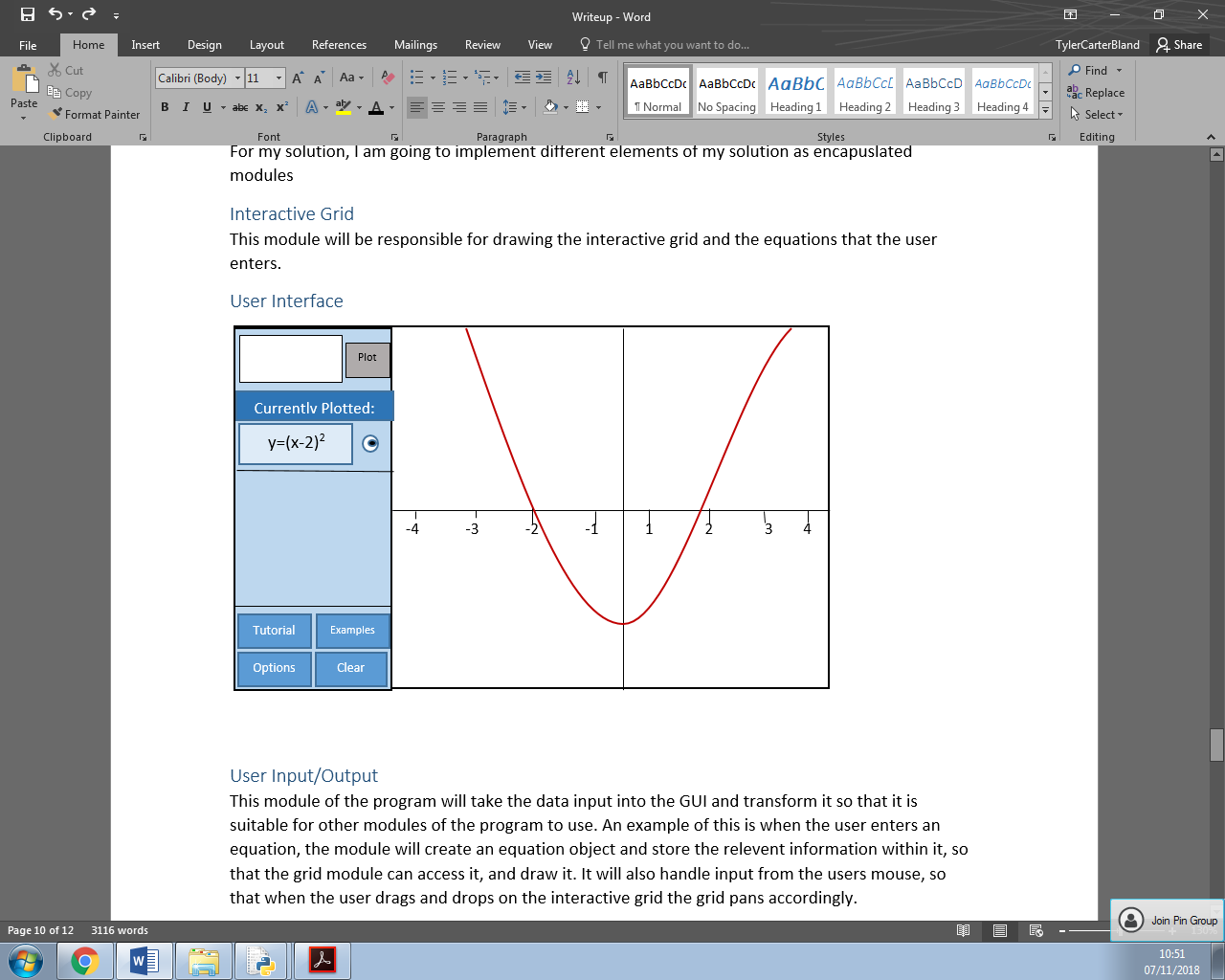
nextX, nextY = gridCoordToScreenCoord( (xValues[i+1],yValues[i+1]) )

pygame.draw.line(graphSurface, red, (x,y), (nextX,nextY), 1 )

## Usability Features

### Navigation

I have designed the GUI of my design to have a simple and easy to use layout:



The main window places the interative grid on the right, and places other elements on a separate panel to create a clear distinction where the interative element of the program is. The options menu will allow for various elements of the program to be altered, such as the size of the font, the colours and the size of the window. This will allow the users to change the program as they wish such that it is easy for them to navigate and use.

### Ease of Use

The program will feature a tutorial window, which will give a visual tutorial of how the program works and the controls of the grid. The tutorial will be launched immediately on the first launch of the program so that new users are taught how to use the program. Additonally, if the user enters an incorrect data type into a field, the program will inform them of what data type to enter via the use of dialog boxes.

### Error Prevention

The program will attempt to prevent the accidental loss of user’s data, as the ‘clear’ button will provide a dialog box asking them if they are sure they want to clear their graphs, and when the user attempts to exit the program a dialog box will open asking them if they are sure they want to exit the program.

## Design Feedback

I created another questionnaire to gain feedback on my design, asking the same 10 stakeholders:

The majority of my stakeholders agreed that the design looked easy-to-use and was appropriate, so I have decided to use this GUI design for my program.

## Variables, Data Structures, Validation

### Key Variables

These are the key variables that will be used in my program:

|  |  |  |
| --- | --- | --- |
| Name of Variable | Data Type | How is it used? |
| gridWidth | Integer | This is the width of the grid in pixels. The user will be able to change this in the options menu. This will be used in many functions to determine if a point is on screen, or to transform a grid coordinate to a screen coordinate and vice-versa. |
| windowWidth | Integer | This is the width of the window in pixels, including the width of the grid. This must always be bigger than gridWidth. This can be changed by the user in the options. |
| windowHeight | Integer | This is the height of the window in pixels. The height of the window and the height of the grid will be the same, so a gridHeight is not needed. |
| cameraX | Integer | This will be the x-coordinate that the grid camera is currently centered on. |
| cameraY | Integer | This will be the y-coordinate the grid camera is centered on. |
| cameraWidth | Integer | This will be how far the user can see horizontally, from the coordinates the camera is centered on. For example, with a cameraWidth of 10, centered on (0,0), the user would be able to see from -5 to 5. |
| cameraHeight | Integer | How far the user can see vertically. |
| gridDX | Floating Point | This will be determined by cameraWidth divided by gridWidth, and will be used when converting a screen coordinate to a grid coordinate. |
| gridDY | Floating Point | Similar to above, but determined by cameraHeight divided by gridHeight |
| equationSamplingRate | Integer | This will be the number of points plotted for a single equation. Higher numbers will produce greater accuracy whereas smaller numbers will be faster to process. |

### Data Structures

This table will detail the data structures that will be used in my program:

|  |  |  |
| --- | --- | --- |
| Name of Data Structure | Type | How is it used? |
| equationList | One-Dimensional Array | This array will store all of the equation objects created by the user. |
| currentSettings | Dictionary | This dictionary will store the values of the users current settings. Using a dictionary allows for the settings to be easily converted to a json format to be saved to a json file. |
| defaultSettings | .json File | This file will store the values of the default settings in the json format. |

### Classes

This table will detail the classes that will be used and their use:

|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | Methods | Attributes | Description |
| guiController | openSettings, openTutorial | windowWidth, gridWidth, windowHeight, Tkinter Widgets | This class will handle the GUI elements of the program. All of the program GUI elements will be initialized as attributes of this class. |
| gridController | getGridCoordinate, getScreenCoordinate, drawGrid | cameraX, cameraY, cameraWidth, cameraHeight, pixelDX, pixelDY | This class will be responsible for the interactive grid element of the program |
| inputController | handleInput, moveGraph, zoom | keys | This element will be responsible for handling the user’s input. The keys attribute stores the keys and mouse buttons currently pressed. |
| equationController | createEquation, calculateNoOfPlots | equationList | This class will be responsible for creating and storing equation objects, as well as calculating the correct number of plots to make on the graph. |
| equation | solve | visible, leftSide, rightSide | This class represents an equation that the user has input. The solve method takes an x value and returns the corresponding y-value(s). |

### Validation

My design will require validation for these inputs fields, as if it were not validated, it could cause unexpected behaviour or a crash which could result in loss of the user’s data. This table shows my plans for validation:

|  |  |  |
| --- | --- | --- |
| Variable to be validated | Expected Data type | Actions taken if data type incorrect |
| Equation Input | An expression using x as a variable, for example ‘x^2’ or ‘5x+3’ | Display a message stating ‘Incorrect equation form’ |
| Settings Input | Integer | Display a message stating ‘Invalid Input’ |
| Grid Width Setting | Integer less than the ‘Window Width’ setting | Display a message stating ‘Grid Width must be smaller than Window Width’ |

## Iterative Development Test Data

When creating my program I will use an iterative development plan, meaning that when I create a new aspect of the program, it will be tested immediately.

|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Test Description | Test Input | Expected Result |
|  | Drawing a linear function | x | The graph is rendered correctly: |
|  | Operations work and affect how the graph is drawn | x+3 | The graph is rendered correctly: |
|  | Polynomials can be drawn | x^2 + 3x + 3 | The graph of x2+3x+3 is drawn correctly: |
|  | Trigonometric functions can be drawn | sin(x) | The graph of sin(x) is drawn correctly: |
|  | Discontinuous graphs are drawn correctly | 1/x | The graph of 1/x is drawn correctly: |
|  | Illegal graph inputs are detected and an error message is displayed. | ‘Foo’ | An error message is displayed stating ‘Incorrect equation form’ |
|  | Plotted Graphs appear in the ‘plotted graphs’ section | x | The graph appears under the ‘plotted graphs’ section on the GUI. |
|  | Plotted graphs can be toggled on and off | Checkbox toggle button | The graph is not drawn when the checkbox is toggled off and appears when toggled on. |
|  | Plotted graphs can be deleted. | Delete button | The graph is not drawn anymore and does not appear under the ‘plotted graphs’ section |
|  | The options button opens the options window | Options button | The options window opens |
|  | The examples button opens the examples window | Examples button | The examples window opens |
|  | The tutorial button opens the tutorial window. | Tutorial button | The tutorial window opens. |
|  | The clear button deletes every currently plotted graph | Clear button | All plotted graphs are deleted and removed from the ‘plotted graphs’ section. |
|  | The ‘grid width’ option works correctly | 1000 | The grid width is 1000 pixels wide. |
|  | The ‘window height’ option works correctly. | 1000 | The window is 1000 pixels high. |
|  | The ‘window width’ setting works correctly | 1000 | The window width, including the grid, is 1000 pixels wide. |
|  | The GUI font size option works correctly | 12 | The font on the GUI are in size 12 font. |
|  | The grid font size option works correctly. | 12 | The grid font is in size 12. |
|  | The Axis Colour option works correctly | Red | The grid axis are drawn in red. |
|  | The grid guidelines option works correctly | Checkbox Toggled On | The grid guidelines appear. |
|  | The equation colour option works correctly | Red | The plotted equations are drawn in red. |
|  | The x-axis settings work correctly | -10 and 10 | The camera is adjusted so that it can see from -10 to 10 on the x-axis |
|  | The y-axis setting work correctly | -10 and 10 | The camera is adjusted so that it can see from -10 to 10 on the y-axis |
|  | The radians option works correctly | Radians radio button checked | Trigonometrical functions plot using radians |
|  | Illegal Inputs in the options are detected. | ‘Foo’ | An error message appears stating ‘Invalid Input’ |
|  | The user’s settings are saved. | ‘Accept’ button in settings | The user’s settings are saved in the programs directory. |
|  | The user’s settings are loaded. | Program Started | The user’s settings are loaded and applied correctly. |
|  | The ‘default’ settings button works correctly. | ‘Default’ button | The user’s options are restored to the default settings. |
|  | Interactive Grid can be panned | Mouse button clicked and dragged | The grid pans according to the movement of the mouse |
|  | Interactive grid can be zoomed in/out | Mouse wheel in/out | The grid zooms in and out. |

## Post Development Test Data

After the program is fully developed, this is the test data I will use to test certain parts of the program:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | Test Description | Input Type | Test Input | Expected Result |
|  | Running the program | Valid Data | Program Started | The program starts |
|  | Placement of GUI | Valid Data | Program started | The GUI is placed correctly |
|  | GUI responsiveness | Valid Data | Clicking each of the pressable GUI elements | The program responds within 100ms |
|  | Interactive Grid Rendering | Valid Data | Program Started | The guidelines, guideline markers and axis are all rendered correctly |
|  | Interactive grid panning | Valid Data | Interactive Grid clicked and dragged | The interactive grid pans according to the movement of the mouse |
|  | Linear Graph Rendering | Valid Data | ‘y=x’ in graph entry | Graph of y=x is rendered |
|  | Use of coefficients in graph | Valid Data | ‘y=5\*x’ in graph entry | Graph of y=5\*x is rendered |
|  | Polynomial Graph Rendering | Valid Data | ‘y=x\*\*2’ in graph entry | Graph of y=x\*\*2 rendered |
|  | Use of addition in graph | Valid Data | ‘y=x + 3’ in graph entry | Graph of y=x+3 rendered |
|  | Use of sin function | Valid Data | ‘y=sin(x)’ | Graph of y=sin(x) rendered |
|  | Use of cos function | Valid Data | ‘y=cos(x)’ | Graph of y=cos(x) rendered |
|  | Graphs with y and x on one side | Valid Data | ‘y+x=15’ | Graph of ‘y+x=15’ rendered |
|  | Reciprocal Equations | Valid Data | ‘y=1/x’ | Graph of y=1/x rendered |
|  | Graph using unknown variable | Invalid Data | ‘y=a’ | Error Message produced |
|  | Graph using invalid syntax | Invalid Data | ‘y=3x’ | Error message produced |
|  | Graph using exponential | Valid Data | ‘y=2\*\*x’ | Graph rendered correctly |
|  | Graph using sinh | Valid Data | ‘y=sinh(x)’ | Graph rendered correctly |
|  | ‘Currently plotted’ | Valid Data | A graph is entered | The graph appears in the ‘currently plotted’ list |
|  | Graph ‘delete’ button | Valid Data | Delete button is pressed | Graph deleted from list and not rendered. |
|  | Graph ‘toggle’ button | Valid Data | Toggle button is pressed | Graph visibility toggled but not deleted from list |
|  | Tutorial Button | Valid Data | Tutorial Button Pressed | The tutorial is opened |
|  | Examples Button | Valid Data | Examples button pressed | The examples are opened |
|  | Save Graphs Button | Valid Data | ‘Save’ button pressed | The save graphs window is opened |
|  | Load Graphs button | Valid Data | ‘Load’ button pressed | The load graphs window is opened |
|  | Options Button | Valid Data | Options button pressed | The options window opens |
|  | Options menu reads current options | Valid Data | Options menu opened | The options fields contain the current options in the options.json file |
|  | Options menu saves changes | Valid Data | Options are changed with valid values | The changed options are saved to options.json in the correct format |
|  | Default Options button | Valid Data | ‘Restore Default’ button pressed | The options in optionsDefault.json are saved into options.json |
|  | Invalid Options Entered, integer fields | Invalid Data | String values entered into integer fields | An error message opens instructing the user how to enter options correctly |
|  | Invalid Options Entered, Colour fields | Invalid Data | Colour values not entered in R G B form | An error message opens instructing the user how to enter options correctly |
|  | WindowWidth Option | Valid Data | ‘1600’ in WindowWidth option | After program restarts, window is 1600 pixels wide |
|  | GridWidth option | Valid Data | ‘1400’ in windowWidth option, ‘1200’ in gridWidth option | After program restarts, grid is 1200 pixels wide |
|  | GridWidth Option with invalid Data | Invalid Data | ‘1000’ in windowWidth and ‘1400’ in gridWidth | An error message opens instructing the user how to enter options correctly |
|  | windowHeight Option | Valid Data | ‘1000’ in windowHeight | After program restarts, window is 1000 pixels high |
|  | axisThickness Option | Valid Data | ‘5’ in axisThickness | After program restarts, axis is 5 pixels thick |
|  | axisColour option | Valid Data | ‘0 255 0’ in axisColour | After program restarts, axis is green coloured |
|  | plottedColour options | Valid Data | ‘0 255 0’ in plottedColour | After program restarts, plotted graphs are green |
|  | plottedThickness option | Valid Data | ‘5’ in plottedThickness | After program restarts, plotted graphs are 5 pixels thick |
|  | backgroundColour option | Valid Data | ‘0 255 0’ in backgroundColour | After program restarts, background is blue |
|  | Guidelines option | Valid Data | Guidelines toggled off | After program restarts, guidelines are not visible |
|  | guidelineColour option | Valid Data | ‘0 255 0’ in guidelineColour | After program restarts, guidelines are blue |
|  | guidelineFontSize option | Valid Data | ‘20’ in guidelineFontsize | After program restarts, guideline font was size 20 |
|  | guidelineThickness option | Valid Data | ‘5’ in guidelineThickness option | After program restarts, guidelines are 5 pixels thick |
|  | fontColour option | Valid Data | ‘0 255 0’ in fontColour | After program restarts, grid font is blue |
|  | noOfPlots option | Valid Data | ‘200’ entered in noOfPlots option | After program restarts, the base number of plots used to plot graphs is 200 |
|  | Examples description | Valid Data | Example button pressed | The description of the example is read from the example file |
|  | Examples loading | Valid Data | Load example button pressed | The example was read from the file and rendered on the screen. |
|  | Graph Saving | Valid Data | Graph entered and ‘save graphs’ button pressed | The equation and description are saved to the file under the entered filename |
|  | Graph Saving Using Invalid Filename | Invalid Data | Invalid Filename entered while saving graph | An error message opens |
|  | Graph Loading Description | Valid Data | Load Graphs button pressed | Description loaded from graph files |
|  | Graph Loading | Valid Data | Load graph button pressed | Equations are loaded correctly from graph files |

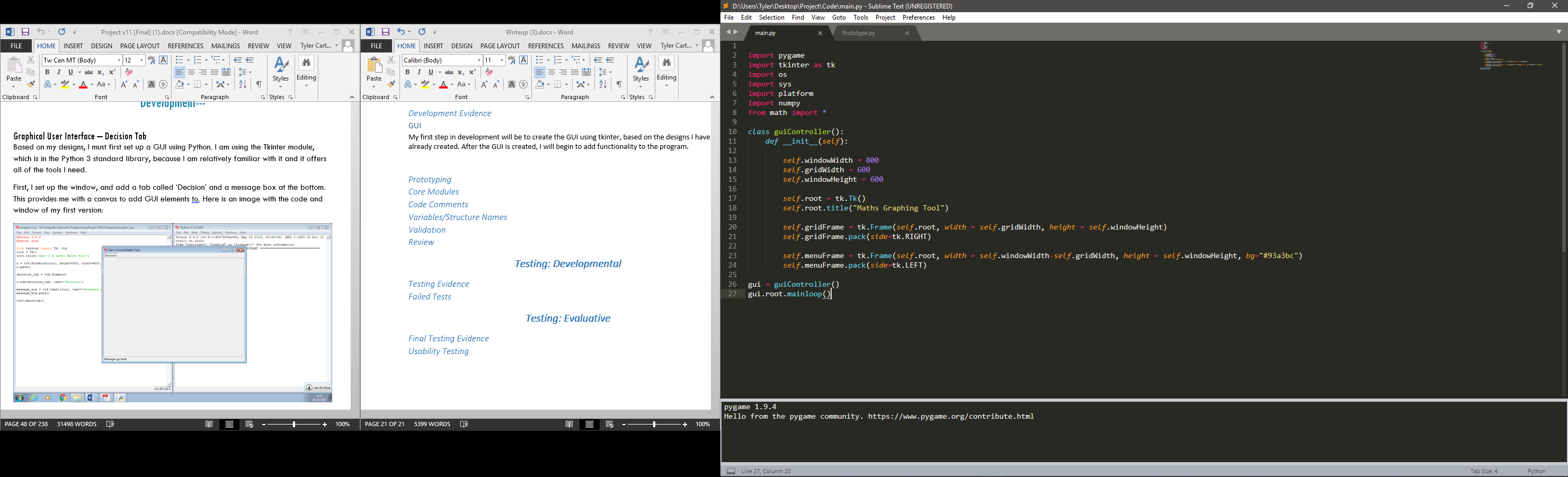
# Development

## Development Evidence

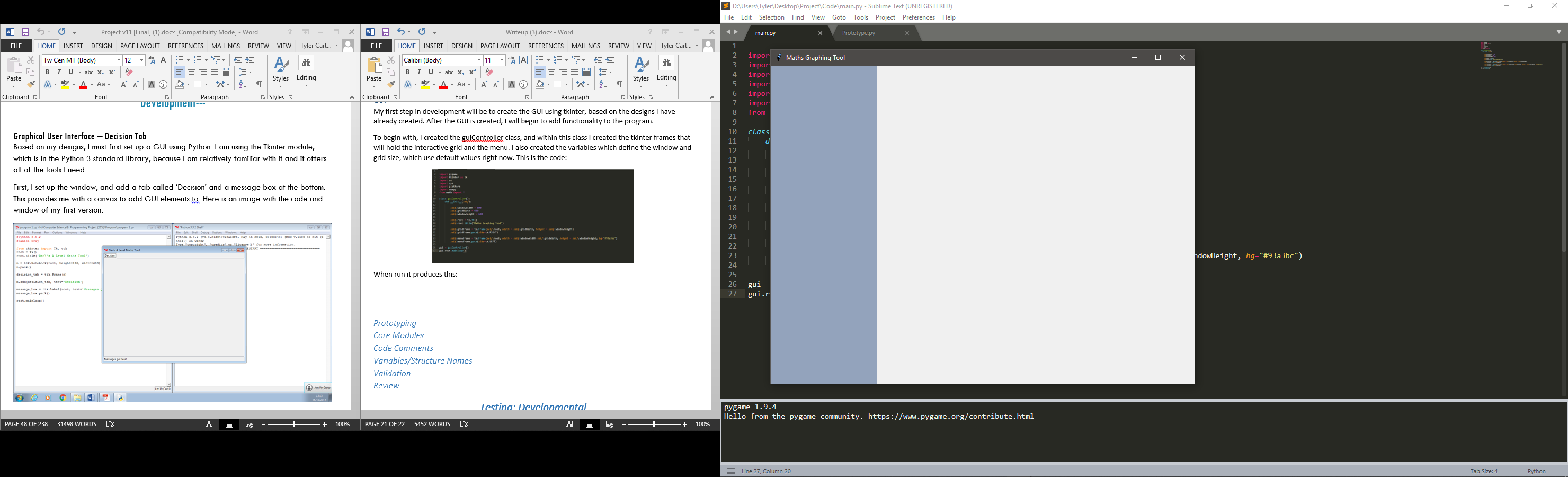
### GUI

My first step in development will be to create the GUI using tkinter, based on the designs I have already created. After the GUI is created, I will begin to add functionality to the program.

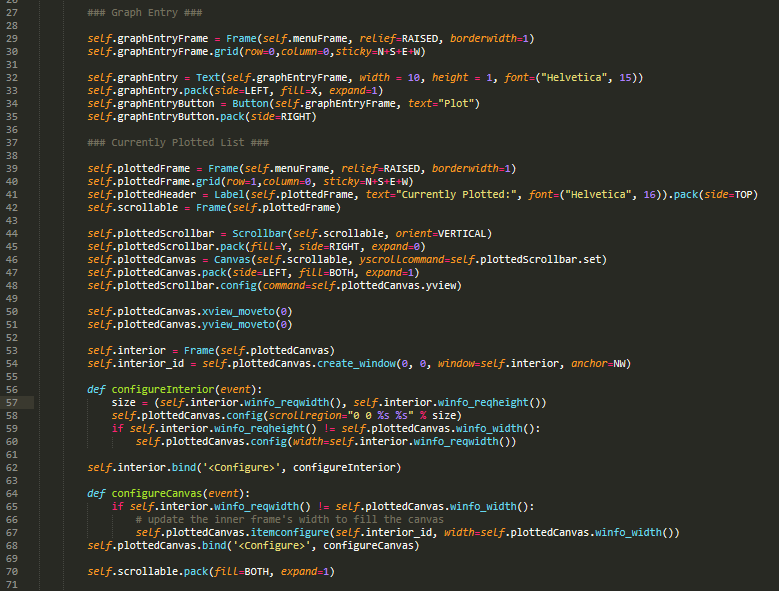
To begin with, I have imported the libraries I need for the program to run; keeping all of the imports in one place allows me to change the libraries I’m using easily, additionally I imported some libraries under different names, to save time when typing out code. I created the guiController class, and within this class I created the tkinter frames that will hold the interactive grid and the menu. I also created the variables which define the window and grid size, which use default values right now. This is the code:



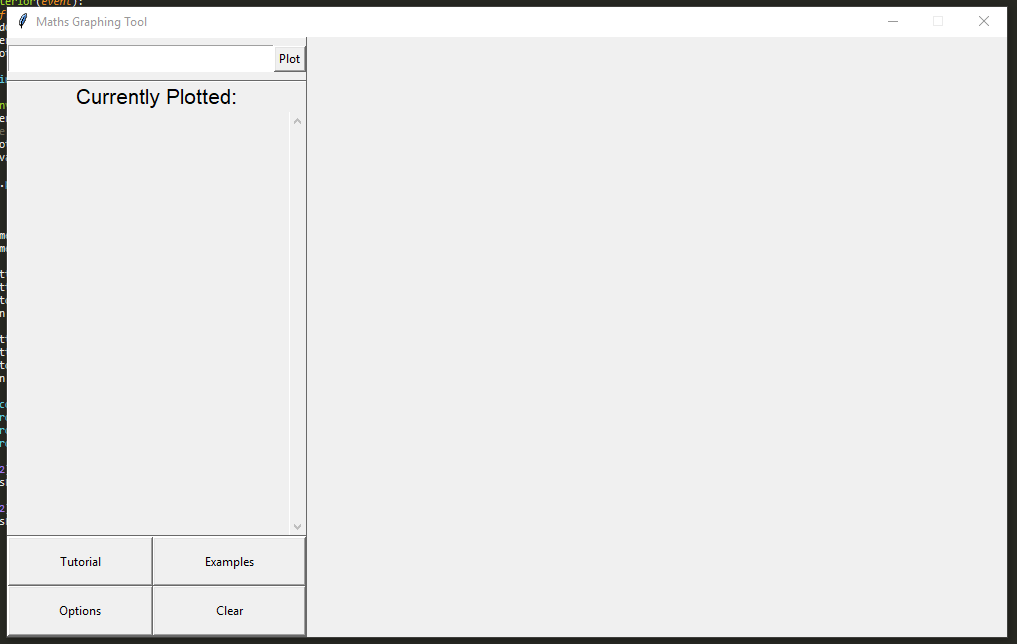
When run it produces this:



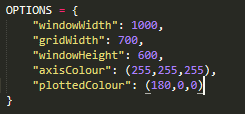
Next, I populated the left frame with the graph entry field, currently plotted list and options buttons. The currently plotted list functions correctly when new widgets are added to it, and the scrollbar allows for the user to scroll through the list. The list uses a canvas widget linked with a scrollbar widget and an interior frame; to make a widget appear on the list it is created as a child under the interior widget. When new widgets are added to the list during runtime it adjusts itself, allowing for all widgets in the list to be accessible as the user can scroll up and down. When the user scrolls the scrollbar, the interior frame is translated accordingly.

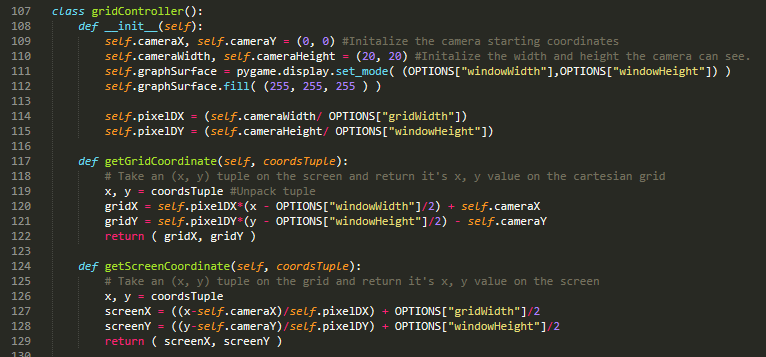


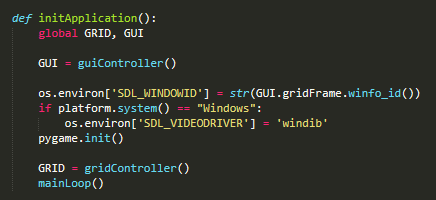




Now that I have created the basic GUI of the program I can begin to work on the interactive grid. This is the next step in my development because the graphing functionality, examples and tutorial all require the grid to be functioning before I can develop them. I will start by initializing a pygame window and embedding it within the gridFrame widget. I have also created a global dictionary which will contain the options selected. I have also coded the getGridCoordinate and getScreenCoordinate functions mentioned in my design, as they will be needed to begin creating the interactive grid. Here is the code:

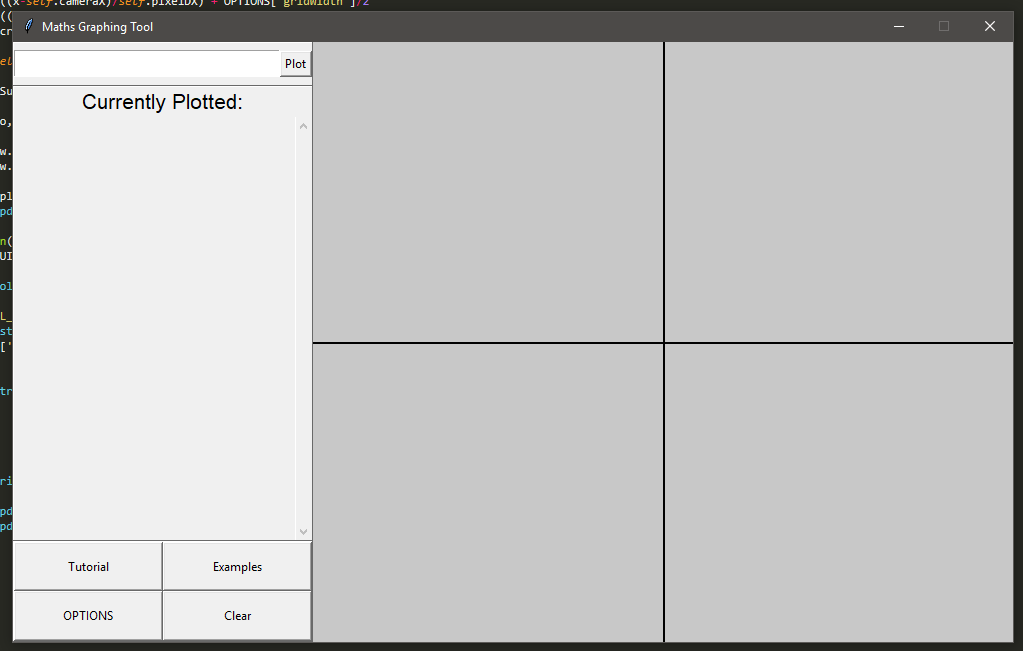


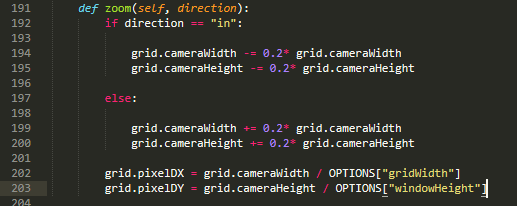


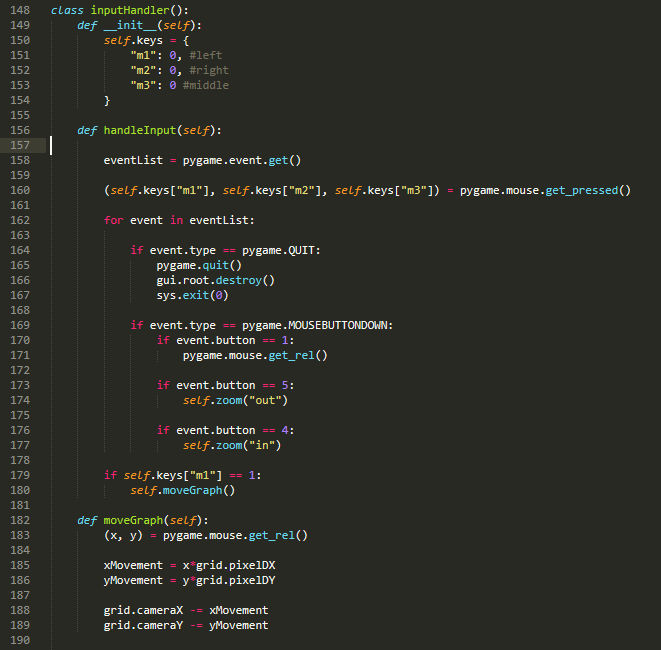


To begin the drawGrid function, I will make it draw the grid axis by finding the screen coordinates of 0, 0 and using the pygame.draw function to create lines spanning the screen. I have also created the mainLoop function, which will be continuously called until the program is exited, therefore I must ensure all functions called within the mainLoop are optimised and efficient to prevent the program from slowing down or using too much memory.





Next, I will create the inputHandler class and create the handleInput and moveGraph methods, so that the grid can be panned and zoomed. The handleInput method uses pygame’s event.get function which returns a list of inputs that the user has given, such as mouse clicks and key presses. I will also use pygame’s mouse.get\_rel function which gives the distance the mouse has moved since the function has last been called, which will be used to calculate the distance and direction the user has panned. The zoom method simply adjusts the current cameraWidth and cameraHeight by a fifth of the current values, zooming in decreases the values and zooming out increases the values. 

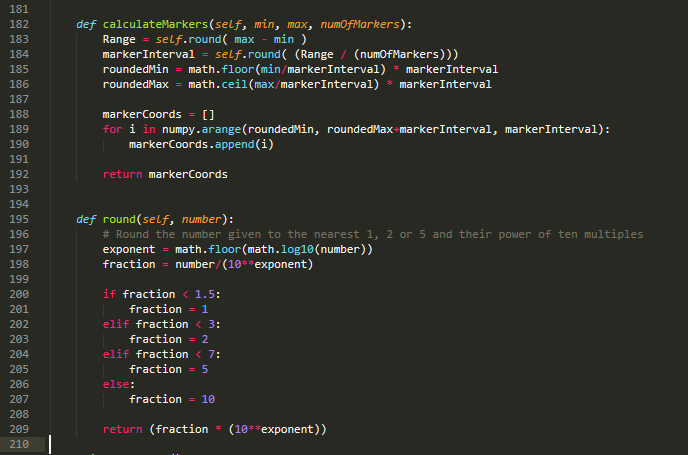


### Test Table 1

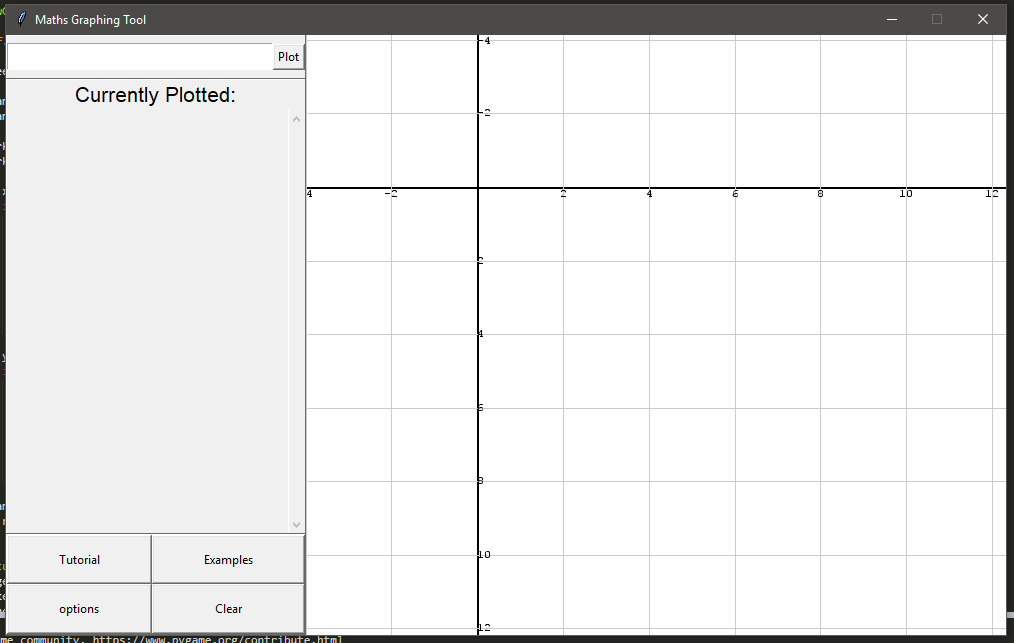
This is the iterative testing I have done so far.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Input | Expected Result | Actual Result | Pass/Fail | Actions Needed |
| 1. | Interactive Grid can be panned | Mouse button clicked and dragged | The grid pans according to the movement of the mouse | The grid pans according to the movement of the mouse | Pass | None |
| 2. | Interactive grid can be zoomed in/out | Mouse wheel in/out | The grid zooms in and out. | The grid zooms in and out. | Pass | None |

Now that the zoom and panning functions are working, I will work on the axis guidelines and markings. A simple solution to the grid markings would be to split the x and y range into n intervals and place a marker on every interval, but this creates markings that are not useful. Usually, the marks on the axis usually step by 1, 2 or 5 and the power of ten multiples of the numbers, for example a grid ranging from x=-10 to x=10 would place markings on -10, -5, 0, 5 and 10. I will create an algorithm to select the appropriate markings given the current x and y range, and I will alter the drawGrid method to include rendering the markings, and draw vertical and horizontal guidelines ontop of the markings. This is the code:







### Test Table 2

Here are the iterative tests I have done on the guidelines:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Input | Expected Result | Actual Result | Pass/Fail | Actions Needed |
| 3. | Markers are rendered in the correct position | Launch Program | Marker text is drawn corresponding with its position | Marker text was drawn correctly, but was often obstructed by the axis | Fail | Render marker text beside the axis rather than on top. |
| 4. | Marker intervals are selected appropriately using a large range. | Zooming out to a range of -100 to 100 in both x and y axis. | Marker interval is 20, markers drawn on -100, -80, -60, -40, -20, 0, 20, 40, 60, 80 and 100. | Marker interval was correct. | Pass | None |
| 5. | Marker intervals are selected appropriately using a small range. | Zooming in to a range of -1 to 1 in both x and y axis | Marker interval is 0.2, markers drawn on -1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8 and 1. | Marker interval was 0.2 but the markers were drawn on numbers such as -0.60000001, suggesting a floating-point rounding error. | Fail | Round markers to an appropriate degree of precision to prevent floating-point rounding errors. |

I resolved the bug in test 3 by rendering the text offset from the axis based on its width and height, so that all numbers will not be drawn onto the axis regardless of their size. The bug in test 5 was resolved by rounding the markers to the order of magnitude of the closest marker interval. The following code was added to the calculate markers function:

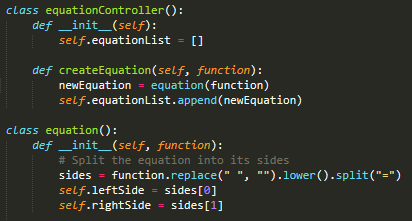


And the pygame blit calls in the drawGrid function were altered:



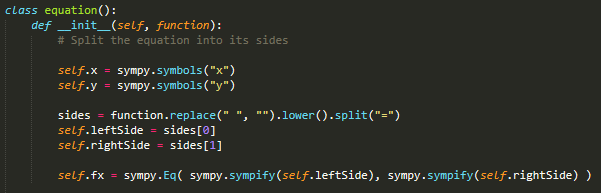


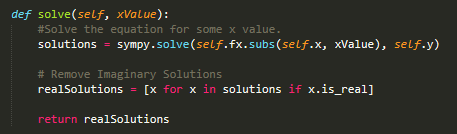
Now that the grid is functioning correctly I can starting developing the graphing functionality; this will be my next step as the tutorial and example graphs cannot be developed as they require the ability to draw graphs. The first step is to create an equation class and an equation controller class.



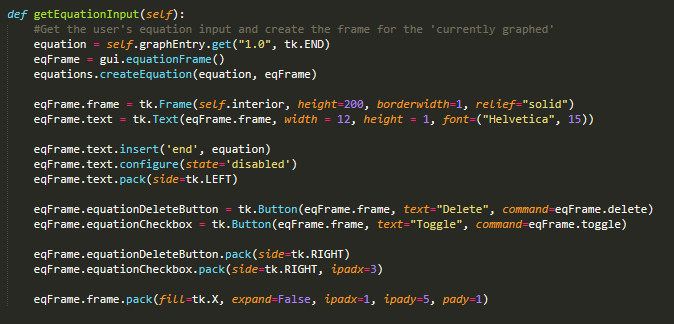
At this point I wanted to develop a function that would solve the equation for y given an x value, so that I could go on to develop the function that plots the functions. An obvious solution would be to try a brute-force method that evaluates both the left and right hand sides for many y-values, and when the left side of the equation equals the right, that is the correct y-value, however this method would be extremely resource consuming, even with optimisations, as it would have to check many thousands or potentially millions of y values to find a solution. This method would also not work for every equation, for example solving y2 = c (where c is a constant) as there would be two possible solutions, of which it would only find one.

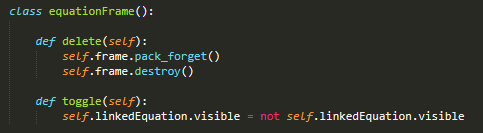
To solve this, I researched and attempted to implement algebraic computation to rearrange and solve any equation however I realised this would be very time consuming and difficult to implement, so I decided against it due to time limitations on this project. Instead, I will be using a python library called SymPy which allows for symbolic computation. I have written the solve function using SymPy; it works by substituting in the x value and then re arranging the equation for y. The outputs of this function can be used when I want to plot a graph, as the program can now identify x, y coordinates for points on a line. Here is the code:

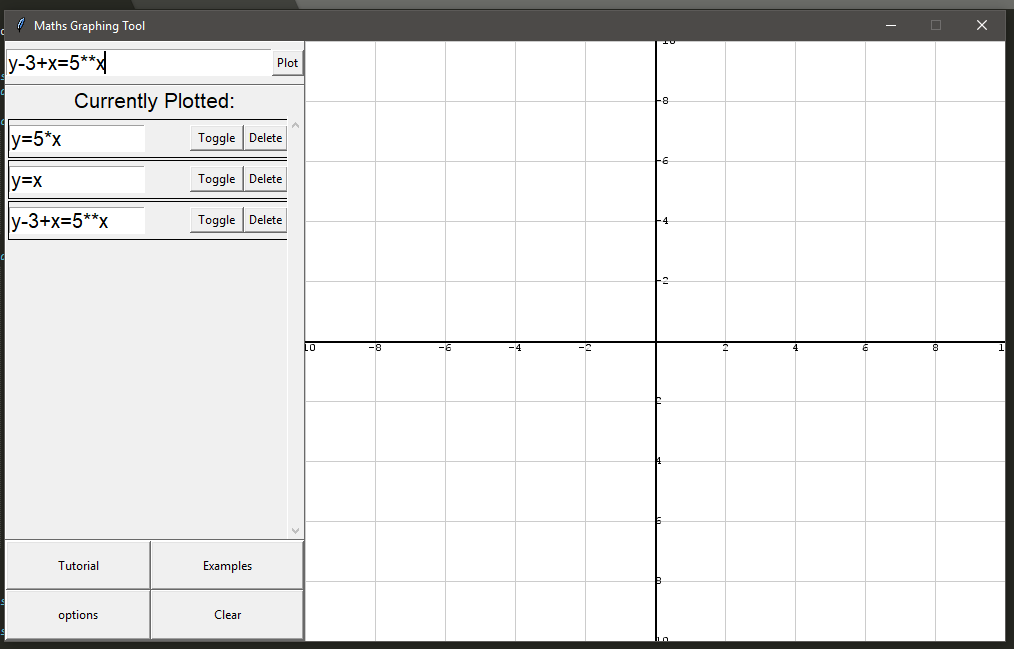




The graph input also needs to be implemented before I can add graph drawing functionality. When the ‘plot’ button is pressed, the getEquationInput function is called so the contents of the input field are fed into the ‘createEquation’ method of the equationController and a new equationFrame class is created. The equationFrame class holds the frame, delete button and toggle button within it, as well as having a reference to the equation it corresponds with. This class is necessary as otherwise the delete method would not know which equation to delete, or which equation to toggle with the toggle method. After the class is initialized the delete and toggle buttons are created, packed and assigned to the equationFrame object. Right now, I am not completely satisfied with the layout and positioning of the GUI however I believe it is more important to ensure the program functions correctly before I should focus on accessibility. Here is a screenshot of the code and the functioning input:

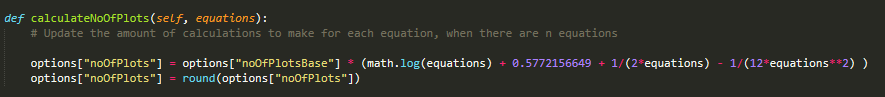


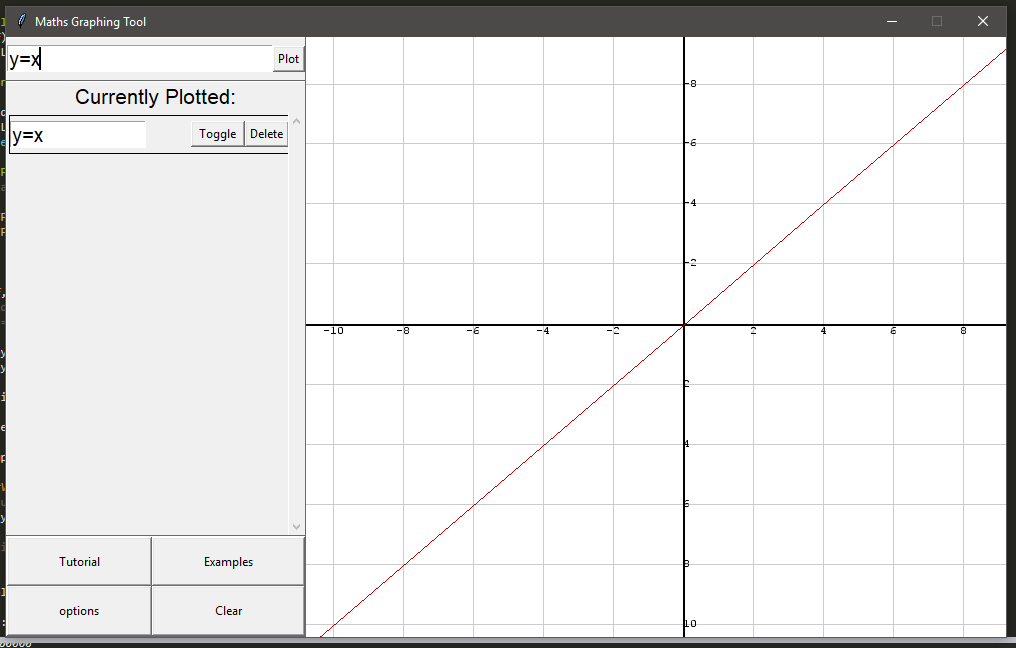




My next step will be to implement the ability for the program to draw graphs. The equation class already has a solve method, so I will use that at various x-values to get the corresponding y-values for an equation, then I will draw lines inbetween those points. It is important that this function is optimised well as it is likely the program will call it several hundred times in order to graph the equations as accurately as possible. Additionally, for each new equation that is added the amount of calculations that are needed would increase linearly, which could cause issues for slower computers or when many equations are being plotted. As high-performance was part of my success criteria, I will make the equation graphing so that for each new equation added, the number of calculations will increase logarithmically in order to balance accuracy and speed. Here is the code I wrote for drawing the graphs, and a screenshot of a graph being drawn:







### Test Table 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Input | Expected Result | Actual Result | Pass/Fail | Actions Needed |
| 6. | Equations can be entered into the program | y=5\*x | The equation is added to the currently plotted list and plotted | Equation was added to the list and plotted | Pass | None |
| 7. | Linear equations entered into the program are plotted correctly | Y=5\*x | The graph is rendered: | Rendered Correctly: | Pass | None |
| 8. | Polynomial equations are plotted correctly | Y=x\*\*2 | The graph is rendered: | Rendered Correctly: | Pass | None |
| 9. | Equations can be deleted from the currently plotted list | y=x entered as a plot, and ‘delete’ button is pressed | The equation is removed from the currently plotted list and is no longer plotted. | Equation was removed from list and wasn’t plotted | Pass | None |
| 10. | Equation’s visibility can be toggled off | y=x entered as a plot, and ‘Toggle’ button is pressed | The equation is not removed from the list, but is no longer plotted. | Equation wasn’t removed and wasn’t plotted | Pass | None |
| 11. | Equation’s visibility can be toggled back on again. | Y=x entered as a plot, and ‘toggle’ is pressed twice. | The equation is plotted again. | Equation was plotted | Pass | None |
| 12. | Equation’s being plotted with little to no delay. | y=x\*\*2 entered as a plot | The equation is plotted quickly, and does not slow down the program | The program slowed down a lot, making it almost unusable. | Fail | Optimisation to the graph drawing process. |
| 13. | Circle Equations can be plotted | x\*\*2 + y\*\*2 = 4 entered as a plot | The circle is plotted | The program crashed | Fail | Rewriting parts of the drawEquations method |

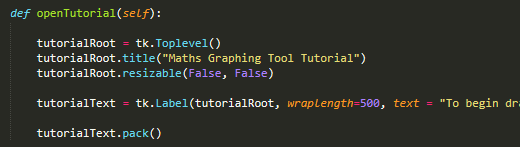
To optimise the program I looked over the steps the program takes when it plots the graph and I noticed that for each x-value it’s plotting, this line of code is running in the equation solve method.

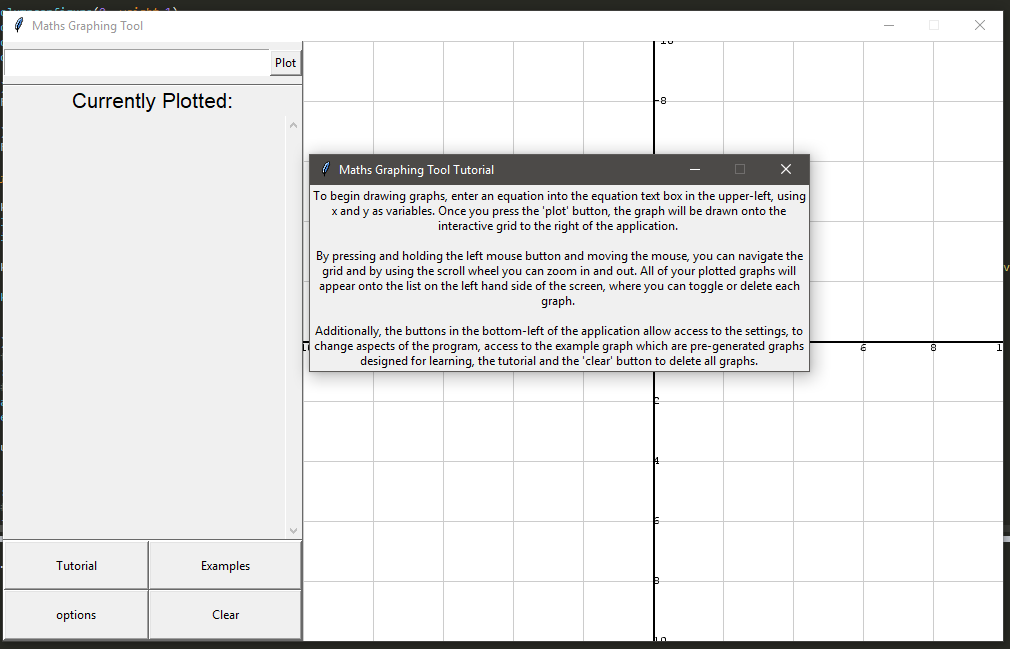


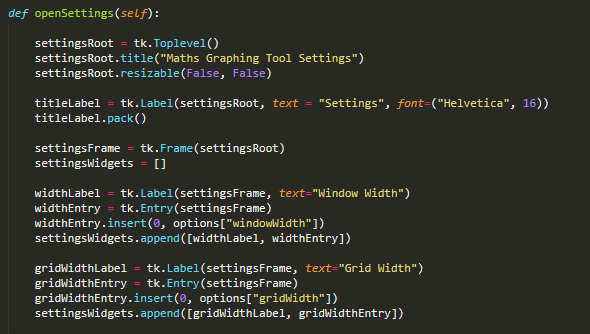
This line of code is rearranging the equation in terms of y so that an x-value can be substituted in, and a y-value found. When I measured the time for this line of code to run using the Python time library, the average time taken was 0.02 seconds, and could take 0.1 seconds when the user was panning the grid. When I looked at the source code of the sympy.solve function it was clear that it was a very compute-intense function, and so should be avoided being called often. I rewrote the equation.solve method so that the sympy.solve function is only called once, when the equation is created and to find a y-value, the program only has to substitute an x-value in. The time for the rewritten code to execute on average was 0.1 milliseconds. Here is the rewritten code:

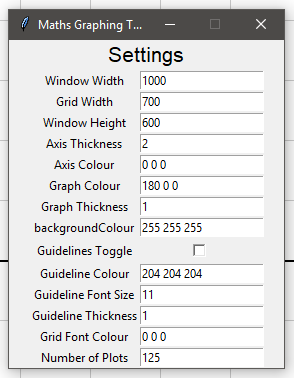


My next step is to implement the options, examples and tutorial as these are part of my ‘must’ success criteria. The tutorial will be a simple text tutorial instructing the user how to use the program; I believe that the design of the interface is intuitive enough for users to be able to use the program without a thorough tutorial however if I find that users are having difficulty during testing I can create a more in-depth tutorial. The tutorial screen is implemented by calling the openTutorial method when the tutorial button is pressed, which creates a new tkinter window and adds a label to it with the tutorial text:

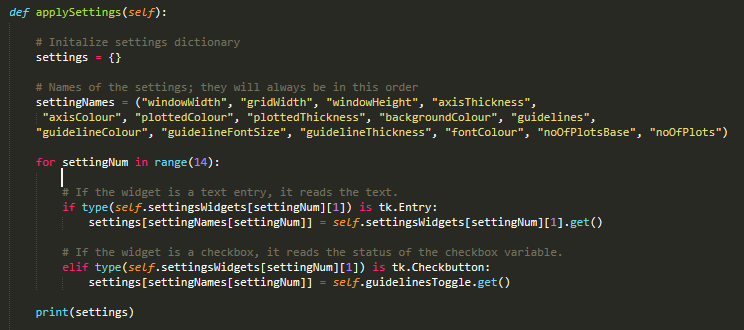




The options menu will use the .json format to store the settings in an external file which can be loaded every time the application launches. I will also be using this method to store the example graphs, and allow the user to save and load their own example graphs. I already designed the GUI for the options menu in my design; here is my implementation: 



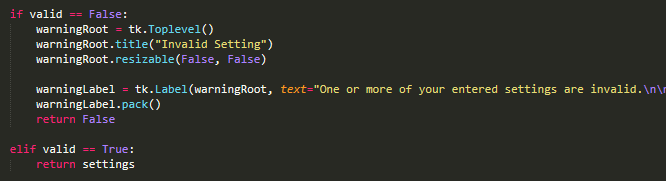
Each widget in the settings window is added to an array called ‘settingsWidgets’ as it allows me to easily iterate through each input to read the user’s input, and allows me to pack each widget iteratively. Now I will implement an ‘apply settings’ button. When this button is pressed it will call a function which will read the input of each setting, validate each input to ensure the program won’t crash and save these settings to a .json file. When the program is launched, it will read the settings in this json file and use those settings instead of the default. Here in my algorithm for reading the widget:

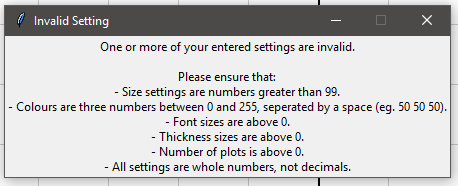


This algorithm will create a dictionary data structure for the settings; I have chosen to use a dictionary instead of an array or list as a dictionary is easily exported into a .json file using the python json library. The algorithm will read each widget and determine if it is an text entry field or a checkbox, and will read the input correspondingly. Next, I will create a function to validate the settings dictionary; if the settings are valid it will save them to a json file and if not an error message will be displayed. This is the validation algorithm:

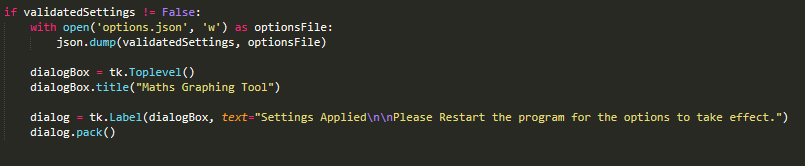


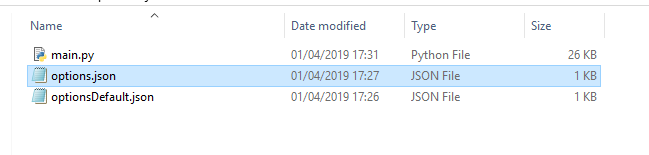


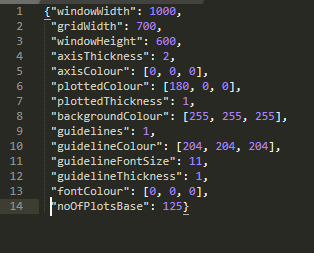




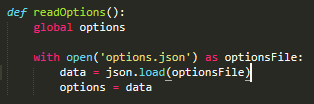
I have used python’s json library to be able to write to a json. The program will write to a file called ‘options.json’ within the directory it is in, and if the file does not exist it will create it and write to it. Additionally, the default options of the program will be stored in a file called ‘optionsDefault.json’ in the same directory, which will be read when the user presses the ‘restore to default’ button in the options menu. Here is the solution, and a screenshot of the options.json file:



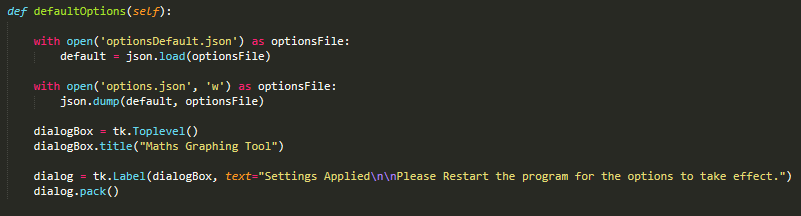


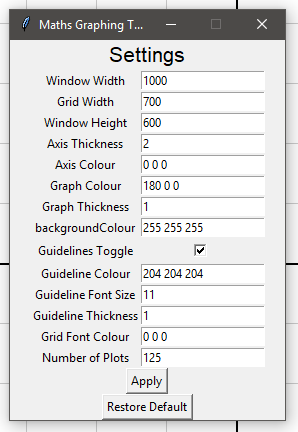


To read the file, I will use the json library again to convert the json format into a python dictionary which I can use as the options dictionary which I have been using. I have created a function named ‘readOptions’ which is called during initialization, which reads the options file. Here is my solution:



To restore the options to default, the program will simply read the default settings file and write the contents into the settings file. Here is the code and a screenshot of the finished options menu:





### Test Table 4

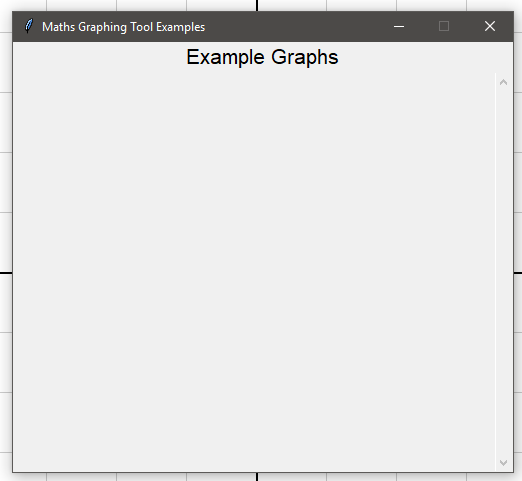
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Input | Expected Result | Actual Result | Pass/Fail | Actions Needed |
| 14. | The tutorial button functions correctly | Clicking the tutorial button | A new window is opened containing instructions | The window was opened, and contained instructions | Pass | - |
| 15. | The options button works correctly | Clicking the options button | The options window is opened | The options window opened. | Pass | - |
| 16. | The options window has the correct layout and options | Opening the options window | The window contains a list of options with entry boxes alongside them | The options window had the expected layout | Pass | - |
| 17. | The ‘apply’ button in the options window works correctly with a correct input | Options are changed correctly, and then ‘apply’ is pressed. | The new options are saved to the options.json file. | The options were saved in the correct format to the options.json file | Pass | - |
| 18. | The ‘apply’ button works correctly with an invalid input | Options are changed with an invalid input, and ‘apply’ is pressed. | The options are not saved, and an error message opens. | The options weren’t saved and the error message opened. | Pass | - |
| 19. | The options are read correctly from the options.json file and applied to the program on launch | The program is launched using altered options. | The program launches, applying the effects of the options. | The program did not apply the Window Width, Grid Width and Window Height options correctly, but others were applied correctly | Fail | Call the readOptions function before the gui module is initialized. |
| 20. | The ‘Restore Default’ button works correctly. | The ‘Restore Default’ button is pressed in the options menu. | The program changes the options.json file back to the default options specified in defaultOptions.json | The options.json file was altered correctly. | Pass | - |

This phase of the testing was mostly successful, the only failed test was resolved very easily as it was simply a matter of two functions being called in the wrong order. The next must criteria I will complete is the saving and loading of user’s graphs and the pre-generated example graphs. Both these features will use the same functions to load graphs from json files. After these features are implemented I can begin implementing the ‘should’ criteria.

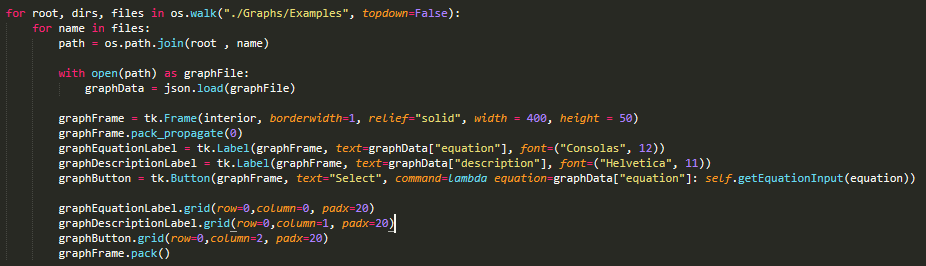
Firstly, I will work on the GUI for the saving and loading of graphs and the loading of pre-made examples. The GUI for loading these will use a similar design as the algorithm list, which can expand with a scrollbar to accommodate new items being added to the list, as there may be too many user-saved graphs to fit on the screen at once. Here is the solution I have created:



And here is the screenshot of the examples window:

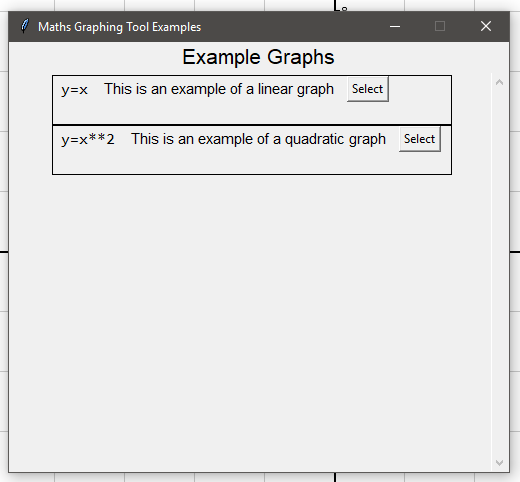


The example graphs will be stored in /Graphs/Examples and the user-saved graphs will be stored under /Graphs. The graphs will be stored in a json format, and will contain the equation and description of the graph as strings. The load GUI and examples GUI will read each json file in the directories and load them into the GUI. The user will then select the graph they want to load, and the equation added similar to a user-entered program. Here is the code for this portion of the program:

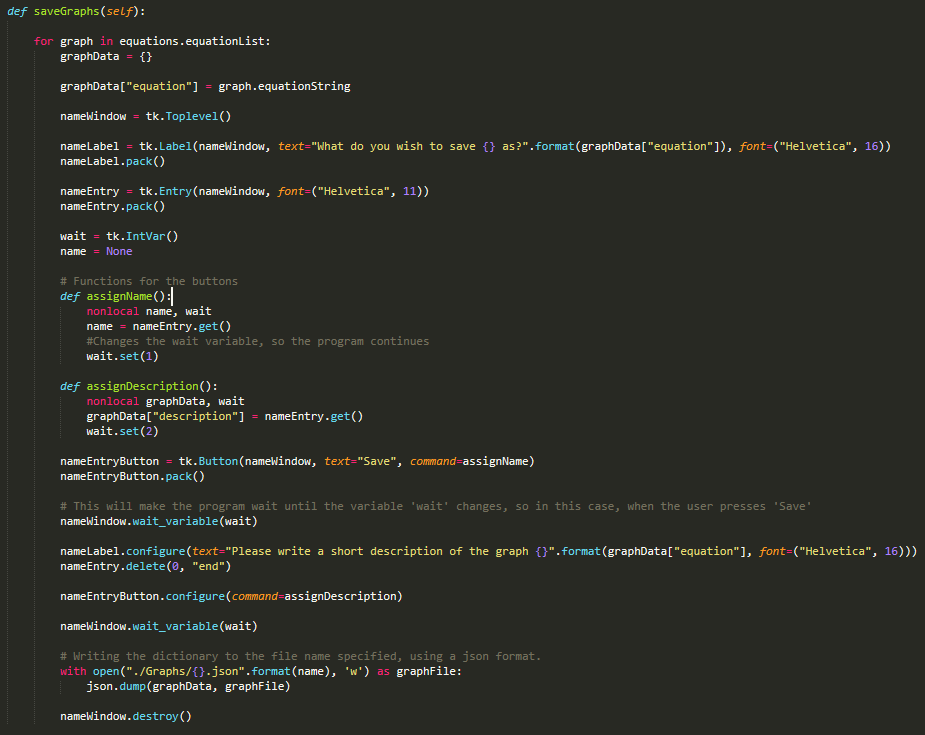


I have used a lambda function to assign the commands to the buttons: this is preferable to using a regular function as I don’t have to define any new data structures or functions to do so. The lambda function calls the method getEquationInput using the contents of ‘equation’ in the graphData dictionary at the time of the lambda function’s creation.

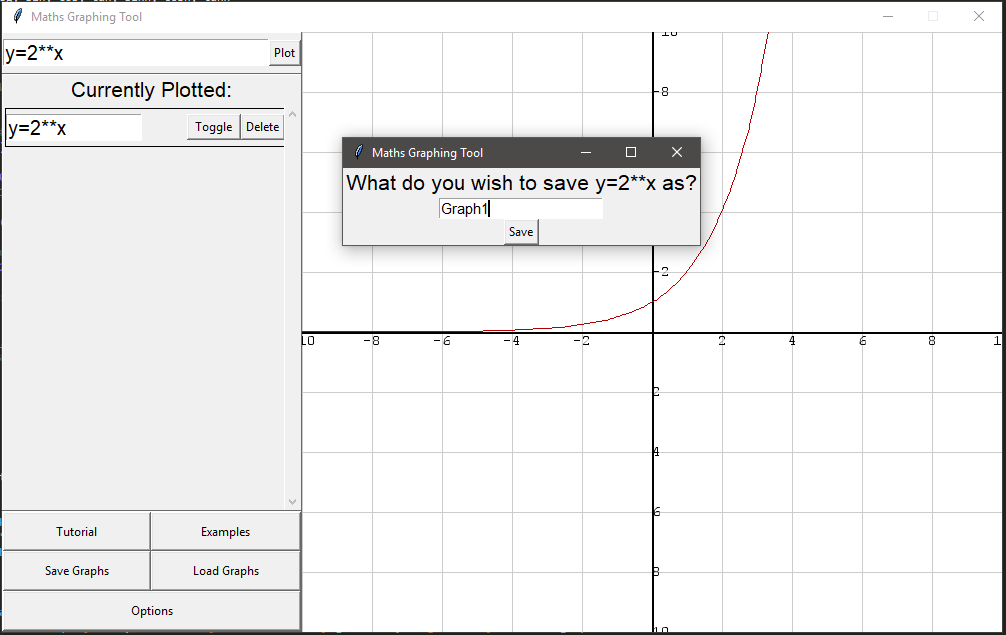
And here is the examples menu, using two test graphs:

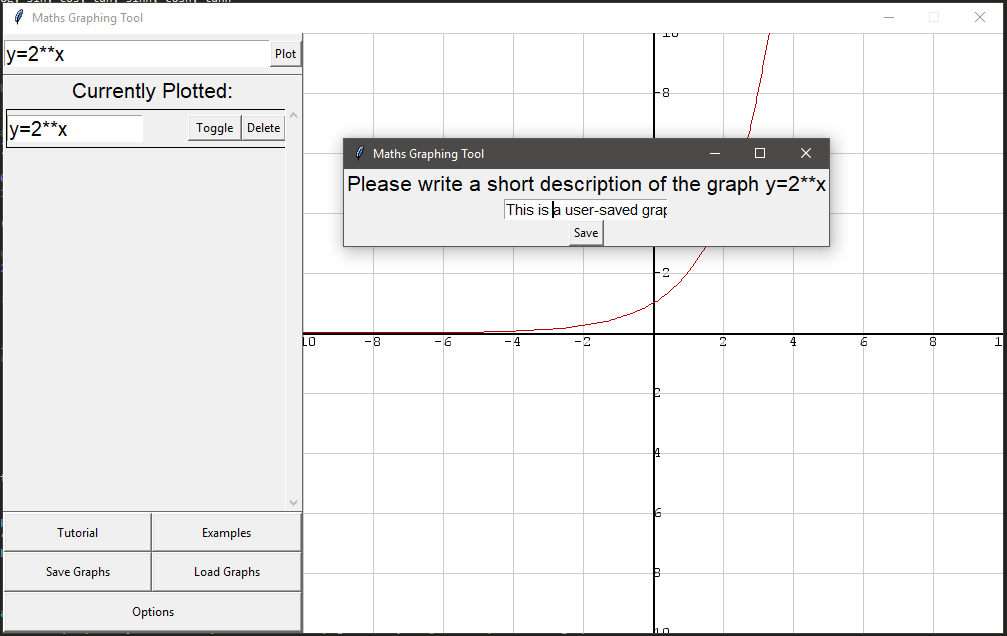


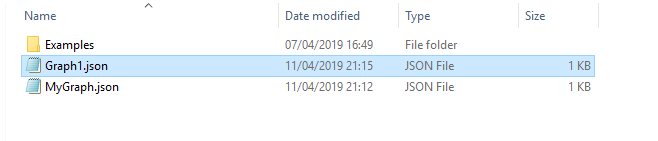
To implement the saving graphs feature, I will create a simple GUI to allow the user to enter a filename and a description of the graph; the program will then write the equation in string form and the description of the equation to the file using the json library. Here is the code:



And here are screenshots of the saving process:

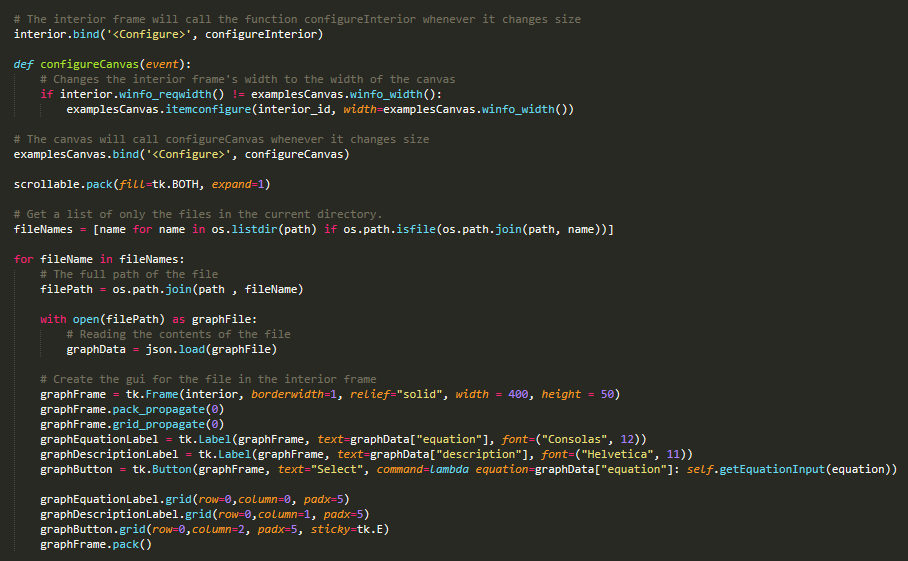


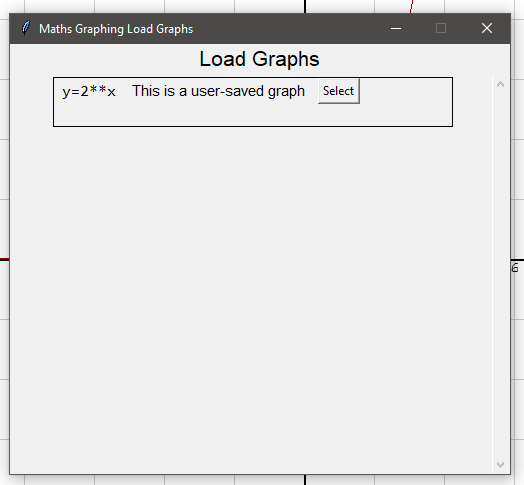




To implement loading user-made graphs, it will uses the exact same logic as loading example graphs, but will use a different path to locate files. To avoid repeating myself, I rewrote the openExamples function to include a parameter for the path, and renamed it to loadGraphs as this name better represents the function’s use. The examples button calls the function using the path ‘./Graphs/Examples’ whereas the load button calls the function using the path ‘./Graphs/’. Here is the code:





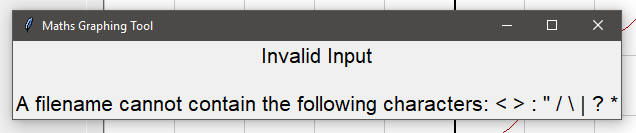


The only ‘must’ criteria left is that invalid inputs entered into the program do not crash the program. The settings menu already includes checks for invalid inputs, however the graph saving feature and the graph entry feature do not. The simplest solution would be to ban certain characters from being entered into input fields; this solution works well for the saving graphs feature as there are certain characters which cannot be in file names, which can be banned. However, for the graph entry feature this solution does not work as many characters are valid in certain functions (for example ‘y=log(x)’ or ‘y=sin(x)’) however would be invalid by themselves (for example ‘y=ax’).

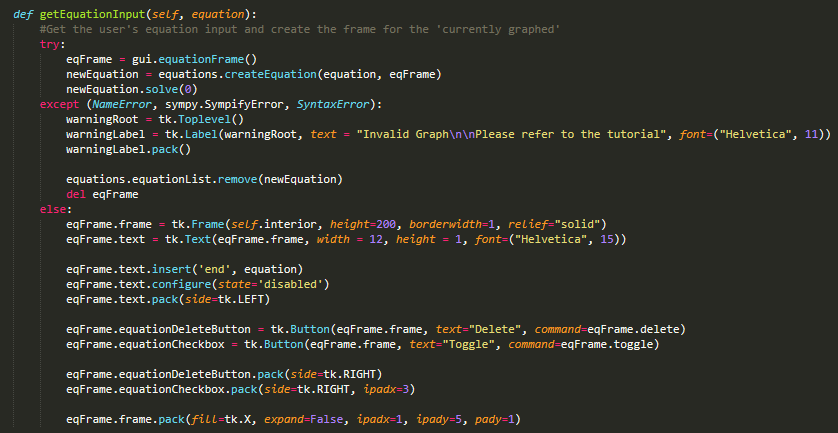
Here is the code for preventing invalid input in the saving feature:



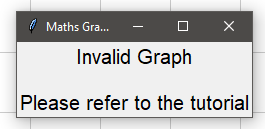
And the warning message that is produced with an invalid input:



The graph entry field will be validated using a try, catch statement, by catching the exception *NameError* which is raised when the user types in invalid variable (for example ‘y = a + x’), catching the exception *sympy.SympifyError* which is raised when an invalid input is entered into the sympify function, and the exception *SyntaxError* which is raised when the user types an equation using invalid syntax (for example ‘y = 5x’ rather than ‘y=5\*x’). Here is the code:



And a screenshot of the error message:



# Testing: Evaluative

## Final Testing Evidence

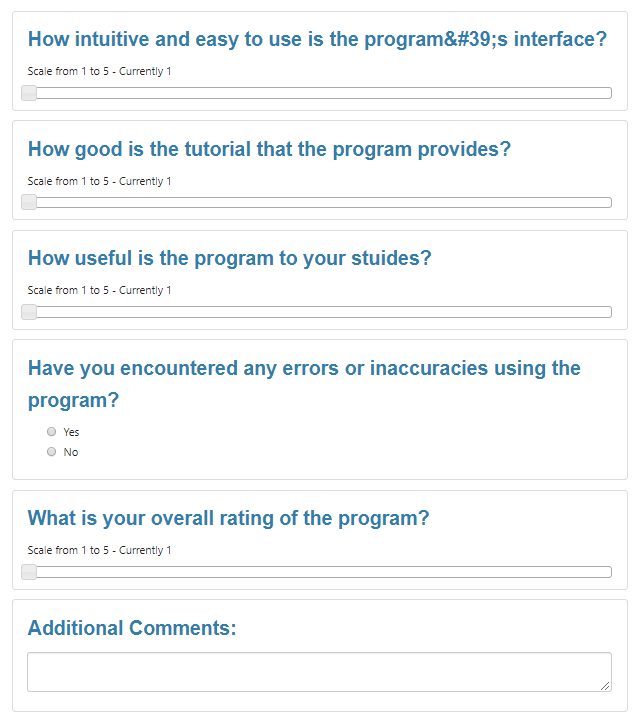
The program is fully developed and I will be using the post-developmental test data I created in my design section to fully test my program.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Input Type | Test Input | Expected Result | Actual Result | Pass/Fail | Actions Needed |
|  | Running the program | Valid Data | Program Started | The program starts | The program starts | Pass | - |
|  | Placement of GUI | Valid Data | Program started | The GUI is placed correctly | The GUI is placed correctly | Pass | - |
|  | GUI responsiveness | Valid Data | Clicking each of the pressable GUI elements | The program responds within 100ms | The program responds within 100ms | Pass | - |
|  | Interactive Grid Rendering | Valid Data | Program Started | The guidelines, guideline markers and axis are all rendered correctly | Guidelines, guideline markers and axis are all rendered correctly | Pass | - |
|  | Interactive grid panning | Valid Data | Interactive Grid clicked and dragged | The interactive grid pans according to the movement of the mouse | The interactive grid panned according to the movement of the mouse | Pass | - |
|  | Linear Graph Rendering | Valid Data | ‘y=x’ in graph entry | Graph rendered: | Rendered Correctly: | Pass | - |
|  | Use of coefficients in graph | Valid Data | ‘y=5\*x’ in graph entry | Graph rendered: | Rendered Correctly: | Pass | - |
|  | Polynomial Graph Rendering | Valid Data | ‘y=x\*\*2’ in graph entry | Graph Rendered: | Rendered correctly: | Pass | - |
|  | Use of addition in graph | Valid Data | ‘y=x + 3’ in graph entry | Graph rendered: | Rendered Correctly: | Pass | - |
|  | Use of sin function | Valid Data | ‘y=sin(x)’ | Graph rendered: | Rendered Correctly: | Pass | - |
|  | Use of cos function | Valid Data | ‘y=cos(x)’ | Graph rendered: | Rendered Correctly: | Pass | - |
|  | Graphs with y and x on one side | Valid Data | ‘y+x=15’ | Graph Rendered: | Rendered Correctly: | Pass | - |
|  | Reciprocal Equations | Valid Data | ‘y=1/x’ | Graph rendered: | Rendered Correctly: | Pass | - |
|  | Graph using unknown variable | Invalid Data | ‘y=a’ | Error Message produced | Error Message Produced | Pass | - |
|  | Graph using invalid syntax | Invalid Data | ‘y=3x’ | Error message produced | Error Message Produced | Pass | - |
|  | Graph using exponential | Valid Data | ‘y=2\*\*x’ | Graph rendered: | Graph rendered correctly but crashed when zoomed out | Fail | Prevented the y-coordinate from becoming too large as it causes the pygame draw line function to crash |
|  | Graph using sinh | Valid Data | ‘y=sinh(x)’ | Graph rendered: | Rendered Correctly: | Pass | - |
|  | ‘Currently plotted’ | Valid Data | A graph is entered | The graph appears in the ‘currently plotted’ list | The graph appeared in the ‘currently plotted’ list | Pass | - |
|  | Graph ‘delete’ button | Valid Data | Delete button is pressed | Graph deleted from list and not rendered. | Graph deleted and not rendered | Pass | - |
|  | Graph ‘toggle’ button | Valid Data | Toggle button is pressed | Graph visibility toggled but not deleted from list | Graph was no longer rendered but was in the list | Pass | - |
|  | Tutorial Button | Valid Data | Tutorial Button Pressed | The tutorial is opened | Tutorial was opened | Pass | - |
|  | Examples Button | Valid Data | Examples button pressed | The examples are opened | Examples were opened | Pass | - |
|  | Save Graphs Button | Valid Data | ‘Save’ button pressed | The save graphs window is opened | Saving window was opened | Pass | - |
|  | Load Graphs button | Valid Data | ‘Load’ button pressed | The load graphs window is opened | Load graphs window opened | Pass | - |
|  | Options Button | Valid Data | Options button pressed | The options window opens | The options window is opened | Pass | - |
|  | Options menu reads current options | Valid Data | Options menu opened | The options fields contain the current options in the options.json file | The options fields contained the correct values | Pass |  |
|  | Options menu saves changes | Valid Data | Options are changed with valid values | The changed options are saved to options.json in the correct format | Options were saved correctly | Pass | - |
|  | Default Options button | Valid Data | ‘Restore Default’ button pressed | The options in optionsDefault.json are saved into options.json | Default options saved correctly | Pass | - |
|  | Invalid Options Entered, integer fields | Invalid Data | String values entered into integer fields | An error message opens instructing the user how to enter options correctly | Error message was opened correctly | Pass | - |
|  | Invalid Options Entered, Colour fields | Invalid Data | Colour values not entered in R G B form | An error message opens instructing the user how to enter options correctly | Error message opened correctly | Pass | - |
|  | WindowWidth Option | Valid Data | ‘1600’ in WindowWidth option | After program restarts, window is 1600 pixels wide | Window was 1600 pixels wide | Pass | - |
|  | GridWidth option | Valid Data | ‘1400’ in windowWidth option, ‘1200’ in gridWidth option | After program restarts, grid is 1200 pixels wide | Grid was 1200 pixels wide | Pass | - |
|  | GridWidth Option with invalid Data | Invalid Data | ‘1000’ in windowWidth and ‘1400’ in gridWidth | An error message opens instructing the user how to enter options correctly | Error message opened correctly | Pass | - |
|  | windowHeight Option | Valid Data | ‘1000’ in windowHeight | After program restarts, window is 1000 pixels high | Window was 1000 pixels high | Pass | - |
|  | axisThickness Option | Valid Data | ‘5’ in axisThickness | After program restarts, axis is 5 pixels thick | Axis was 5 pixels thick | Pass | - |
|  | axisColour option | Valid Data | ‘0 255 0’ in axisColour | After program restarts, axis is green coloured | Axis was green colour | Pass | - |
|  | plottedColour options | Valid Data | ‘0 255 0’ in plottedColour | After program restarts, plotted graphs are green | Graphs were green | Pass | - |
|  | plottedThickness option | Valid Data | ‘5’ in plottedThickness | After program restarts, plotted graphs are 5 pixels thick | Graphs were 5 pixels thick | Pass | - |
|  | backgroundColour option | Valid Data | ‘0 255 0’ in backgroundColour | After program restarts, background is blue | Background was blue | Pass | - |
|  | Guidelines option | Valid Data | Guidelines toggled off | After program restarts, guidelines are not visible | Guidelines were not visible | Pass | - |
|  | guidelineColour option | Valid Data | ‘0 255 0’ in guidelineColour | After program restarts, guidelines are blue | Guidelines were blue | Pass | - |
|  | guidelineFontSize option | Valid Data | ‘20’ in guidelineFontsize | After program restarts, guideline font was size 20 | Font size changed correctly | Pass | - |
|  | guidelineThickness option | Valid Data | ‘5’ in guidelineThickness option | After program restarts, guidelines are 5 pixels thick | Guidelines were 5 pixels thick | Pass | - |
|  | fontColour option | Valid Data | ‘0 255 0’ in fontColour | After program restarts, grid font is blue | Grid font was blue | Pass | - |
|  | noOfPlots option | Valid Data | ‘200’ entered in noOfPlots option | After program restarts, the base number of plots used to plot graphs is 200 | Base number of plots was 200 | Pass | - |
|  | Examples description | Valid Data | Example button pressed | The description of the example is read from the example file | Description read correctly | Pass | - |
|  | Examples loading | Valid Data | Load example button pressed | The example was read from the file and rendered on the screen. | Example loaded correctly | Pass | - |
|  | Graph Saving | Valid Data | Graph entered and ‘save graphs’ button pressed | The equation and description are saved to the file under the entered filename | File saved correctly | Pass | - |
|  | Graph Saving Using Invalid Filename | Invalid Data | Invalid Filename entered while saving graph | An error message opens | Error message opened correctly | Pass | - |
|  | Graph Loading Description | Valid Data | Load Graphs button pressed | Description loaded from graph files | Description loaded correctly | Pass | - |
|  | Graph Loading | Valid Data | Load graph button pressed | Equations are loaded correctly from graph files | Equation loaded correctly | Pass | - |

## Usability Testing

To perform usability testing I created a survey to gather the opinion of my stakeholders. Using the results of the survey I can make an accurate evaluation and make changes to aspects of the program to my stakeholders liking. I sent the survey to 10 stakeholders and they all gave a response.

Here is a screenshot of the survey:



Here are the results of the survey:

### Survey Analysis

The program’s interface was scored at 3.8/5 on average, this is a good score and I believe this shows the success criteria ‘Intuitive, Easy to use interface’ has been fulfilled.

The tutorial question received 2.5/5 on average, indicating that most users found the mediocre. If I given a greater amount of development time, I would design an interactive tutorial that guides the user through the features of the program.

The usefulness of the program was scored at 3.2/5 on average; this score is good as the survey was answered by mostly A-Level students however I feel that the usefulness of the program was impacted by the inability to graph certain equations, such as circles.

No users reported that they encountered errors or inaccuracies using the program, which is very important as it shows my testing was successful.

Overall, the program was rated at 3.3.

# Evaluation

## Evaluating Success Criteria

This table shows the completion of success criteria:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criteria | Category | Met | Evidence | Comments |
| Grid can be panned across | Must | Fully met | See test table 1 and post-development testing. The test results show the criteria has been met. | - |
| Grid can be zoomed in and out | Must | Fully met | See test table 1 and post-development testing. The test results show the criteria has been met. | - |
| Coordinate Axis appears and can be seen easily | Must | Fully met | See test table 2 and post-development testing. The test results show the criteria has been met. | - |
| Coordinate Axis has labels | Must | Fully met | See test-table 2 and post-development testing. The test results show the criteria has been met. | - |
| Labels dynamically adjust to the level of zoom | Must | Fully met | See test-table 2 and post-development testing | - |
| Users can enter graphs | Must | Fully met | See test-table 3 and post-development testing. The test results show the criteria has been met. | - |
| Equations inputted are drawn accurately with little to no delay | Must | Fully met | See test-table 3. The test results show the criteria has been met. | - |
| Intuitive, easy to use interface | Must | Fully met | See usability testing. The survey shows the criteria has been met | - |
| A tutorial that explains how the program is used | Must | Fully met | See usability testing and test-table 4. The test results of usability testing show the criteria has been met | - |
| The ability to save and load graphs to local storage | Must | Fully met | See post-development testing. The test results show the criteria has been met. | - |
| Pre-created educational demonstrations to teach users about graphs | Must | Fully met | See post-development testing. The test results show the criteria has been met. | - |
| A settings menu to adjust certain aspects of the application | Must | Fully met | See test table 4. The test results show the criteria has been met. | - |
| Invalid Inputs entered into entry fields do not crash the program | Must | Fully met | See post-development testing. The test results show the criteria has been met. | - |
| Points of interest highlighted on grid | Should | Not Met | - | Not met due to time constraints. This criteria could be achieved easily in further development as the current code facilitates this feature. |
| The ability to plot trigonometric, logarithmic and exponential functions | Should | Fully met | See post-development testing. The test results show the criteria has been met. |  |
| The ability to plot circles in the form (x-a)2 + (y-b)2 = r2 | Should | Not Met | - | Not met due to time constraints. This criteria could be achieved in futher development but would require the equation solving to be rewritten as this feature is not facilitated |
| Export graphs into a pdf document | Could | Not Met | - | Not met due to time constraints. This criteria could be met as pygame includes a feature to take screenshot of pygame regions, which could then be exported to a pdf document |
| The ability to plot polar graphs | Could | Not met | - | Not met due to time constraints. This criteria would be difficult to meet in further development as polar graphs are entered, store and plotted differently to Cartesian graphs. |
| The ability to use degrees instead of radians | Could | Not met | - | Not met due to time constraints. This criteria could be achieved easily by converting the degrees input into radians before calculating the answer |
| The ability to plot hyperbolic functions | Could | Fully met | See post-development testing. The test results show the criteria has been met. | - |

Additionally, I will be assessing if the program meets the performance criteria set:

|  |  |  |
| --- | --- | --- |
| Criteria | Met | Evidence |
| Program does not use more than 1gb of RAM | Fully met |  |
| Program does not use over 20% of the CPU during use | Fully met | See above |
| Program does not use more than 100mb of hard disk space | Fully met |  |

## Evaluating Usability Features

The usability features I designed in the design section have mostly been successful; the GUI was rated 3.8/5 on average in the user-feedback survey so I would consider this feature very successful. The options menu was also very successful as almost all parts of the GUI can be altered to the user’s liking and the user can restore default options.

The tutorial I would consider to be partially successful, as it was rated at 2.5/5 on average in the user-feedback survey. The reason for the tutorial’s partial success is the inclusion of the tutorial and the error messages which both help guide the user through the use of the program. The reason the tutorial did not achieve a full success is the lack of intractability as the tutorial consists of a large paragraph telling the user how to use the program. In further development I would consider including pictures in the tutorial or creating an interactive tutorial that guides the user through using the program.

## Limitations

Hardware limitations are 33KB of free space on secondary storage. The program’s saved graphs take up additional space and a limit is not included on the amount of user-saved graphs so if many graphs are stored the space limitations would be increased. A user-saved graph file consists of only text and will not take up more space than approximately 100 Bytes. This limitation could be overcome by compressing the user-saved graphs and the example graphs so that they use less space.

The program will use around 50mb of RAM on average but will use more if many graphs are plotted. The program uses a small amount of the CPU using default settings, however the number of plots setting is high or many graphs are plotted the program may use a significant portion of the CPU. The hardware limitations could be overcome by refactoring the code and improving the efficiency of the computationally demanding parts.

Software limitations are Python 3 using the default libraries and the pygame, numpy and sympy libraries. The program has only been tested successfully using Windows 10.

Another limitation in my solution is that prior mathematic knowledge of graphs is required before the use of the program. This limitation could be overcome by implementing tutorials and examples designed specifically for users without prior knowledge.

## Maintenance

The maintenance of the program would be performed mostly by other developers as the code of the program requires changes however users could maintain the program by creating their own example graphs.

### Further Development

There are some new features that would be introduced during maintenance that couldn’t be completed during development. Firstly, I would introduce a more suitable tutorial for the program. The tutorial would specifically show the use which parts of the GUI do what, and guide the user through creating their first graph. Moreover, a user manual could be created to give a much more in-depth tutorial for students of A-Level Maths and Further Maths.

Additionally, I would implement the feature to graph circle equations. This would require the drawEquations method and the way graphs are stored to be rewritten as the current system assumes that each y-value only has one x-value as a solution, which is not the case for circle graphs.

Other features that could be developed during maintenance were mentioned during the success criteria evaluation and include highlighting points of interest, exporting graphs to pdf documents and plotting polar graphs.

### Code Maintenance

There are parts of the code that I was unsatisfied with after development. The main part being the settings menu, as each setting’s frame and entry box was coded in manually, which was time consuming and makes adding new settings difficult. In the future, I would create an algorithm that automatically reads the options in the options dictionary and create the appropriate GUI dynamically in the options menu. This would be advantageous as it would make adding new options be very easy and so would make further maintenance easier.

# Appendix

## Code

**import** **pygame**

**import** **tkinter** **as** **tk**

**import** **os**

**import** **sys**

**import** **platform**

**import** **numpy**

**import** **math**

**from** **math** **import** sqrt, log, sin, cos, tan, sinh, cosh, tanh

**import** **sympy**

**import** **json**

options = {

"windowWidth": 1000, *# Width of the window, in pixels.*

"gridWidth": 700, *# Width of the grid inside the window in pixels*

"windowHeight": 600, *# Height of window in pixels*

"axisThickness": 2, *# Thickness of the axis guides in pixels*

"axisColour": (0,0,0), *# Colour of the axis guides (R, G, B)*

"plottedColour": (180,0,0), *# Colour of the plotted graphs (R, G, B)*

"plottedThickness": 1, *# Thickness of the plotted graphs in pixels*

"backgroundColour": (255,255,255), *# Colour of the background of the grid (R, G, B)*

"guidelines": 1, *# Guidelines toggle on/off*

"guidelineColour": (204, 204, 204), *# Colour of the guidelines ( R, G, B )*

"guidelineFontSize": 11, *# Font size of guideline markers*

"guidelineThickness": 1, *# Thickness of guidelines in pixels*

"fontColour": (0, 0, 0), *# Colour of the guideline marker font ( R, G, B )*

"noOfPlotsBase": 125, *# Base sampling rate of plotted graphs ( Higher = More Accurate )*

"noOfPlots": 125 *# Sampling rate adjusted for the amount of plotted graphs*

}

**class** **guiController**():

**def** \_\_init\_\_(self):

self.windowWidth = options["windowWidth"]

self.gridWidth = options["gridWidth"]

self.windowHeight = options["windowHeight"]

*# Initialize the tkinter window*

self.root = tk.Tk()

self.root.title("Maths Graphing Tool")

self.root.resizable(False, False)

*# Initialize the frames for the grid and the menu*

self.gridFrame = tk.Frame(self.root, width = self.gridWidth, height = self.windowHeight)

self.gridFrame.pack(side = tk.RIGHT, fill="both", expand=True)

self.menuFrame = tk.Frame(self.root, width = self.windowWidth-self.gridWidth, height = self.windowHeight)

self.menuFrame.pack(side = tk.LEFT, fill="both", expand=True)

self.menuFrame.grid\_propagate(0)

*# Initialize the graph entry box and 'plot' button*

self.graphEntryFrame = tk.Frame(self.menuFrame, relief=tk.RAISED, borderwidth=1)

self.graphEntryFrame.grid(row=0,column=0,sticky=tk.N+tk.S+tk.E+tk.W)

self.graphEntry = tk.Text(self.graphEntryFrame, width = 10, height = 1, font=("Helvetica", 15))

self.graphEntry.pack(side=tk.LEFT, fill=tk.X, expand=1)

self.graphEntryButton = tk.Button(self.graphEntryFrame, text="Plot", command=self.newEquation)

self.graphEntryButton.pack(side=tk.RIGHT)

*# Initalize the frame for the 'currently plotted' list*

self.plottedFrame = tk.Frame(self.menuFrame, relief=tk.RAISED, borderwidth=1)

self.plottedFrame.grid(row=1,column=0, sticky=tk.N+tk.S+tk.E+tk.W)

self.plottedHeader = tk.Label(self.plottedFrame, text="Currently Plotted:", font=("Helvetica", 16)).pack(side=tk.TOP)

self.scrollable = tk.Frame(self.plottedFrame)

*# Initalize the scrollbar and canvas*

self.plottedScrollbar = tk.Scrollbar(self.scrollable, orient=tk.VERTICAL)

self.plottedScrollbar.pack(fill=tk.Y, side=tk.RIGHT, expand=0)

self.plottedCanvas = tk.Canvas(self.scrollable, yscrollcommand=self.plottedScrollbar.set) *# The yscrollcommand attribute links the canvas to the scrollbar*

self.plottedCanvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=1)

self.plottedScrollbar.config(command=self.plottedCanvas.yview)

*# Set the inital scroll values for the canvas*

self.plottedCanvas.xview\_moveto(0)

self.plottedCanvas.yview\_moveto(0)

*# The interior frame is created in the canvas, and will contain the widgets which will be moved when the user scrolls*

self.interior = tk.Frame(self.plottedCanvas)

self.interior\_id = self.plottedCanvas.create\_window(0, 0, window=self.interior, anchor=tk.NW)

**def** configureInterior(event):

*# Changes the size of the interior frame and the scrollable region of the canvas*

size = (self.interior.winfo\_reqwidth(), self.interior.winfo\_reqheight())

self.plottedCanvas.config(scrollregion="0 0 **%s** **%s**" % size)

**if** self.interior.winfo\_reqheight() != self.plottedCanvas.winfo\_width():

self.plottedCanvas.config(width=self.interior.winfo\_reqwidth())

*# The interior frame will call the function configureInterior whenever it changes size*

self.interior.bind('<Configure>', configureInterior)

**def** configureCanvas(event):

*# Changes the interior frame's width to the width of the canvas*

**if** self.interior.winfo\_reqwidth() != self.plottedCanvas.winfo\_width():

self.plottedCanvas.itemconfigure(self.interior\_id, width=self.plottedCanvas.winfo\_width())

*# The canvas will call configureCanvas whenever it changes size*

self.plottedCanvas.bind('<Configure>', configureCanvas)

self.scrollable.pack(fill=tk.BOTH, expand=1)

*# The options, tutoiral, examples and clear buttons are initalized*

self.optionsFrame = tk.Frame(self.menuFrame, relief=tk.RAISED, borderwidth=1)

self.optionsFrame.grid(row=2,column=0, sticky=tk.N+tk.S+tk.E+tk.W)

self.tutorialButton = tk.Button(self.optionsFrame, text="Tutorial", command=self.openTutorial)

self.examplesButton = tk.Button(self.optionsFrame, text="Examples", command=**lambda** path="./Graphs/Examples": self.loadGraphs(path))

self.optionsButton = tk.Button(self.optionsFrame, text="Options", command=self.openSettings)

self.saveButton = tk.Button(self.optionsFrame, text="Save Graphs", command=self.saveGraphs)

self.loadButton = tk.Button(self.optionsFrame, text="Load Graphs", command=**lambda** path="./Graphs/": self.loadGraphs(path))

self.tutorialButton.grid(row=0,column=0,sticky=tk.N+tk.S+tk.E+tk.W)

self.examplesButton.grid(row=0,column=1,sticky=tk.N+tk.S+tk.E+tk.W)

self.optionsButton.grid(row=2,column=0, columnspan=2,sticky=tk.N+tk.S+tk.E+tk.W)

self.saveButton.grid(row=1,column=0,sticky=tk.N+tk.S+tk.E+tk.W)

self.loadButton.grid(row=1,column=1,sticky=tk.N+tk.S+tk.E+tk.W)

self.menuFrame.columnconfigure(0, weight=1)

self.menuFrame.rowconfigure(0, weight=1)

self.menuFrame.rowconfigure(1, weight=10)

self.menuFrame.rowconfigure(2, weight=3)

**for** i **in** range(3):

self.optionsFrame.rowconfigure(i, weight=1)

**for** i **in** range(2):

self.optionsFrame.columnconfigure(i, weight=1)

**def** openTutorial(self):

tutorialRoot = tk.Toplevel()

tutorialRoot.title("Maths Graphing Tool Tutorial")

tutorialRoot.resizable(False, False)

tutorialText = tk.Label(tutorialRoot, wraplength=500, text = "To begin drawing graphs, enter an equation into the equation text box in the upper-left, using x and y as variables. Once you press the 'plot' button, the graph will be drawn onto the interactive grid to the right of the application. **\n\n**By pressing and holding the left mouse button and moving the mouse, you can navigate the grid and by using the scroll wheel you can zoom in and out. All of your plotted graphs will appear onto the list on the left hand side of the screen, where you can toggle or delete each graph. **\n\n**Additionally, the buttons in the bottom-left of the application allow access to the settings, to change aspects of the program, access to the example graph which are pre-generated graphs designed for learning, the tutorial and the 'clear' button to delete all graphs.")

tutorialText.pack()

**def** openSettings(self):

settingsRoot = tk.Toplevel()

settingsRoot.title("Maths Graphing Tool Settings")

settingsRoot.resizable(False, False)

titleLabel = tk.Label(settingsRoot, text = "Settings", font=("Helvetica", 16))

titleLabel.pack()

settingsFrame = tk.Frame(settingsRoot)

self.settingsWidgets = []

widthLabel = tk.Label(settingsFrame, text="Window Width")

widthEntry = tk.Entry(settingsFrame)

widthEntry.insert(0, options["windowWidth"])

self.settingsWidgets.append([widthLabel, widthEntry])

gridWidthLabel = tk.Label(settingsFrame, text="Grid Width")

gridWidthEntry = tk.Entry(settingsFrame)

gridWidthEntry.insert(0, options["gridWidth"])

self.settingsWidgets.append([gridWidthLabel, gridWidthEntry])

heightLabel = tk.Label(settingsFrame, text="Window Height")

heightEntry = tk.Entry(settingsFrame)

heightEntry.insert(0, options["windowHeight"])

self.settingsWidgets.append([heightLabel, heightEntry])

axisThicknessLabel = tk.Label(settingsFrame, text="Axis Thickness")

axisThicknessEntry = tk.Entry(settingsFrame)

axisThicknessEntry.insert(0, options["axisThickness"])

self.settingsWidgets.append([axisThicknessLabel, axisThicknessEntry])

axisColourLabel = tk.Label(settingsFrame, text="Axis Colour")

axisColourEntry = tk.Entry(settingsFrame)

axisColourEntry.insert(0, options["axisColour"])

self.settingsWidgets.append([axisColourLabel, axisColourEntry])

plottedColourLabel = tk.Label(settingsFrame, text="Graph Colour")

plottedColourEntry = tk.Entry(settingsFrame)

plottedColourEntry.insert(0, options["plottedColour"])

self.settingsWidgets.append([plottedColourLabel, plottedColourEntry])

plottedThicknessLabel = tk.Label(settingsFrame, text="Graph Thickness")

plottedThicknessEntry = tk.Entry(settingsFrame)

plottedThicknessEntry.insert(0, options["plottedThickness"])

self.settingsWidgets.append([plottedThicknessLabel, plottedThicknessEntry])

backgroundColourLabel = tk.Label(settingsFrame, text="backgroundColour")

backgroundColourEntry = tk.Entry(settingsFrame)

backgroundColourEntry.insert(0, options["backgroundColour"])

self.settingsWidgets.append([backgroundColourLabel, backgroundColourEntry])

self.guidelinesToggle = tk.IntVar()

self.guidelinesToggle.set(options["guidelines"])

guideLinesLabel = tk.Label(settingsFrame, text="Guidelines Toggle")

guideLinesCheckbox = tk.Checkbutton(settingsFrame, variable = self.guidelinesToggle)

self.settingsWidgets.append([guideLinesLabel, guideLinesCheckbox])

guidelineColourLabel = tk.Label(settingsFrame, text="Guideline Colour")

guidelineColourEntry = tk.Entry(settingsFrame)

guidelineColourEntry.insert(0, options["guidelineColour"])

self.settingsWidgets.append([guidelineColourLabel, guidelineColourEntry])

guidelineFontSizeLabel = tk.Label(settingsFrame, text="Guideline Font Size")

guidelineFontSizeEntry = tk.Entry(settingsFrame)

guidelineFontSizeEntry.insert(0, options["guidelineFontSize"])

self.settingsWidgets.append([guidelineFontSizeLabel, guidelineFontSizeEntry])

guidelineThicknessLabel = tk.Label(settingsFrame, text="Guideline Thickness")

guidelineThicknessEntry = tk.Entry(settingsFrame)

guidelineThicknessEntry.insert(0, options["guidelineThickness"])

self.settingsWidgets.append([guidelineThicknessLabel, guidelineThicknessEntry])

fontColourLabel = tk.Label(settingsFrame, text="Grid Font Colour")

fontColourEntry = tk.Entry(settingsFrame)

fontColourEntry.insert(0, options["fontColour"])

self.settingsWidgets.append([fontColourLabel, fontColourEntry])

noOfPlotsBaseLabel = tk.Label(settingsFrame, text="Number of Plots")

noOfPlotsBaseEntry = tk.Entry(settingsFrame)

noOfPlotsBaseEntry.insert(0, options["noOfPlotsBase"])

self.settingsWidgets.append([noOfPlotsBaseLabel, noOfPlotsBaseEntry])

**for** row **in** range(len(self.settingsWidgets)):

**for** i **in** range(2):

self.settingsWidgets[row][i].grid(row=row, column=i)

settingsFrame.pack(padx = 20)

applyButton = tk.Button(settingsRoot, text="Apply", command=self.applySettings)

applyButton.pack()

defaultButton = tk.Button(settingsRoot, text="Restore Default", command = self.defaultOptions)

defaultButton.pack()

**def** applySettings(self):

*# Initalize settings dictionary*

settings = {}

*# Names of the settings; they will always be in this order*

settingNames = ("windowWidth", "gridWidth", "windowHeight", "axisThickness",

"axisColour", "plottedColour", "plottedThickness", "backgroundColour", "guidelines",

"guidelineColour", "guidelineFontSize", "guidelineThickness", "fontColour", "noOfPlotsBase", "noOfPlots")

**for** settingNum **in** range(14):

*# If the widget is a text entry, it reads the text.*

**if** type(self.settingsWidgets[settingNum][1]) **is** tk.Entry:

settings[settingNames[settingNum]] = self.settingsWidgets[settingNum][1].get()

*# If the widget is a checkbox, it reads the status of the checkbox variable.*

**elif** type(self.settingsWidgets[settingNum][1]) **is** tk.Checkbutton:

settings[settingNames[settingNum]] = self.guidelinesToggle.get()

validatedSettings = self.validateSettings(settings)

**if** validatedSettings != False:

**with** open('options.json', 'w') **as** optionsFile:

json.dump(validatedSettings, optionsFile)

dialogBox = tk.Toplevel()

dialogBox.title("Maths Graphing Tool")

dialog = tk.Label(dialogBox, text="Settings Applied**\n\n**Please Restart the program for the options to take effect.")

dialog.pack()

**def** validateSettings(self, settings):

valid = True

**try**:

settings["windowWidth"] = int(settings["windowWidth"])

settings["gridWidth"] = int(settings["gridWidth"])

settings["windowHeight"] = int(settings["windowHeight"])

settings["axisThickness"] = int(settings["axisThickness"])

settings["plottedThickness"] = int(settings["plottedThickness"])

settings["guidelines"] = int(settings["guidelines"])

settings["guidelineFontSize"] = int(settings["guidelineFontSize"])

settings["guidelineThickness"] = int(settings["guidelineThickness"])

settings["noOfPlotsBase"] = int(settings["noOfPlotsBase"])

settings["axisColour"] = tuple([int(x) **for** x **in** settings["axisColour"].split(" ")])

settings["plottedColour"] = tuple([int(x) **for** x **in** settings["plottedColour"].split(" ")])

settings["backgroundColour"] = tuple([int(x) **for** x **in** settings["backgroundColour"].split(" ")])

settings["guidelineColour"] = tuple([int(x) **for** x **in** settings["guidelineColour"].split(" ")])

settings["fontColour"] = tuple([int(x) **for** x **in** settings["fontColour"].split(" ")])

**except** **ValueError**:

valid = False

**else**:

**if** settings["windowWidth"] < 100:

valid = False

**elif** settings["gridWidth"] > settings["windowWidth"]:

valid = False

**elif** settings["windowHeight"] < 100:

valid = False

**elif** settings["axisThickness"] < 1:

valid = False

**elif** settings["plottedThickness"] < 1:

valid = False

**elif** settings["guidelineFontSize"] < 1:

valid = False

**elif** settings["guidelineThickness"] < 1:

valid = False

**elif** settings["noOfPlotsBase"] < 1:

valid = False

**for** colourSetting **in** ("axisColour", "plottedColour", "backgroundColour", "guidelineColour", "fontColour"):

**for** value **in** settings[colourSetting]:

**if** value > 255 **or** value < 0:

valid = False

**if** valid == False:

warningRoot = tk.Toplevel()

warningRoot.title("Invalid Setting")

warningRoot.resizable(False, False)

warningLabel = tk.Label(warningRoot, text="One or more of your entered settings are invalid.**\n\n**Please ensure that:**\n**- Size settings are numbers greater than 99.**\n**- Colours are three numbers between 0 and 255, seperated by a space (eg. 50 50 50).**\n**- Font sizes are above 0.**\n**- Thickness sizes are above 0.**\n**- Number of plots is above 0.**\n**- All settings are whole numbers, not decimals.")

warningLabel.pack()

**return** False

**elif** valid == True:

**return** settings

**def** defaultOptions(self):

**with** open('optionsDefault.json') **as** optionsFile:

default = json.load(optionsFile)

**with** open('options.json', 'w') **as** optionsFile:

json.dump(default, optionsFile)

dialogBox = tk.Toplevel()

dialogBox.title("Maths Graphing Tool")

dialog = tk.Label(dialogBox, text="Settings Applied**\n\n**Please Restart the program for the options to take effect.")

dialog.pack()

**def** loadGraphs(self, path):

examplesRoot = tk.Toplevel()

examplesRoot.title("Maths Graphing Load Graphs")

examplesRoot.resizable(False, False)

titleLabel = tk.Label(examplesRoot, text = "Load Graphs", font=("Helvetica", 16))

titleLabel.pack()

examplesFrame = tk.Frame(examplesRoot, borderwidth=1, width=500, height=400)

examplesFrame.pack\_propagate(0)

examplesFrame.pack(fill="both", expand=True)

scrollable = tk.Frame(examplesFrame)

*# Initalize the scrollbar and canvas*

examplesScrollbar = tk.Scrollbar(scrollable, orient=tk.VERTICAL)

examplesScrollbar.pack(fill=tk.Y, side=tk.RIGHT, expand=0)

examplesCanvas = tk.Canvas(scrollable, yscrollcommand=examplesScrollbar.set) *# The yscrollcommand attribute links the canvas to the scrollbar*

examplesCanvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=1)

examplesScrollbar.config(command=examplesCanvas.yview)

*# Set the inital scroll values for the canvas*

examplesCanvas.xview\_moveto(0)

examplesCanvas.yview\_moveto(0)

*# The interior frame is created in the canvas, and will contain the widgets which will be moved when the user scrolls*

interior = tk.Frame(examplesCanvas)

interior\_id = examplesCanvas.create\_window(0, 0, window=interior, anchor=tk.NW)

**def** configureInterior(event):

*# Changes the size of the interior frame and the scrollable region of the canvas*

size = (interior.winfo\_reqwidth(), interior.winfo\_reqheight())

examplesCanvas.config(scrollregion="0 0 **%s** **%s**" % size)

**if** interior.winfo\_reqheight() != examplesCanvas.winfo\_width():

examplesCanvas.config(width=interior.winfo\_reqwidth())

*# The interior frame will call the function configureInterior whenever it changes size*

interior.bind('<Configure>', configureInterior)

**def** configureCanvas(event):

*# Changes the interior frame's width to the width of the canvas*

**if** interior.winfo\_reqwidth() != examplesCanvas.winfo\_width():

examplesCanvas.itemconfigure(interior\_id, width=examplesCanvas.winfo\_width())

*# The canvas will call configureCanvas whenever it changes size*

examplesCanvas.bind('<Configure>', configureCanvas)

scrollable.pack(fill=tk.BOTH, expand=1)

*# Get a list of only the files in the current directory.*

fileNames = [name **for** name **in** os.listdir(path) **if** os.path.isfile(os.path.join(path, name))]

**for** fileName **in** fileNames:

*# The full path of the file*

filePath = os.path.join(path , fileName)

**with** open(filePath) **as** graphFile:

*# Reading the contents of the file*

graphData = json.load(graphFile)

*# Create the gui for the file in the interior frame*

graphFrame = tk.Frame(interior, borderwidth=1, relief="solid", width = 400, height = 50)

graphFrame.pack\_propagate(0)

graphFrame.grid\_propagate(0)

graphEquationLabel = tk.Label(graphFrame, text=graphData["equation"], font=("Consolas", 12))

graphDescriptionLabel = tk.Label(graphFrame, text=graphData["description"], font=("Helvetica", 11))

graphButton = tk.Button(graphFrame, text="Select", command=**lambda** equation=graphData["equation"]: self.getEquationInput(equation))

graphEquationLabel.grid(row=0,column=0, padx=5)

graphDescriptionLabel.grid(row=0,column=1, padx=5)

graphButton.grid(row=0,column=2, padx=5, sticky=tk.E)

graphFrame.pack()

**class** **equationFrame**():

*# Class that stores the equation object and the equation frame so that they are linked*

**def** delete(self):

*# Deletes the equation object and removes it's corrosponding frame*

self.frame.pack\_forget()

self.frame.destroy()

equations.equationList.remove(self.linkedEquation)

**if** len(equations.equationList) > 0:

equations.calculateNoOfPlots( len(equations.equationList ) )

**def** toggle(self):

*# Toggle the visibility of the equation*

self.linkedEquation.visible = **not** self.linkedEquation.visible

**def** newEquation(self):

self.getEquationInput(self.graphEntry.get("1.0", tk.END))

**def** getEquationInput(self, equation):

*#Get the user's equation input and create the frame for the 'currently graphed'*

**try**:

eqFrame = gui.equationFrame()

newEquation = equations.createEquation(equation, eqFrame)

newEquation.solve(0)

**except** (**NameError**, sympy.SympifyError, **SyntaxError**):

warningRoot = tk.Toplevel()

warningLabel = tk.Label(warningRoot, text = "Invalid Graph**\n\n**Please refer to the tutorial", font=("Helvetica", 15))

warningLabel.pack()

equations.equationList.remove(newEquation)

**del** eqFrame

**else**:

eqFrame.frame = tk.Frame(self.interior, height=200, borderwidth=1, relief="solid")

eqFrame.text = tk.Text(eqFrame.frame, width = 12, height = 1, font=("Helvetica", 15))

eqFrame.text.insert('end', equation)

eqFrame.text.configure(state='disabled')

eqFrame.text.pack(side=tk.LEFT)

eqFrame.equationDeleteButton = tk.Button(eqFrame.frame, text="Delete", command=eqFrame.delete)

eqFrame.equationCheckbox = tk.Button(eqFrame.frame, text="Toggle", command=eqFrame.toggle)

eqFrame.equationDeleteButton.pack(side=tk.RIGHT)

eqFrame.equationCheckbox.pack(side=tk.RIGHT, ipadx=3)

eqFrame.frame.pack(fill=tk.X, expand=False, ipadx=1, ipady=5, pady=1)

**def** saveGraphs(self):

bannedCharacters = ["<", ">", ":", '"', "/", "**\"**", "|", "?" ,"\*"]

**for** graph **in** equations.equationList:

graphData = {}

graphData["equation"] = graph.equationString

nameWindow = tk.Toplevel()

nameLabel = tk.Label(nameWindow, text="What do you wish to save {} as?".format(graphData["equation"]), font=("Helvetica", 16))

nameLabel.pack()

nameEntry = tk.Entry(nameWindow, font=("Helvetica", 11))

nameEntry.pack()

wait = tk.IntVar()

name = None

*# Functions for the buttons*

**def** assignName():

nonlocal name, wait

name = nameEntry.get()

*#Changes the wait variable, so the program continues*

**if** any(character **in** bannedCharacters **for** character **in** name):

warningRoot = tk.Toplevel()

warningLabel = tk.Label(warningRoot, text = 'Invalid Input**\n\n**A filename cannot contain the following characters: < > : " / \ | ? \*', font = ("Helvetica", 16))

warningLabel.pack()

**else**:

wait.set(1)

**def** assignDescription():

nonlocal graphData, wait

graphData["description"] = nameEntry.get()

wait.set(2)

nameEntryButton = tk.Button(nameWindow, text="Save", command=assignName)

nameEntryButton.pack()

*# This will make the program wait until the variable 'wait' changes, so in this case, when the user presses 'Save'*

nameWindow.wait\_variable(wait)

nameLabel.configure(text="Please write a short description of the graph {}".format(graphData["equation"], font=("Helvetica", 16)))

nameEntry.delete(0, "end")

nameEntryButton.configure(command=assignDescription)

nameWindow.wait\_variable(wait)

*# Writing the dictionary to the file name specified, using a json format.*

**with** open("./Graphs/{}.json".format(name), 'w') **as** graphFile:

json.dump(graphData, graphFile)

nameWindow.destroy()

**class** **gridController**():

**def** \_\_init\_\_(self):

self.cameraX, self.cameraY = (0, 0) *#Initalize the camera starting coordinates*

self.cameraWidth, self.cameraHeight = (20, 20) *#Initalize the width and height the camera can see.*

self.graphSurface = pygame.display.set\_mode( (options["windowWidth"],options["windowHeight"]) )

self.graphSurface.fill( (255, 255, 255 ) )

self.pixelDX = (self.cameraWidth/ options["gridWidth"])

self.pixelDY = (self.cameraHeight/ options["windowHeight"])

self.numXMarkers = 10

self.numYMarkers = 10

self.gridFont = pygame.font.SysFont("monospace", options["guidelineFontSize"])

**def** getGridCoordinate(self, coordsTuple):

*# Take an (x, y) tuple on the screen and return it's x, y value on the cartesian grid*

x, y = coordsTuple *#Unpack tuple*

gridX = self.pixelDX\*(x - options["windowWidth"]/2) + self.cameraX

gridY = self.pixelDY\*(y - options["windowHeight"]/2) - self.cameraY

**return** ( gridX, gridY )

**def** getScreenCoordinate(self, coordsTuple):

*# Take an (x, y) tuple on the grid and return it's x, y value on the screen*

x, y = coordsTuple

screenX = ((x-self.cameraX)/self.pixelDX) + options["gridWidth"]/2

screenY = ((-y-self.cameraY)/self.pixelDY) + options["windowHeight"]/2

**return** ( screenX, screenY )

**def** drawGrid(self):

*# Draws the entire grid*

*# graphSurface is cleared*

self.graphSurface.fill(options["backgroundColour"])

*# Finds where the screen coordinate of where (0, 0) is*

screenXZero, screenYZero = self.getScreenCoordinate( (0, 0) )

*# Calls the calculate markers function to calulate the coordinates of the x and y markers*

xMarkers = self.calculateMarkers( (self.cameraX - 0.5\*self.cameraWidth), (self.cameraX + 0.5\*self.cameraWidth), self.numXMarkers)

yMarkers = self.calculateMarkers( (self.cameraY - 0.5\*self.cameraHeight), (self.cameraY + 0.5\*self.cameraHeight), self.numYMarkers)

**for** x **in** xMarkers:

**if** x != 0:

*# Find the screen coordinate of the marker*

screenX, screenY = self.getScreenCoordinate( (x, 0) )

*# Draws the guidelines for the marker if toggled on*

**if** options["guidelines"] == 1:

pygame.draw.line(self.graphSurface, options["guidelineColour"], (screenX,0), (screenX, options["windowHeight"]), 1)

markerLabel = self.gridFont.render(str(x), 1, options["fontColour"])

textWidth, textHeight = self.gridFont.size(str(x))

self.graphSurface.blit(markerLabel, (screenX-textWidth//2, screenY) )

**for** y **in** yMarkers:

**if** y != 0:

screenX, screenY = self.getScreenCoordinate( (0, -y) )

**if** options["guidelines"] == 1:

pygame.draw.line(self.graphSurface, options["guidelineColour"], (0,screenY), (options["gridWidth"], screenY), 1)

markerLabel = self.gridFont.render(str(y), 1, options["fontColour"])

textWidth, textHeight = self.gridFont.size(str(y))

self.graphSurface.blit(markerLabel, (screenX, screenY-textHeight//2) )

*# Draw the axis lines onto the grid*

pygame.draw.line(self.graphSurface, options["axisColour"], (screenXZero,0), (screenXZero,options["windowHeight"]), options["axisThickness"] )

pygame.draw.line(self.graphSurface, options["axisColour"], (0,screenYZero), (options["gridWidth"],screenYZero), options["axisThickness"] )

self.drawEquations()

*# Updates the display*

pygame.display.flip()

gui.root.update()

**def** calculateMarkers(self, min, max, numOfMarkers):

*# Calculates where the markers should be placed*

Range = self.round( max - min )

markerInterval = self.round( (Range / (numOfMarkers)))

roundedMin = math.floor(min/markerInterval) \* markerInterval

roundedMax = math.ceil(max/markerInterval) \* markerInterval

markerCoords = []

**for** i **in** numpy.arange(roundedMin, roundedMax+markerInterval, markerInterval):

*# Round the result to the same order of magnitude as the marker interval to prevent floating point errors*

markerCoords.append( round(i, -int(math.floor(math.log10(markerInterval)))))

**return** markerCoords

**def** round(self, number):

*# Round the number given to the nearest 1, 2 or 5 and their power of ten multiples*

*# Finds the order of magnitude of the number*

exponent = math.floor(math.log10(number))

fraction = number/(10\*\*exponent)

**if** fraction < 1.5:

fraction = 1

**elif** fraction < 3:

fraction = 2

**elif** fraction < 7:

fraction = 5

**else**:

fraction = 10

**return** (fraction \* (10\*\*exponent))

**def** drawEquations(self):

*# Draw every equation*

**for** equation **in** equations.equationList:

**if** equation.visible:

*# Generate a list of evenly spaced numbers from the left side of the screen to the right side.*

xValues = numpy.linspace( (grid.cameraX - grid.cameraWidth//2)-1 , (grid.cameraX + grid.cameraWidth//2)+1, options["noOfPlots"]).tolist()

yValues = []

**for** xValue **in** xValues:

solution = equation.solve(xValue)

**if** solution == [sympy.zoo] **or** solution == [] **or** solution == None:

*# sympy.zoo is the result of dividing something by zero, when this happens*

*# the solution at that x-value is undefined and cannot be plotted.*

**pass**

**else**:

yValues.append(solution[0])

*# This is outside the loop to avoid unnecessary calculations.*

x, y = grid.getScreenCoordinate( (xValues[0], yValues[0]) )

**for** i **in** range(len(xValues)-1):

**if** i+1 >= len(yValues):

**break**

nextX, nextY = grid.getScreenCoordinate( (xValues[i+1], yValues[i+1]) )

**if** nextY > 10000:

nextY = 10000

**elif** nextY < -10000:

nextY = -10000

pygame.draw.line(grid.graphSurface, options["plottedColour"], (x, y), (nextX, nextY), options["plottedThickness"])

x, y = nextX, nextY

**class** **inputHandler**():

**def** \_\_init\_\_(self):

self.keys = {

"m1": 0, *#left*

"m2": 0, *#right*

"m3": 0 *#middle*

}

**def** handleInput(self):

*# Checks for user input inside of the grid and responds accordingly*

*# Get the list of user inputs*

eventList = pygame.event.get()

*# Get the current state of the mouse*

(self.keys["m1"], self.keys["m2"], self.keys["m3"]) = pygame.mouse.get\_pressed()

**for** event **in** eventList:

**if** event.type == pygame.QUIT:

pygame.quit()

gui.root.destroy()

sys.exit(0)

**if** event.type == pygame.MOUSEBUTTONDOWN:

*# If the user has the left mouse button pressed, it will call get\_rel()*

*# which calculates the difference in the position of the mouse since the*

*# last time it has been called.*

**if** event.button == 1:

pygame.mouse.get\_rel()

*# If the user scrolls up or down, the zoom method is called with the corrosponding input*

**if** event.button == 5:

self.zoom("out")

**if** event.button == 4:

self.zoom("in")

**if** self.keys["m1"] == 1:

self.moveGraph()

**def** moveGraph(self):

(x, y) = pygame.mouse.get\_rel()

*# Converts the screen coordinates of the mouse movement to graph coordinates.*

xMovement = x\*grid.pixelDX

yMovement = y\*grid.pixelDY

grid.cameraX -= xMovement

grid.cameraY -= yMovement

**def** zoom(self, direction):

*# Adjusts the width and height of the camera and adjusts the pixelDX and pixelDY accordingly.*

**if** direction == "in":

grid.cameraWidth -= 0.1\* grid.cameraWidth

grid.cameraHeight -= 0.1\* grid.cameraHeight

**else**:

grid.cameraWidth += 0.1\* grid.cameraWidth

grid.cameraHeight += 0.1\* grid.cameraHeight

grid.pixelDX = grid.cameraWidth / options["gridWidth"]

grid.pixelDY = grid.cameraHeight / options["windowHeight"]

**class** **equationController**():

**def** \_\_init\_\_(self):

self.equationList = []

**def** createEquation(self, function, frame):

*# Create a new equation object and links the frame parameter to the equation*

newEquation = equation(function)

frame.linkedEquation = newEquation

newEquation.frame = frame

self.equationList.append(newEquation)

self.calculateNoOfPlots( len(self.equationList ) )

**return** newEquation

**def** calculateNoOfPlots(self, equations):

*# Update the amount of calculations to make for each equation, when there are n equations*

options["noOfPlots"] = (options["noOfPlotsBase"] \* (math.log(equations) + 0.5772156649 + 1/(2\*equations) - 1/(12\*equations\*\*2) )) / equations

options["noOfPlots"] = round(options["noOfPlots"])

**class** **equation**():

**def** \_\_init\_\_(self, function):

*# Split the equation into the left-hand side and the right-hand side*

self.visible = True

*# Initalize the sympy symbols*

self.x = sympy.symbols("x")

self.y = sympy.symbols("y")

self.equationString = str(function).rstrip()

sides = function.replace(" ", "").lower().split("=")

self.leftSide = sides[0]

self.rightSide = sides[1]

*# Create a sympy equation and solve it for y.*

self.fx = sympy.Eq( sympy.sympify(self.leftSide), sympy.sympify(self.rightSide) )

self.fx = sympy.solve(self.fx, self.y)

**for** i **in** range(len(self.fx)):

self.fx[i] = str(self.fx[i]).replace("-x\*\*2", "-(x\*\*2)")

**def** solve(self, xValue):

*#Solve the equation for some x value.*

solutions = []

**for** equation **in** self.fx:

x = xValue

**try**:

solutions.append( eval( str(equation) ) )

**except** **ValueError**:

**pass**

**if** solutions != []:

**return** solutions

**def** readOptions():

**global** options

**with** open('options.json') **as** optionsFile:

data = json.load(optionsFile)

options = data

**def** initApplication():

**global** grid, gui, input, equations

readOptions()

gui = guiController()

os.environ['SDL\_WINDOWID'] = str(gui.gridFrame.winfo\_id())

**if** platform.system() == "Windows":

os.environ['SDL\_VIDEODRIVER'] = 'windib'

pygame.init()

grid = gridController()

input = inputHandler()

equations = equationController()

mainLoop()

**def** mainLoop():

**while** True:

grid.drawGrid()

gui.root.update\_idletasks()

gui.root.update()

input.handleInput()

pygame.quit()

sys.exit(0)

initApplication()