# Design Section

## 1. Structure

The starting point of the program is the program object, which stores references to the different parts of the program. The program object has three such members: window, diagram, and preferences. diagram and window both have references to the program object, allowing them to communicate.

The diagram object stores the currently loaded diagram including all of the current plots. It also handles serialising diagrams to .arg files.

The window object is a QMainWindow subclass which stores the entire GUI. It is responsible for handling inputs and drawing plots, as well as creating dialogs boxes.

preferences is a simple object which stores a few diagram nonspecific preferences.

## 2. Data Dictionary

### File Formats

#### config.ini

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Purpose** |
| label\_points | Boolean | Whether labels should be drawn for points. |
| label\_axes | Boolean | Whether labels should be drawn for axes. |
| font\_size | Integer | Font size to use when drawing labels. |
| stroke | Integer | Thickness in pixels of plotted lines. |
| window\_width | Integer | Stores the last known width of the window. |
| window\_height | Integer | Stores the last known height of the window. |

#### .arg File (Saved Diagram)

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Purpose** |
| plots... | Array | Array of serialized plot objects that can be recreated using pickle. |
| translate | Coordinate | Stores the last point the display was centred around. |
| zoom | Number | Stores the last zoom value used for the diagram. |

### Object Classes

#### Program

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Purpose** |
| window | Window object | Permanent reference to the main window object. |
| diagram | Diagram object | Reference to the currently loaded diagram object. It is mainly accessed by the GUI code when it needs to add/remove/display plots on screen. |
| preferences | Preferences object | Permanent reference to the preferences object. This is also accessed by the GUI code when displaying plots. |

#### Window

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Purpose** |
| program | Program object | Permanent reference to the program object. |
| widgets... | Several PyQt object types | Several separate references to PyQt widget and layout objects that make up the on screen GUI. |
| actions... | QAction objects | References to actions for the window. These are functions which can be placed in the menu bar and assigned keyboard shortcuts and icons. |

#### Diagram

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Purpose** |
| program | Program object | Permanent reference to the program object. |
| path | String | Path to the associated .arg file (including the filename). |
| filename | String | Name of the associated .arg file (excluding the .arg extension). |
| plots | QModel object | QModel containing the plot objects to be drawn to the screen. |
| translate | Coordinate | Stores the point the display should be centred around when the diagram is drawn. |
| zoom | Number | Stores the zoom level used when drawing the diagram. |

#### Preferences

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Purpose** |
| program | Program object | Permanent reference to the program object. |
| stroke | Integer | The width with which individual strokes (lines) should be drawn on the diagram. |
| label\_axes | Boolean | Whether the axes should be labelled with numbers. |
| label\_points | Boolean | Whether points should be labelled with their values. |

#### Plot

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Purpose** |
| equation | String | The original text-input equation, used for display. |
| type | Integer (enumeration) | An integer which represents the type of plot stored:   * TYPE\_POINT * TYPE\_CIRCLE * TYPE\_DISK * TYPE\_NEGATIVE\_DISK * TYPE\_LINE * TYPE\_HALF\_PLANE * TYPE\_RAY * TYPE\_DUAL\_RAY * TYPE\_SECTOR * TYPE\_NULL (a special type for inputs which are valid, but contain no points, e.g.: ) |
| relation | Integer (enumeration) | An integer which represents the relational operator used in the equation: <=, <, =, >, or >=. |
| shape | Shape subclass object | A shape object which contains the type-specific information needed to draw the object, e.g. it’s size, position, orientation etc. |
| color | QColor object | The color that should be used when drawing the plot. |

## 3. Storage

The program will not contain any large images, audio, or video, but will need to be distributed with Python and PyQt binaries. Nonetheless, it should only be a few megabytes in size, and may be installed completely in the Program Files directory.

.arg files will use a binary format that should be small enough to be stored in large numbers in the users’ home directories, be uploaded to the internet, or kept on external mass storage devices.

## 4. GUI DesignE:\Documents\Design\file_menu.jpgE:\Documents\Design\help_menu.jpgE:\Documents\Design\view_menu.jpg

## 5. Processes

### Procedures

**Parse an entered locus:**

* **Validate the locus:**
  + Does the locus take the form , where:
    - is one of , , , , , , or a constant .
      * All are a single letter other than .
      * Each is either a real number, a real number succeeded by the letter (an imaginary number), or a sum of the two.
    - is one of , , , , or .
  + If is or , is a constant, and is the locus an equation?
  + If is or , is a real constant or another modulus function?
  + If is or , is a real constant?
  + These cases cover all the diagrams the software is required to plot. The following cases may also be simple to implement, but are not required:
  + If is or , is or (where the are equal)?
  + If is or , is also an argument function?
* **Store the locus:**
  + If the locus represents a point, *i.e.* takes the form , store the point as .
  + If the locus represents the perpendicular bisector of points and , find and store the Cartesian equation of the line.
    - The midpoint of and is .
    - The gradient of the line is given by .
    - Hence, the imaginary intercept of the line is .
    - So the equation is written .
  + If the locus represents a circle, *i.e.* takes the form , store the centre and radius as and respectively.
  + If the locus represents a ray, *i.e.* takes the form , store the start point as , and the gradient as .

**Draw a stored locus:**

* **If the locus is a point:**
  + Draw the point using the graphics library.
  + If enabled in the preferences, label the point with its coordinates.
* **If the locus is a circle:**
  + Use the graphics library to draw a circle with the stored radius and centre.
* **If the locus is a disk:**
  + Draw a circle with a dashed stroke if applicable, and use a solid colour fill.
* **If the locus is a perpendicular bisector:**
  + Draw the line:
    - Find the current lower and upper visible values on screen.
    - Plug these into the line equation to find the start and end-points of the line.
* **If the locus is a half-plane:**
  + Draw the line, with a dashed stroke if applicable.
  + Draw the inequality region with a solid colour fill.

**Set window size:**

* **On window close:**
  + If config.ini does not exist, create it.
  + Update variables window\_width and window\_height.
* **On window open:**
  + If config.ini does exist:
    - Set the window size to the values of window\_width and window\_height.
  + Otherwise create it, and use the default window size.

### Regular Expressions

This regular expression matches a single numerical term. This could be used to find values of .

/([+-]\s\*)?(\d\*\.)?\d+i?(?=\s\*[\)|<=>+-])/g

This regular expression matches a single occurrence of an unknown, with or without a coefficient. This can be used to find instances of during validation, and to check the letter used for the unknown is consistent.

/([+-]\s\*)?((\d\*\.)?\d+)?[a-hj-z](?=\s\*($|[\)|<=>+-]))/g

This regular expression is needed to match a group of numerical or unknown terms separated by whitespace, without matching whitespace on either end.

/([+-]\s\*)?((\d\*\.)?\d+i?|((\d\*\.)?\d+)?[a-hj-z])(\s\*([+-]\s\*)?((\d\*\.)?\d+i?|((\d\*\.)?\d+)?[a-hj-z]))\*(?=\s\*($|[\)|<=>+-]))/g

However, if all whitespace is stripped from the input, the expression can be reduced to this.

/(([+-])?((\d\*\.)?\d+i?|((\d\*\.)?\d+)?[a-hj-z]))+(?=$|[\)|<=>+-])/g

*From this point, all regular expressions require the input contain no whitespace.*

This regular expression matches a modulus function and its contents.

/\|[^|]\*\|(?=$|[<=>+-])(?<=[<=>+-])/g

This similar regular expression matches an argument function and its contents.

/arg\([^\(]\*\)(?=$|[<=>+-])(?<=[<=>+-])/g

Finally, this regular expression matches all valid equality symbols: “<=”, “<”, “=”, “>” and “>=”.

/(=|[<>]=?)(?=[^<=>]+)(?<=[^<=>]+)/g

## 6. Security & Integrity of Data

The system will not store any sensitive data, nor will it communicate over any network. There is very little security risk involved with the system, so it does not require any special measures.

Stored data (.arg files) will be kept in a binary format, so it is likely that any corruption to a file will cause the loading process to fail, rather than read false values. Input data will be parsed by a mathematical parser which should prevent incorrectly entered equations from being loaded into the state (which could cause errors when drawn).

## 7. Test Strategy

As the system is very complex, it would be impossible to use white-box testing on every element of the code. For this reason, I intend to use a black-box testing method. Testing will mainly involve creating equations for which I know the correct output, inputting them into the program and checking the output given.