Cab320 Assignment- Group 420

Jordy Garland, Joshua Inglis, Jonathan Salazar, Mikhayla Stephenson

Features

Basic functionality

Shapes

Rectangle

The Rectangle feature has been implemented with full functionality. Clicking on the canvas while the rectangle tool is selected will trigger a visually responsive rectangle to be drawn. The rectangle will 'stick' to the user's cursor until they click for a second time, disengaging with the shape and finalizing the rectangle on the canvas. The rectangle has customisable border and fill colours.

Ellipse

The Ellipse's functionality is very similar to the rectangle. The first click on the canvas will cause an ellipse shape to render and follow the cursor, the second will release the ellipse and leave it in place. Fill and border colours are customisable as specified.

Plot

The Plot shape's functionality is simplistic. When selected, the user can click anywhere on the canvas, leaving a small dot at the location of the cursor. The plot shape's colour can be changed by choosing a new pen colour.

Line

The implemented Line tool is again very similar to the rectangle and ellipse options. The user clicks once to set the starting point of the shape. As they move the cursor, the shape will follow, clicking a second time will release the shape. This shape's colour is customisable.

Polygon

The Polygon shape noticeably differs from the other shapes. The polygon initializes when the user clicks on the canvas. Once initialized, a line will appear following the cursor, any following clicks will finalize the users' current line and begin a new one. To release and close the polygon, the user can press 'enter'. This will connect the polygon's last entered vector point with the first entered vector point. This shape has customisable border and fill colours.

Tools

Pen

The pen selector tool implemented can be seen in the bottom left corner of the running application. This tool selects the border colour of shapes about to be drawn. Functionality is simple. Users click on the pen tool then click on the desired colour they wish to draw in. The pen will display the chosen colour. Shapes being drawn will have a border colour matching the current pen colour selected.

Fill

The fill tool is located close to the pen tool and functions very similarly to it. Users can click on this tool, then their desired colour. While the fill button is engaged, shapes being drawn will be filled with the corresponding fill colour. It is possible to unengaged the fill button by pressing the 'no fill' button below it. This will set shapes' fill to transparent.

Colour Picker

The colour picker is implemented in two ways. The first is for a palette of common colour choices to always be displayed in the bottom left of the application. To accommodate more choices, a second, more in-depth colour picker is available to the user. This picker has multiple modes and allows for an infinite number of colour possibilities for the user.

Open

The open functionality allows a user to open a vec file. This then displays the image that the vec file contained. This is stored under file and the user can select what file to open then in a file picker window.

Save As

The save as functionality allows a user to save a vec file. The user goes to file and then save as, they will be able to select a location to save their vec file. It is set to be vec as default and the user can change the name of it. This file should be able to be opened using the open function and then edit in the application.

Additional Functionality

Grid

A grid has been developed as a part of the additional functionality for this app. When the grid is present on the canvas, any shape the user draws will snap to a nearby grid intersection. This allows for a more structured and measured drawing experience. The grid size is customizable. Pressing ctrl 'UP' or 'DOWN' will make the grid panels appear larger or smaller.

BitMap

This app supports the ability to export the current canvas as a bitmap image. By navigating to the file drop down and clicking export, the user will be navigated through creating a bitmap export which will be saved to a location and size of their choosing.

Zoom

A function to allow users to zoom in and out of the canvas view has been created. Zooming automatically scales any shapes drawn onto the canvas. To prevent errors, there is a minimum canvas size, meaning that the extent a user can zoom out to is mildly limited.

Contribution

Our group consists of 4 members, Josh, Jordan, Jonathan and Mikhayla. On the formation of the group, Josh created a project on GitHub and invited each team member to view it. This project had many classes made by Josh, so each member had the same starting point. This was a GUI class, a Main class to test our code, and an initial VectorCanvas class where we could draw shapes. There were also some initial utility classes such as VectorColour and VectorPoint where the application was able to get what colour each shape would be (used in pen and fill). And classes such as Tool, Point and CanvasMouse as starting points for interaction with the canvas. We met up and discussed what sections will be delegated to each member.

Jonathan took up the design of the GUI. This involved making the GUI display all the buttons specified in the design brief and ensuring they functioned correctly. He drew plans and designed the look of the GUI and wrote most the classes associated with the GUI implementation. He also worked on the Pen and Fill colour and Undo functionality. As well as the additional functionality of Zoom. He wrote the *How to Use Application* section of the report and help proofread.

Mikhayla had completed the working tools (polygon, plot, etc) that display on the canvas. It was implemented that the user can select the tool from the toolbar and draw each shape on the canvas. She was committing these to GitHub as they were completed or improved upon. Along with this she also made some tests for the project. Mikhayla also implemented the additional functionality of the grid being displayed on the canvas and having the user snap to it. She has also been working with Jonathan regarding some aspects of the GUI.

Jordan was working with the File IO class to allow users to open, save and save as their drawings as vector files. Along with this allowing users to create a new canvas to draw on. He has been completing the report alongside the rest of the group who each have an area that they know well and have been adding where they feel necessary. Jordan was having issues late in the project between IntelliJ and GitHub therefore some of his commits have been done through sending code to Mikhayla and Josh to commit.

Josh took leadership in changing/creating classes and refactoring/fixing code where necessary. He also worked on any outstanding parts of the project that were not currently being done by the rest of the team. He worked on the classes that allowed the rest of the team to complete their tasks such as the VectorCanvas and most of the utility classes.

There were weekly meetings that every member was there for where progress was discussed, and allocation of jobs were sorted out. (Add what else you have done)

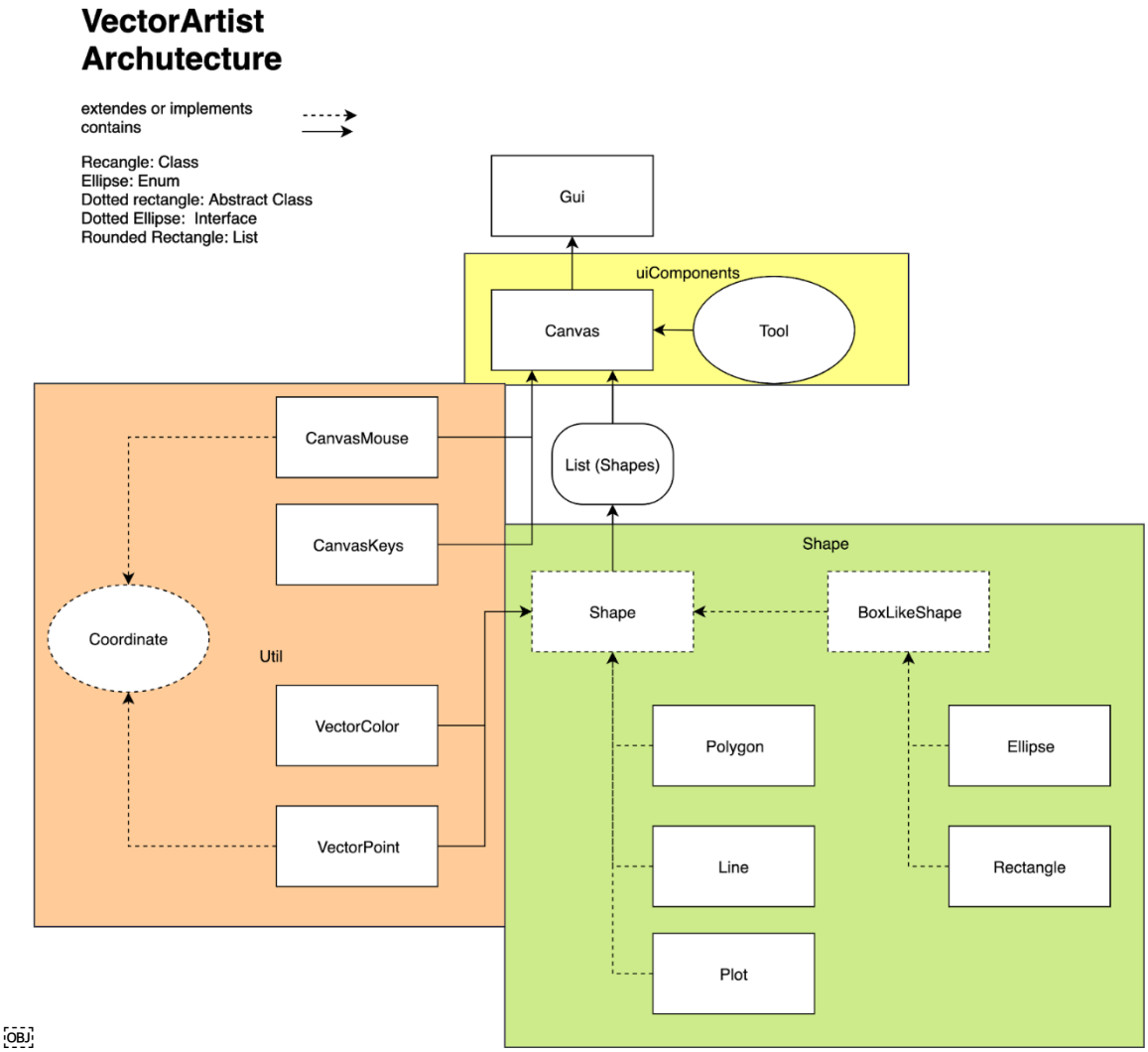
Agile Development

In order to comply with agile development, we required developers implement features on an iterative process. This was achieved by ensuring all commits done by developers to be only to add one specific feature, bug fix, etc., and then tested before developer commits to source. Each week the team met up to review the code and discuss features to be added next. In between weekly meetings, an open line of communication was maintained. This allowed developers to share information and advice as well as collectively approve changes to the dev branch.

Architecture

Overview

The architecture of VectorArtist has a GUI object displaying a VectorCanvas object and awt components to create the interface. The VectorCanvas object stores the selected tool, active colors and shapes created by the user. Attached to VectorCanvas are Mouse and Keyboard listeners (CanvasMouse and CanvasKeys) that listen to user inputs as the main control for the user.



GUI

Handles all graphical interface using Swing and awt api, acts as button listener and comprises on a menu bar, command palette and canvas. Elements of the UI were created with JPanel, VectorCanvas and JButtons. File menu buttons call methods in FileIO. Command pallet buttons alter various aspects of the currently active canvas, like active tool and active colors.

The command pallet is dynamically generated from Enum in uiComponents package. Should the client want to add new button, the developer simply adds a constant with the desired size and image specification.

VectorCanvas

It's the central component of VectorArtists, used to store and display shapes. VectorCanvas extends JFrame, therefore is a Swing component and can be added as such to other Swing Components. VectorCanvas is added to the frame main JFrame created in the GUI class. VectorCanvas overwrites the paintComponent method to paint its contents (VectorShapes) to itself. VectorCanvas has a Tool field and two VectorColor fields to indicate the shape type and colors of the next drawn shape and changed by JButtons on the GUI. The two VectorColors fields indicate the fill and pen colors.

Notable Attributes:

- Shapes (private List): stores all shapes
- selectedTool (private Tool): stores the tool, ie ellipse, rectangle ect. The next shape created will be this shape.
- selectedPenColor (private VectorColor): pen color of the next shape to be created.
- selectedFillColor (private VectorColor): fill color of the next shape to be created.
- getWidth() (public, int): returns the width of the canvas on the screen (in pixels). Since the canvas is always a square this is the height of the canvas as well. Inherited by JPanel
- isShapeCreating() (public, boolean): returns true if a shape is being created, else false.

Interactions

- There are 3 ways for canvas to be modified, with the mouse, and keyboard and buttons. The mouse and keyboard are listened by CanvasMouse and CanvasKeys object and buttons are controlled in the GUI class. Example interactions:
- User clicks on screen -> canvasMouse.mousePressed is invoked -> a new Shape is created and added to shapes
- User presses ctrl-z -> canvasKeys.keyPressed is invoked -> undo is invoked -> the last shape from the shapes is removed
- User presses red color button -> GUI.addColourFunctionality is invoked -> selectedPenColor/selectedFillColor is changed
- The shape created when the canvas is clicked will depend on the value of selectedTool (more on Tool below)

VectorShape:

All shapes in VectorArtist are subclasses of VectorShape and created and handled by a VectorCanvas object, the purpose of which are to represent shapes on the canvas. A VectorShapes have a number of VectorPoint objects to represent key points of the shape on the canvas, this number dependent on the subclass, ie type of shape. For example, the dimensions of a Rectangle object are represented by two points, the top left and bottom right, therefore has two maximum allowable points. Once shapes have been created, they cannot be updated by the user, only deleted. VectorShapes store two private VectorColor fields, to represent pen and fill colors.

Every VectorShape contains an abstract method draw that is called by the canvas whenever the canvas is to be painted to the screen. The draw method draw itself to the canvas Graphic.

If the client requires additional shapes, code has been optimised to accommodate addition of new shape types. The developer creates a subclass of VectorShape and adds an entry to Tool enum.

Notable Attributes:

- Shapes (private List): stores all shapes
- selectedTool (private Tool): stores the tool, ie ellipse, rectangle ect. The next shape created will be this shape.
- selectedPenColor, selectedFillColor (private VectorColor): pen color of the next shape to be created.
- getWidth() (public, int): returns the width of the canvas on the screen (in pixels). Since the canvas is always a square this is the height of the canvas as well. Inherited by JPanel
- isShapeCreating() (public, boolean): returns true if a shape is being created, else false.

Interactions

- User clicks on screen -> canvasMouse.mousePressed is invoked -> a new Shape is created -> Shape size is changed by updating a particular point, depending on the shape type, against the current position of the mouse.
- User opens file -> FileIO interprets string with parseString -> add shape for each line a shape is called in the vec file.
- User saves file -> FileIO requests a vec representation of the shape.
- VectorCanvas.paintComponent is called-> VectorCanvas calls VectorShape.draw, -> VectorShape updates VectorCanvas's Graphics

Extended by: BoxLikeShape, Line, Polygon and Plot

VectroPoint

VectorPoint implement the Point interface and represent a Coordinate in its purest form. (a Coordinate being a dimensionless representation of relative positions between points represented by an x and y between 0 and 1 inclusive). Given the size of canvas, a VectorPoint object can give a swing Point object. Point Object. The purpose of VectorPoint is to provide a scalable represent key points of a shape. So, the shape can be displayed on any size screen and exported to any size image.

Notable Attributes:

- x, y (private, int): Number between 0 and 1
- Update (public, Coordinate):

Interactions:

- User clicks on screen -> Recis created by VectorCanvas -> A key point is

VectorColor:

VectorColor represents the color of a shape in a hex form and serves as an extended form of awt.Color as it contains an addition boolean field, indicating no color. The color data is stored as a hexadecimal int and active state stored as a boolean. The active color of a canvas is changed by JButtons on the GUI class.

FileIO

FileIO is a static class to control the flow of data to and from vec files. When the user saves a file, FileIO takes in the working canvas, and a File object and converts the data into the vec format and writes to the file. When the user loads the file, FileIO takes in a file object, reads the file contents and pareses the buffer into a temp VectorCavas and copy the data into the currently active VectorCanvas.

CanvasKeys
Keyboard listener for VectorCanvas, its sole purpose is to listen for ctrl-z. When ctrl-z is pressed, VectorCanvas.undo of the attached VectorCanvas is invoked.

CanvasMouse
Mouse listener for VectorCanvas, if canvas is clicked on and no shape is currently being created, a new shape will be created. Includes boolean field gridToggle to indicate if the grid should be on or off.

FrameResize
Frame resize listener for the main JFrame of the application. When the user resizes the window, a new thread is created, and the canvas is updated to accommodate the new size of the widow.

ShapeException
Thrown when an undesired event occurs relating, for example switch case in Tool enmun does not correlate to any action (developer error).

VecFileException
Thrown when any unknown commands are found, or command is invalid. Invalid commands may be shapes with invalid number of points, an illegal point (greater than 1 or less than 0) or invalid hex code.

UndoException
Thrown when user tries to undo where doing so would be inappropriate, eg, when there is nothing to undo, or when creating a shape.

Tool
Enum belonging to VectorCanvas to indicate the next shape to create. GUI dynamically creates command buttons for each enum value.

SidebarPanels
Configuration details about the size and placement of the sidebar.

Utilities
Sets the dimensions and images of utility controls JButtons(undo, zoom in/out) on the command pallet

ColourTools
Sets the dimensions and images of utility controls JButtons(undo, zoom in/out) on the command pallet

ColourQuickSelect
Sets the dimensions and images of utility controls JButtons(undo, zoom in/out) on the command pallet

BoxLikeShape
Abstract class that extends VectorShape. Represents all shapes that have two points, a top left and bottom right, have both pen and fill functions. Examples are rectangles and ellipses.

(Shapes)
- Ellipse: Extends BoxLikeShape

- Rectangle: Extends BoxLikeShape
- Line: Similar to BoxLikeShape, however does not have fill.
- Plot: Singular Point, only has pen.
- Polygon: When creating, points are added when clicked on the screen. Shape is finished when user presses enter.

Advanced Software Principles

Abstraction has been used throughout the project in how the classes have been set up. Related objects have been defined under an encompassing class as each have some form of relation to it. This abstraction is used to hide the complexity of the program. Java classes such as fileIO and tools are stored within the utilities package. Inside such classes there are methods that are formed from abstract methods from another class or package. This is an example of abstraction being used in the creation of this software.

Encapsulation is demonstrated in the software through the use of getters and setters. An example of this is the use of getting and setting the pen colour and fill colour. Encapsulation is about whether it works or not, rather than how. There are methods to sort out how to convert a chosen colour into a usable object, but when setting or getting the colours, that is hidden away due to the use of Encapsulation. So when using the colour picker, it may only call a getter or setter, however behind that are many other methods to make sure they work. It also keeps these values private. This reduces the chance they will be changed accidentally. A getter or setter is needed to access these values so they are safely kept and would need to be accessed on purpose for something to change.

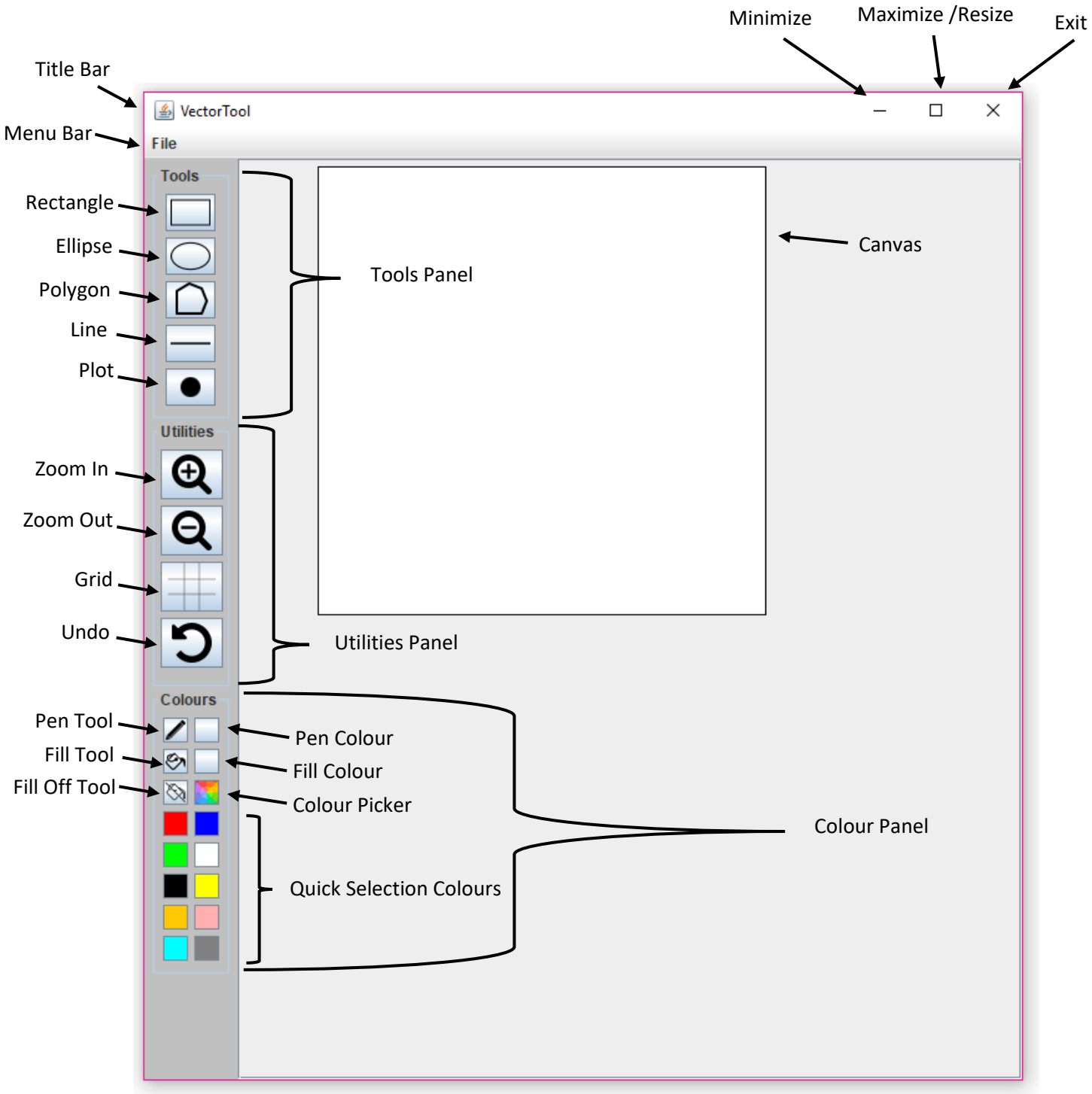
Inheritance is important as it promotes code reuse as everything that is a subclass will inherit anything from its superclass. This project uses inheritance extensively to minimise code redundancy and offer flexible and scalable program architecture. Within the context of the shape tools, inheritance was used in conjunction with abstraction. The base SuperClass VectorShape is inherited directly by the line, plot and polygon tools. The ellipse and rectangle tools, however, inherit from BoxLikeShape which extends VectorShape. Each subclass is more specific but still fits under the encompassing superclass. This structure of inheritance allows for shapes with similar attributes to be grouped and duplicate code to be minimised.

Polymorphism is useful when the superclass is known, but not the subclass. It makes it easier when there is an unknown about what exact subclass is going to be used. In the software, an example of polymorphism is used for the getString and parseShapes used in opening and saving files. Not every shape is going to be known, so by having these methods in the superclass, these shape subclases are still affected by the methods and allow the software to work as intended. This is similar to the toString method that converts an object to a string to make it more usable.

How to Use Application

Application layout

The image below shows what is displayed when the application is executed. The surrounding text describes the layout of the application.



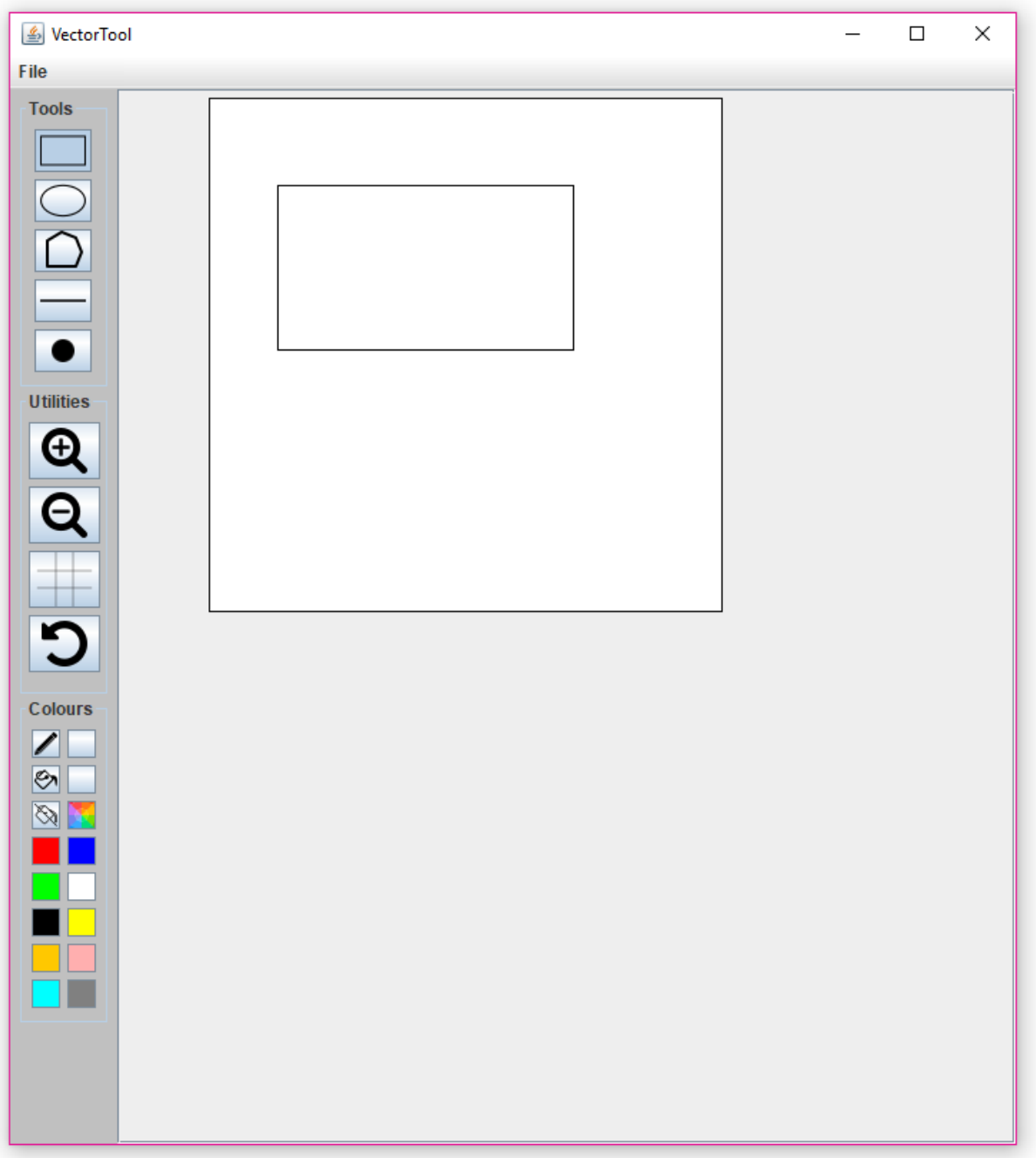
How to use buttons on Tools Panel

The Tools Panel holds buttons that determine what shape will be drawn. To select a shape, click on the corresponding button. The button colour will change to a darker shade when successfully selected. The location of the mouse will determine the points of the shapes. The following sections will detail instructions on how to draw each shape.

Rectangle

Requires two mouse clicks. An outline of the final shape is displayed.

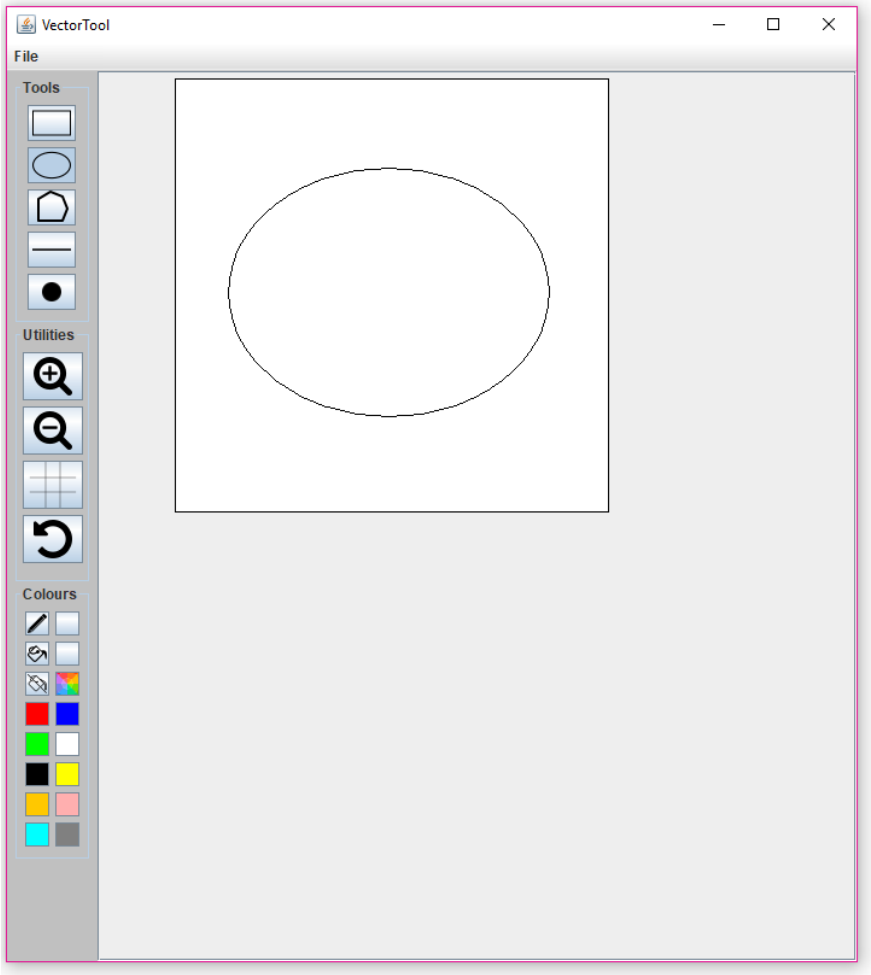
1. Click on the canvas to set top left corner of shape
2. Click again to set bottom right corner of shape



Ellipse

Requires two mouse clicks. An outline of the final shape is displayed.

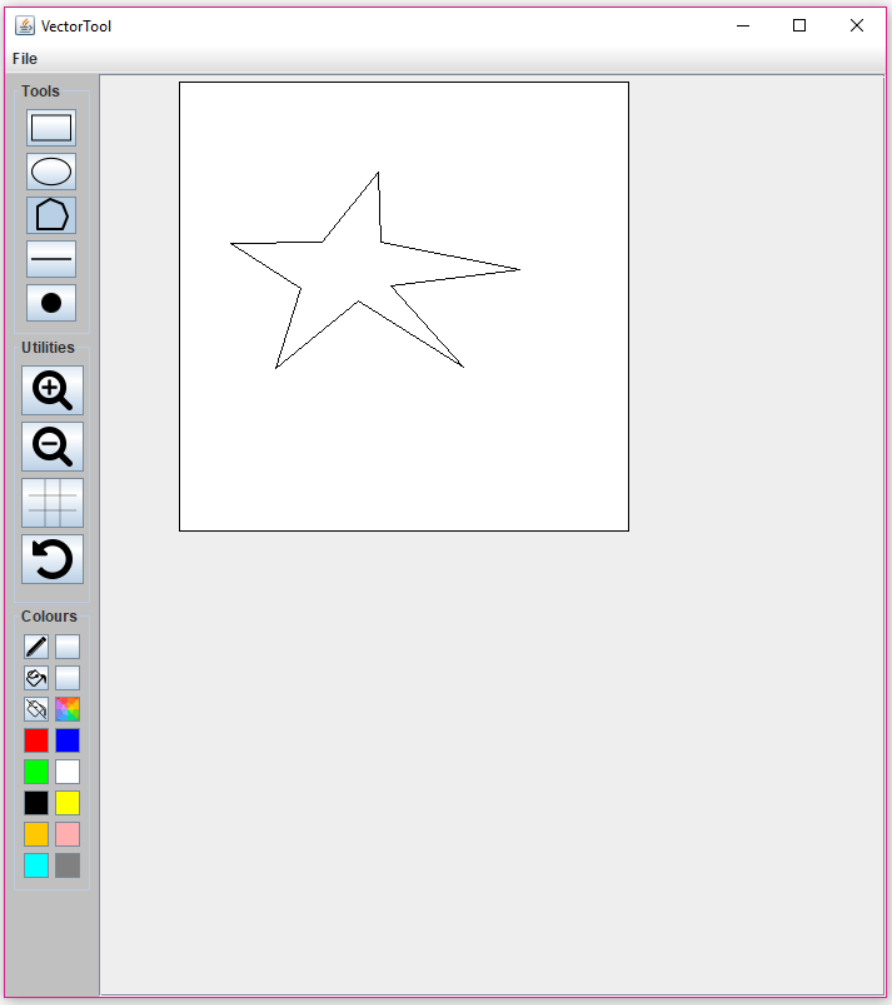
1. Click on the canvas to set top left corner of shape
2. Click again to set bottom right corner of shape



Polygon

Does not have a limit on mouse clicks. Each click forms another corner/point of the polygon. An outline of the next polygon side is always shown and is determined on the location of the mouse. The last side of the shape will be between the first point and current location of the mouse.

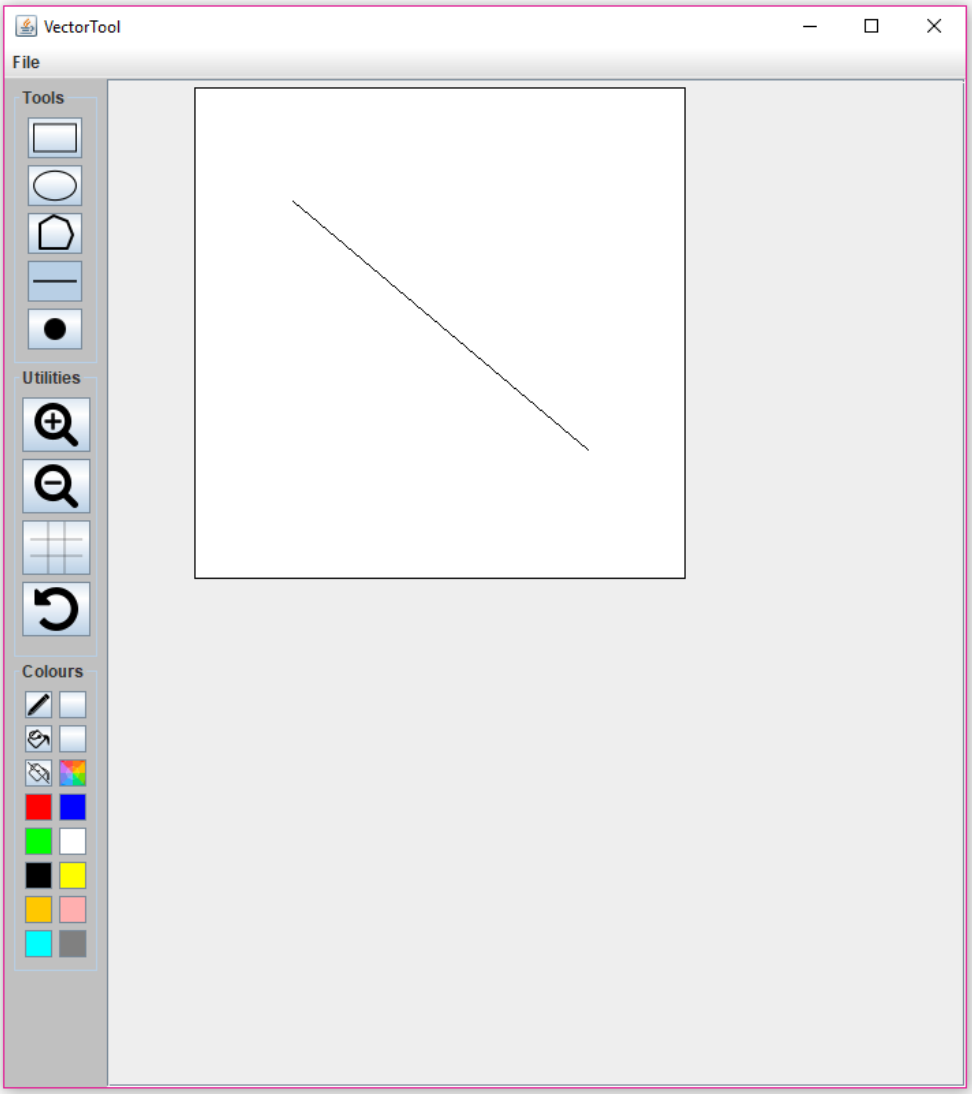
1. Click on the canvas to set starting point of shape
2. Continue clicking on canvas to form desired shape
3. To complete shape, place mouse on last corner of shape and press the *Enter* key



Line

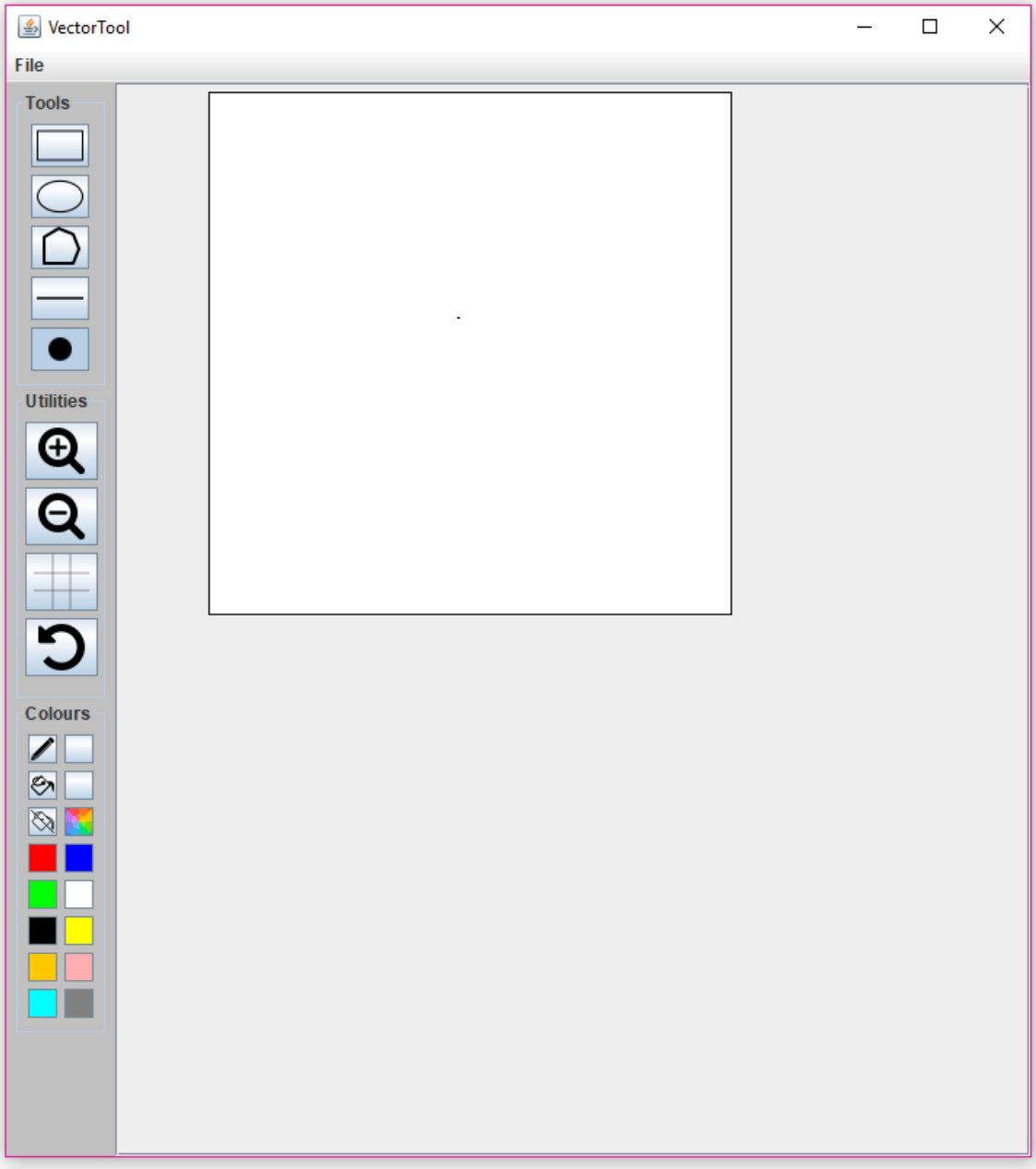
Requires two mouse clicks. An outline of the final shape is displayed.

1. Click on the canvas to set starting point of shape
2. Click again to set ending point of shape



Plot

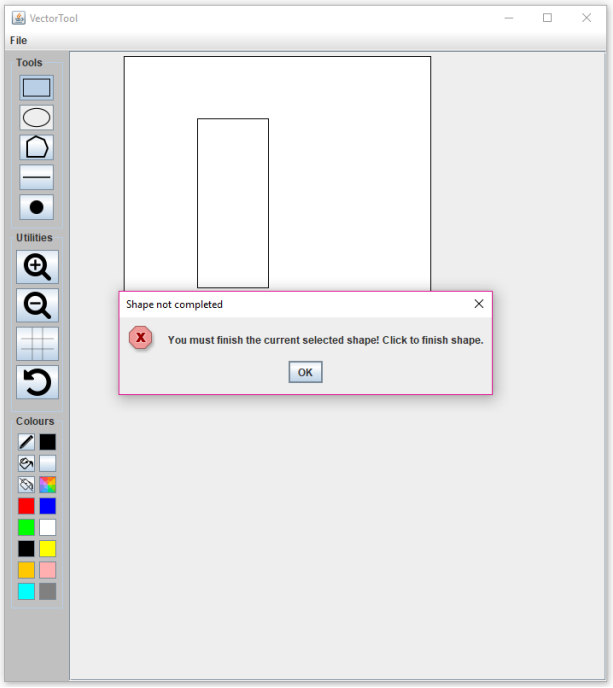
1. Click on canvas to set point



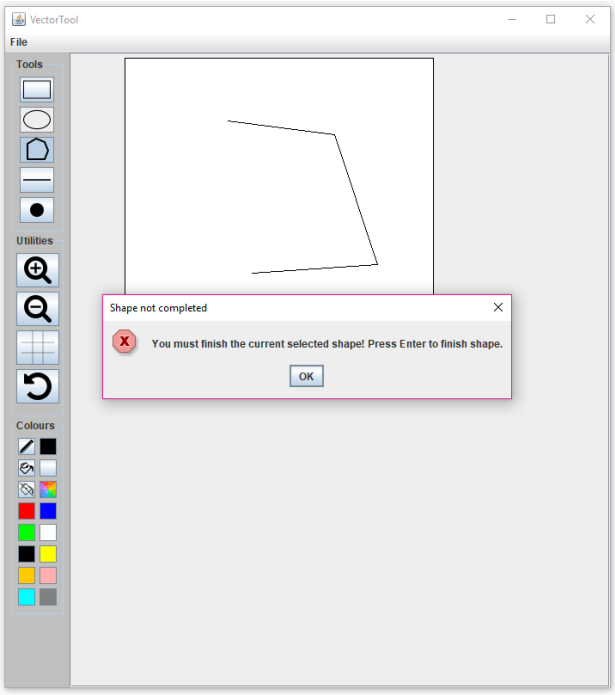
Selecting another shape without completing the current shape

A dialog box will appear if the user has begun drawing a shape, but then decides the select another tool without completing the current shape. This dialog box informs the user, that the current shape needs to be completed before selecting another shape tool and gives instructions on how to complete shape. There are two types of dialog boxes, one for Polygon and the other for every other Tool except Plot.

The images below show both dialog boxes.



Dialog box for Rectangle Tool



Dialog box for Polygon Tool

How to use buttons on Utilities Panel

The Utilities Panel holds buttons that execute utility functions on the canvas. Each utility can be selected by clicking on it. However, if the button is clicked and held on, and the mouse is dragged out of the button, the utility will not be selected. The following sections will detail instructions on how to use each utility.

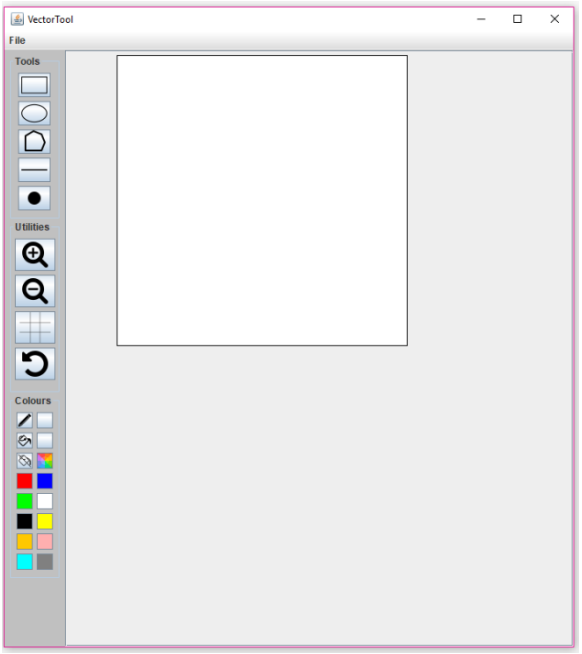
Zoom In

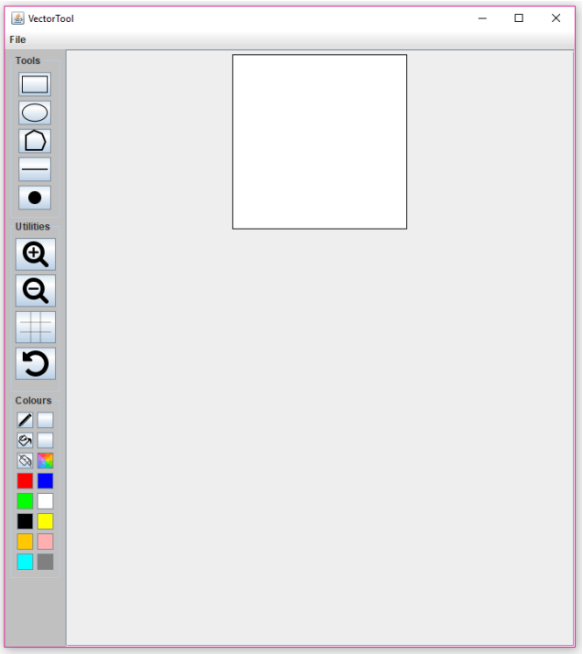
Clicking button will cause the canvas to be enlarged. Subsequent clicks will continue to enlarge the canvas by the same amount. If the window is too small, a scroller bar on either the X or Y axis will be presented to scroll along the canvas.

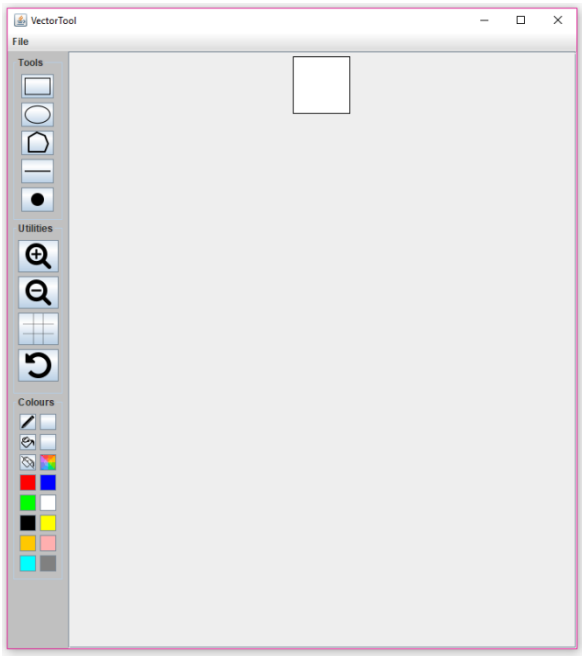
Zoom Out

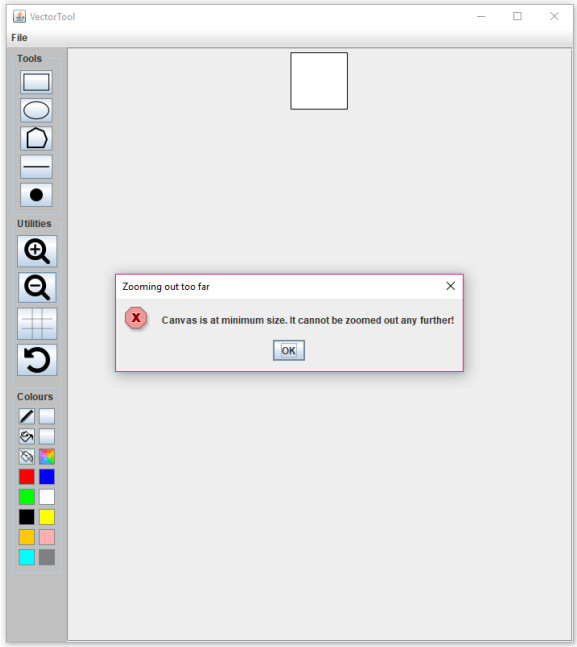
Clicking the button will cause the canvas to be scaled smaller. Subsequent clicks will continue to scale down the canvas by the same amount. If the canvas is zoomed out so it is no longer visible, a dialog box will appear, and the canvas will be zoomed in to the previous size. The dialog box is used to inform the user that the canvas cannot be zoomed out any further.

The images below show the Zoom Out function being used until the dialog box appears.







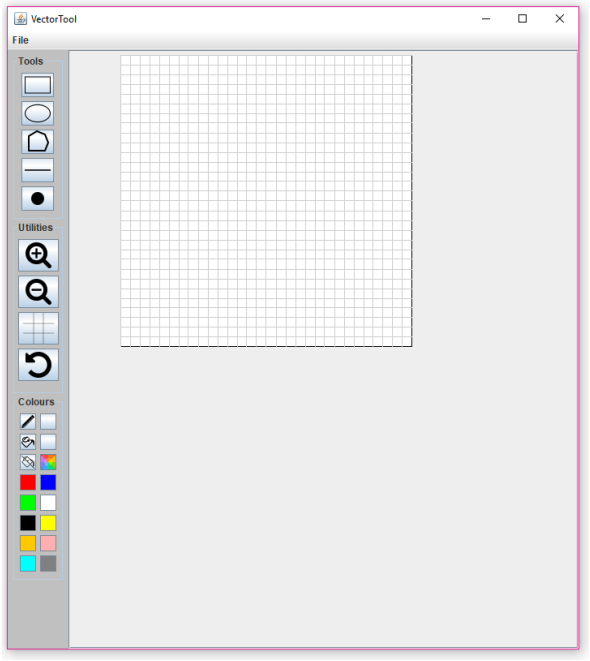


Dialog Box showing user canvas is at minimum size

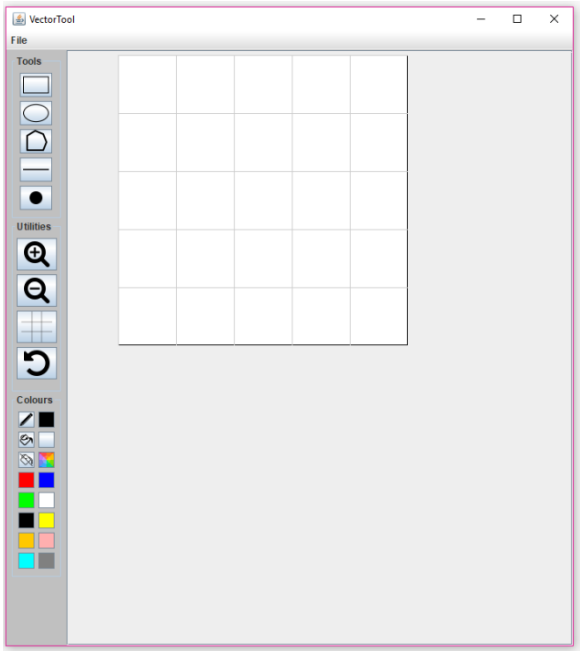
Grid

Clicking the button will toggle a grid layout to be displayed on the canvas. Subsequent clicks will toggle the Grid ON/OFF state. The default state of the Grid is OFF. The Grid size can be changed by pressing Ctrl + Arrow Key (Up or Down). Arrow Key Up will make the Grid larger, whilst Arrow Key Down will make the Grid smaller.

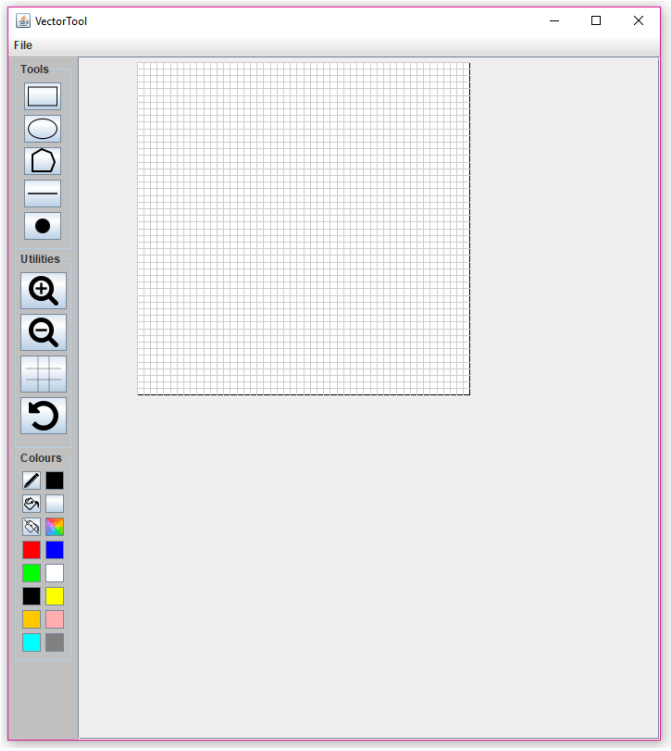
The images below show the original Grid size and its maximum and minimum sizes.



Normal Grid size



Minimum Grid size

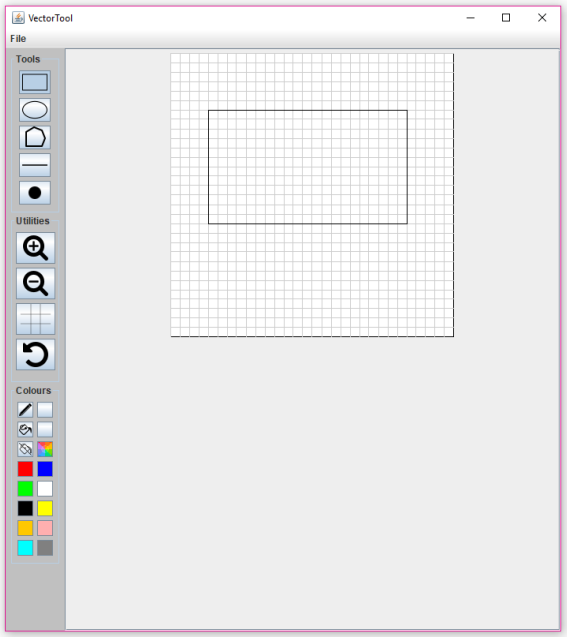


Maximum Grid size

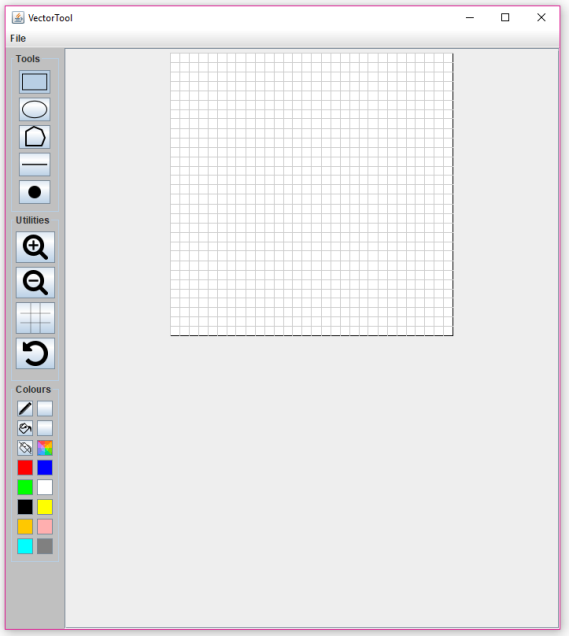
Undo

Clicking the button will remove the last shape drawn on the canvas. Subsequent click will continue to remove the next last drawn shape. If there are no more shapes to be removed and the button is clicked, a dialog box will appear alerting the user that there are no shapes to be undone. Another dialog box is also displayed when the user tries to use Undo whilst drawing a shape. The Undo function can also be used by pressing Ctrl + Z on the keyboard.

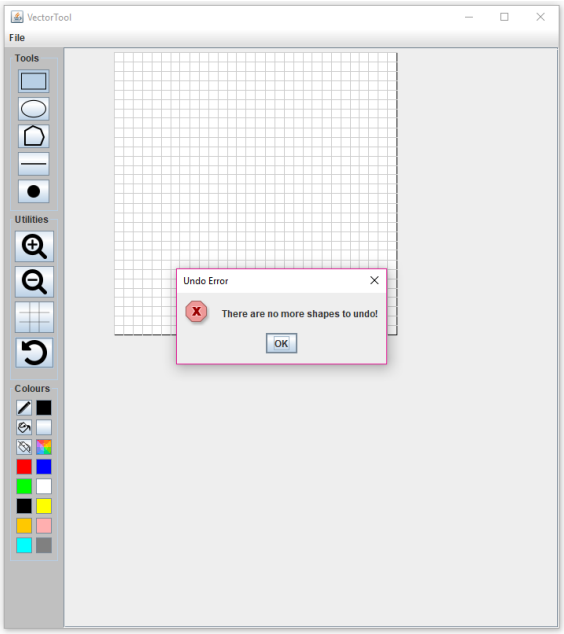
The images below show the Undo function being used and situations where the dialog boxes are presented.



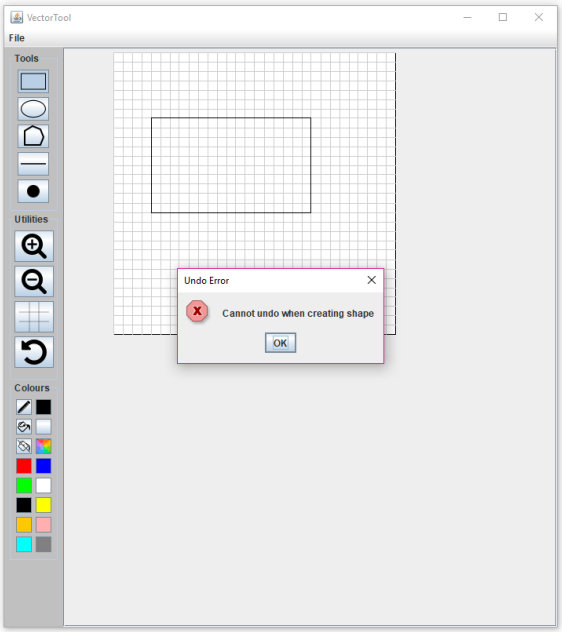
Rectangle shape drawn



Undo function used to remove last shape drawn



Dialog box displayed because there are no shapes to be undone



Dialog box because Undo function was used whilst shape is being drawn

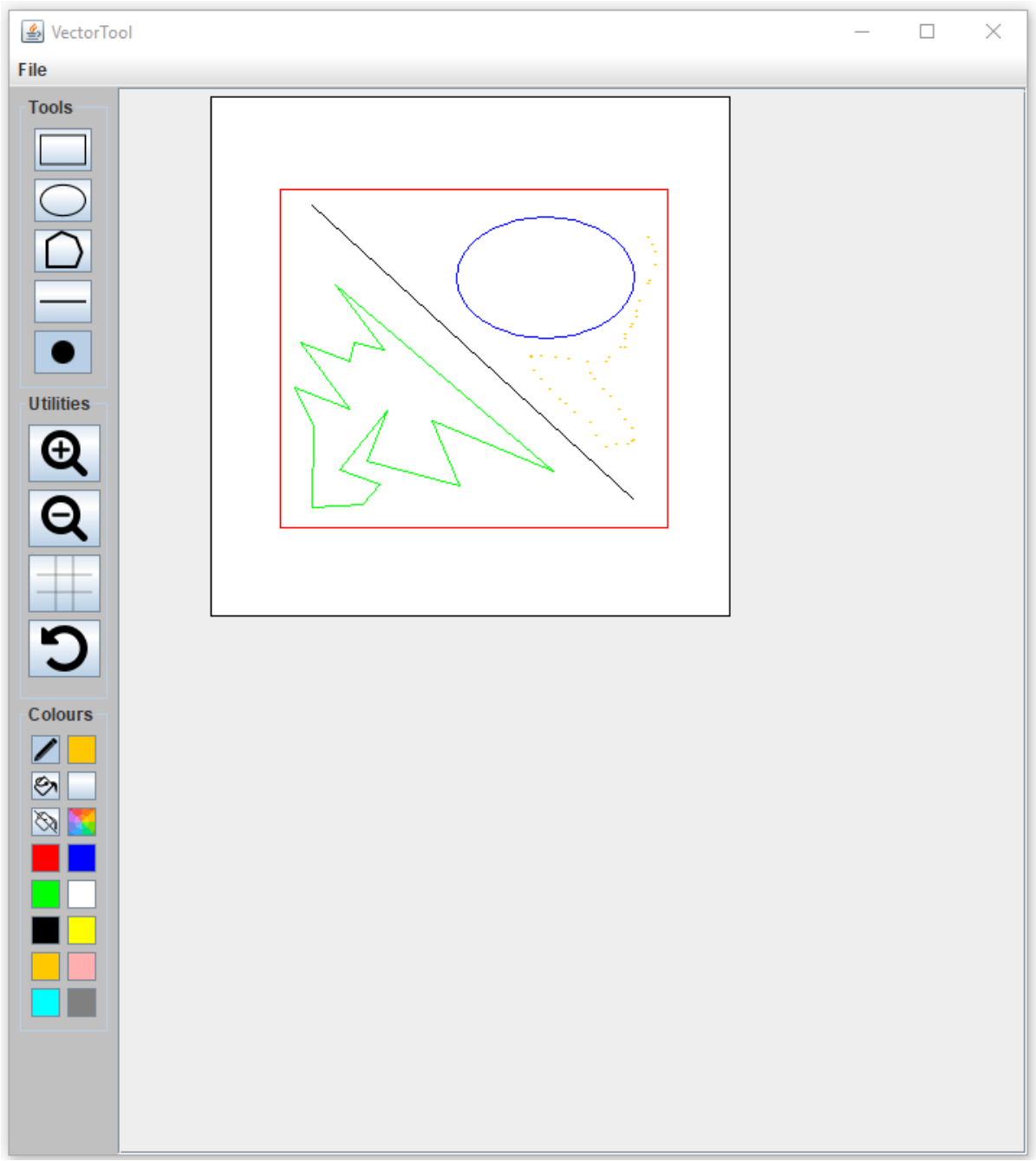
How to use buttons on Colours Panel

The Colours Panel holds buttons that determine how the drawn shape will be coloured. The Pen, Fill and Fill Off buttons turn to a darker shade when successfully selected. The following sections detail instructions on how to use these buttons.

Pen Tool

Clicking on the button will allow the user to choose the outline colour of the shape. The Pen Colour button displays the current Pen Tool colour. The default colour is *black* which is set upon initialization of application. Colours can be selected from either the Colour Picker or Quick Select Colours.

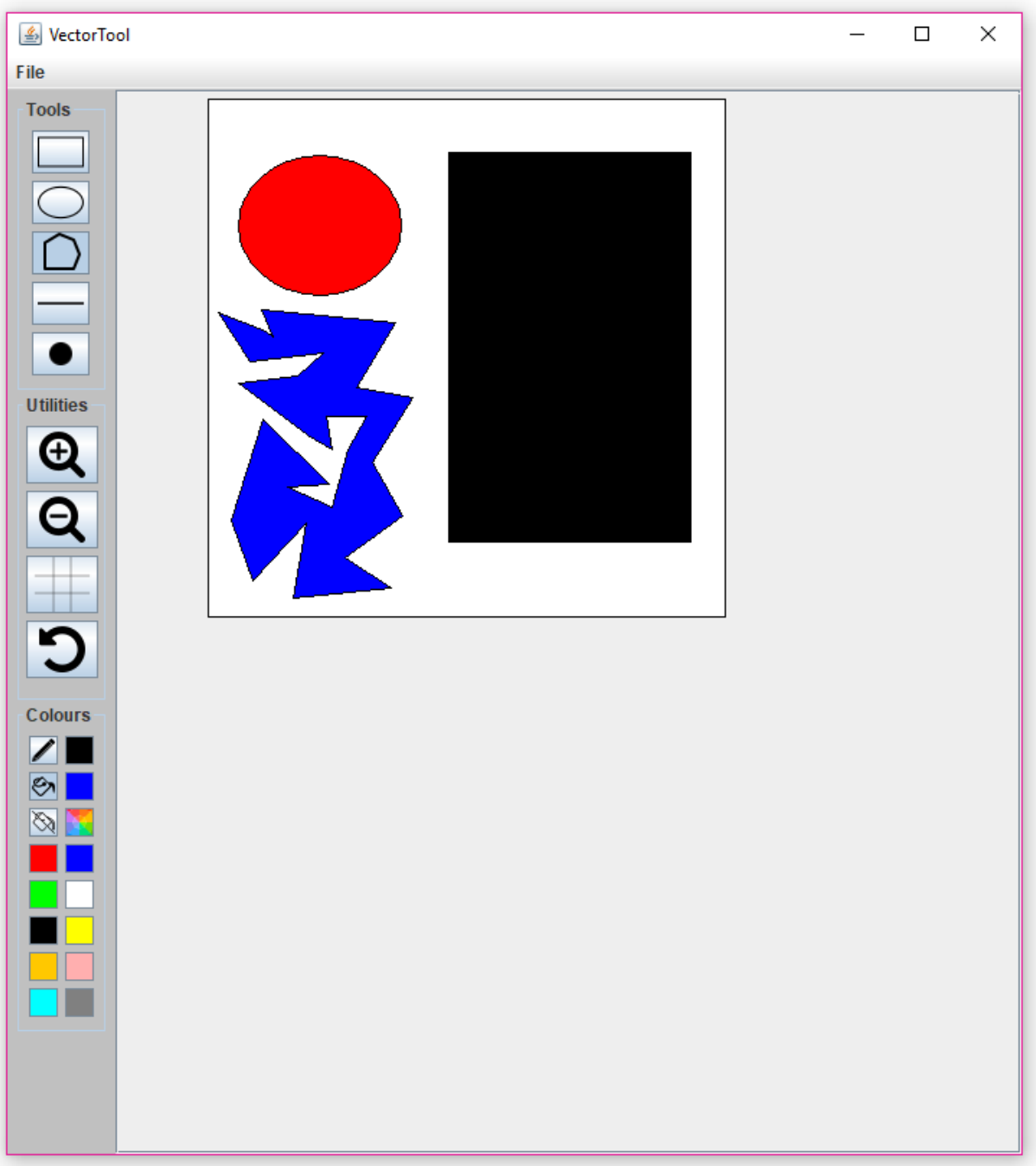
The images below show all five drawing tools drawn with different pen colours.



Fill Tool

Clicking on this button will allow the user to choose the fill colour of the shape. The Fill Colour button displays the current Fill Tool colour. The default colour is *No Fill* which is set upon initialization of application. Colours can be selected from either the Colour Picker or Quick Select Colours. Only Tools Rectangle, Ellipse and Polygon can use the Fill Tool.

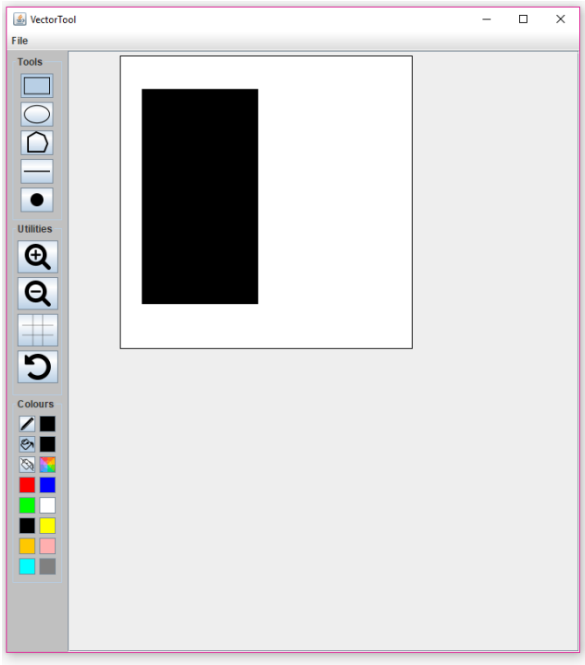
The images below show the relevant drawing tools drawn with different fill colours and with the default pen colour *black*.



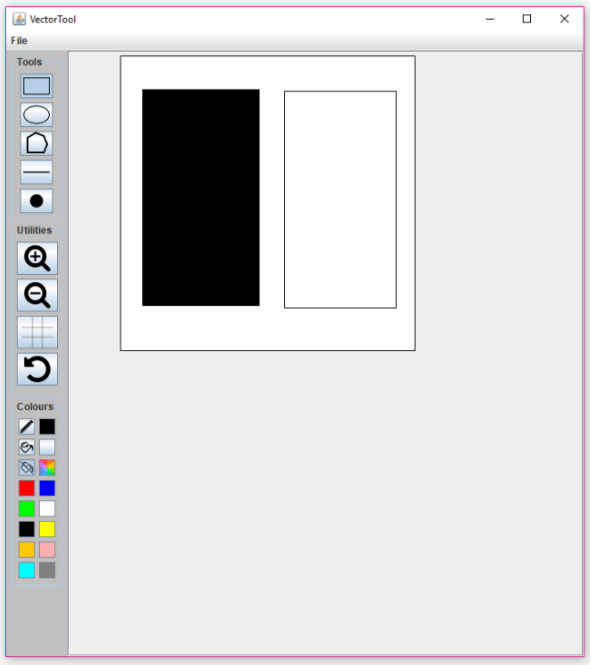
Fill Off Tool

Clicking on this button will clear the current Fill colour. This is seen as the Fill Colour button colour will turn to *No Fill*. If the user tries to select a colour, whilst the Fill Off Tool is selected, a dialog box is displayed alerting the user, that either a Pen or Fill Tool must be selected.

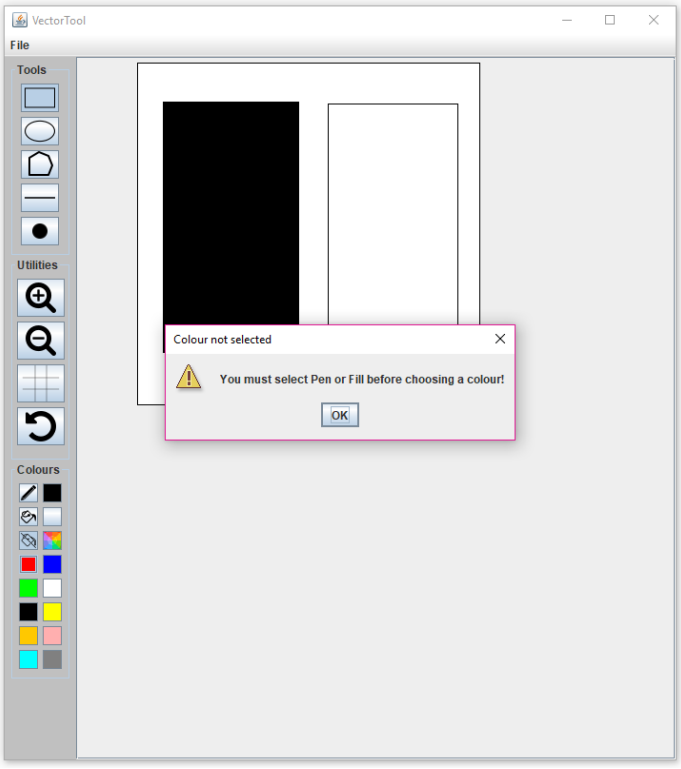
The images below show the Rectangle Tool being drawn with, and without a fill colour, and the dialog box displayed when a colour is selected without first selecting a Pen or Fill Tool.



Fill Tool used to set fill colour to *black*



Fill Off Tool used to clear fill in new shape



Dialog box displayed because Pen or Fill Tool not selected

Colour Picker

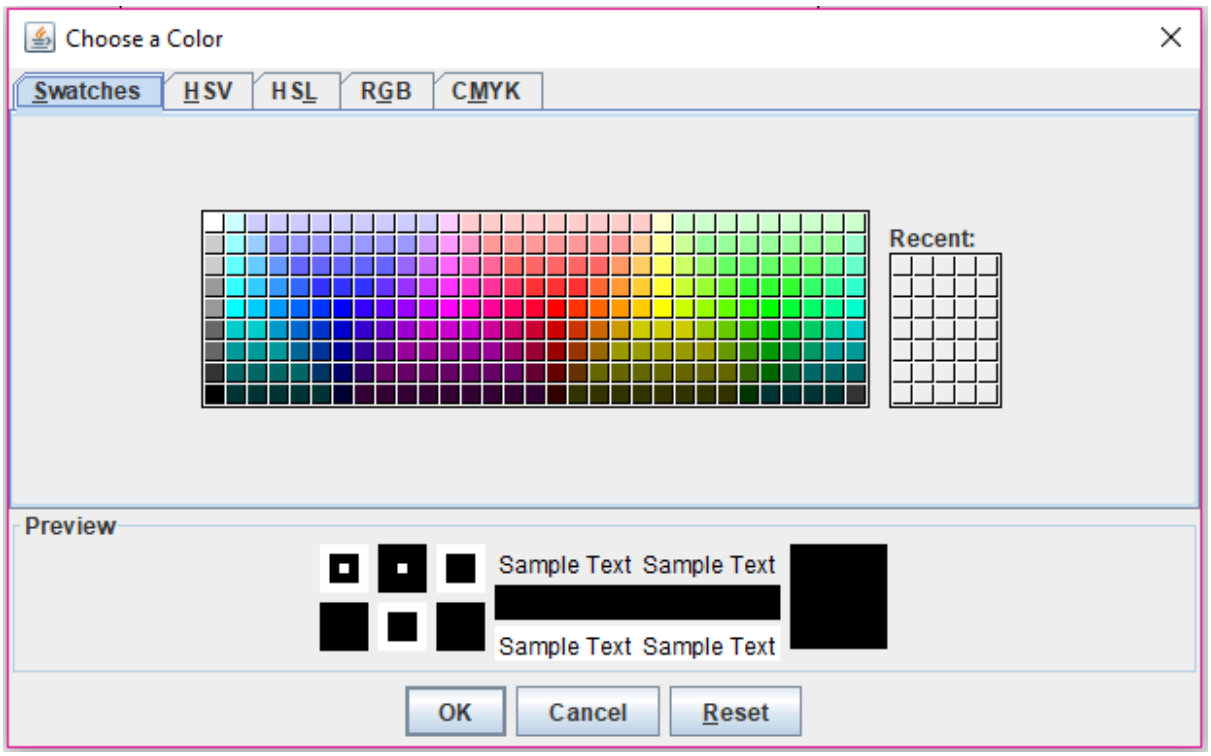
Clicking this button will display a new window allowing the user to select any possible colour. This is done through a range of tabbed panes. The tabbed panes include, Swatches, HSV, HSL, RGB and CMYK. A Preview section is displayed in each pane which allows the user to see what colour will be set. The bottom section holds buttons OK, Cancel and Reset,

- OK – Set Tool colour to be colour in the Preview section
- Cancel – Close the Colour Picker and Tool colour remains the same
- Reset – Colour Picker will return to default colour *black*

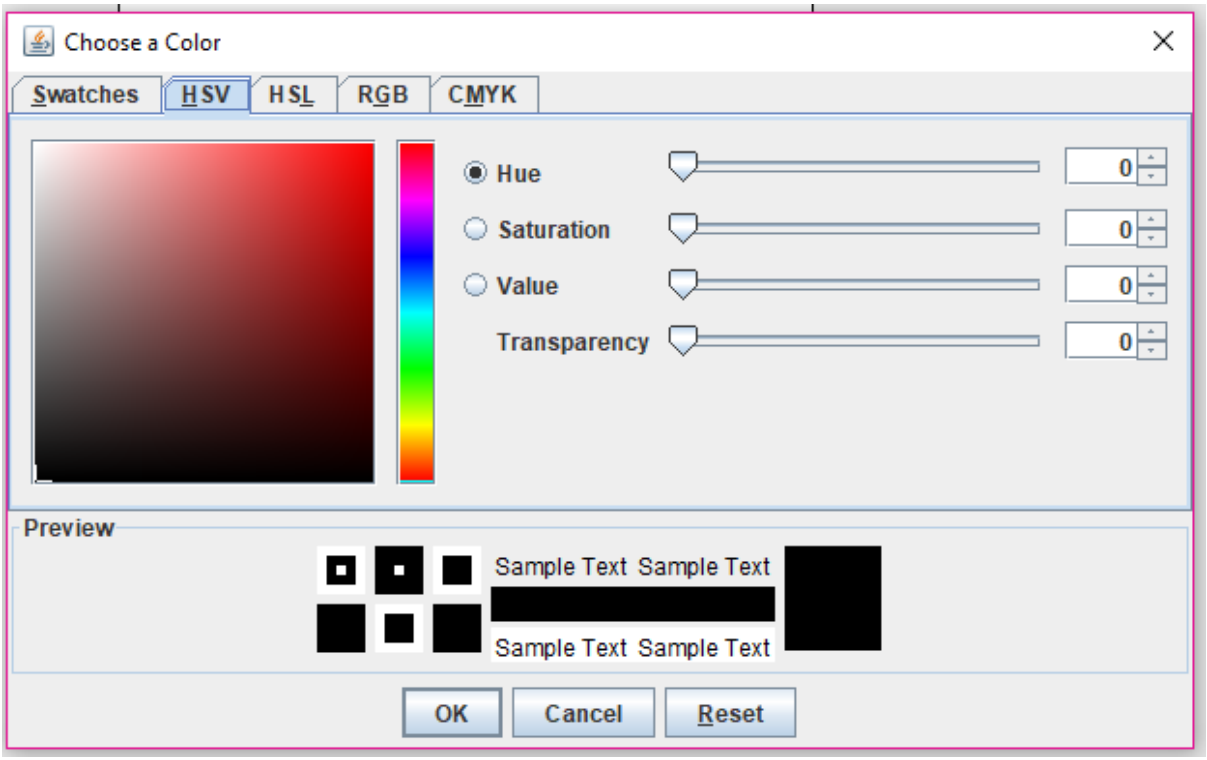
The selected colour will change the colour of the Pen or Fill Colour, depending on the Tool chosen.

The images below show the Colour Picker tabbed panes, the functionality of the OK, Cancel and Reset buttons, and a Rectangle being drawn with the selected colour.

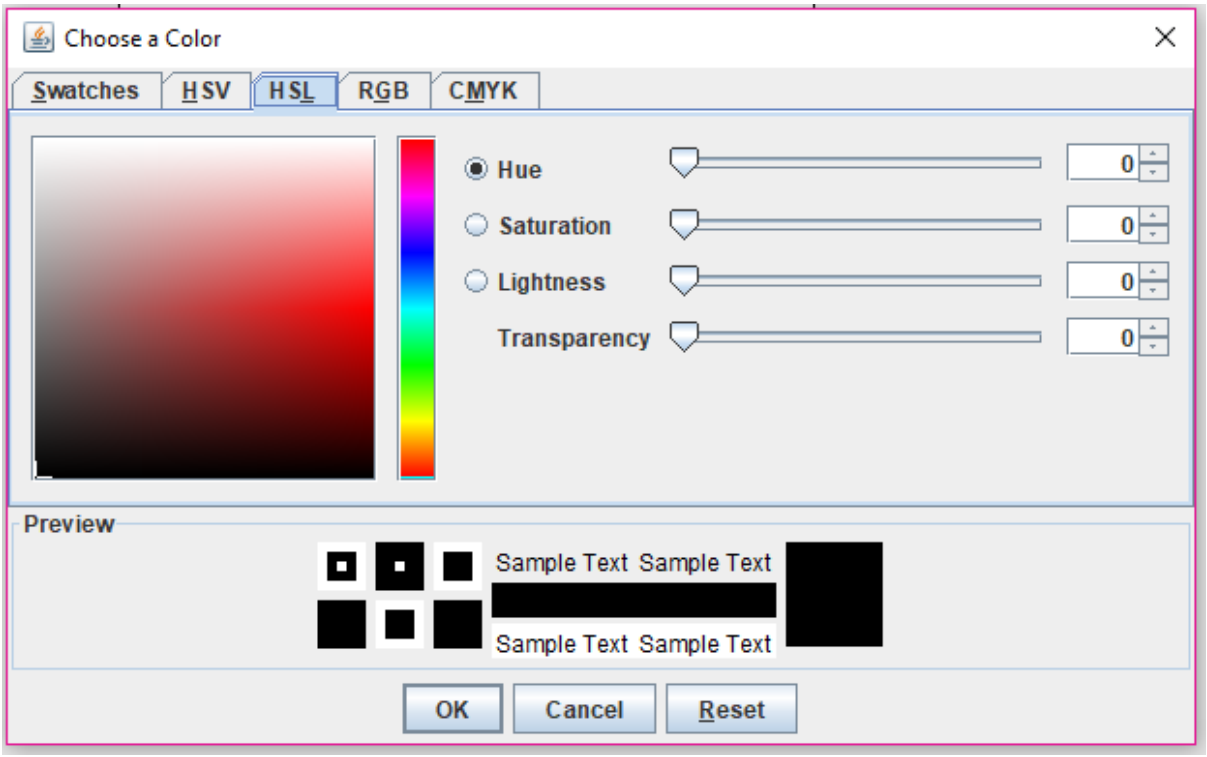
Tabbed Panes



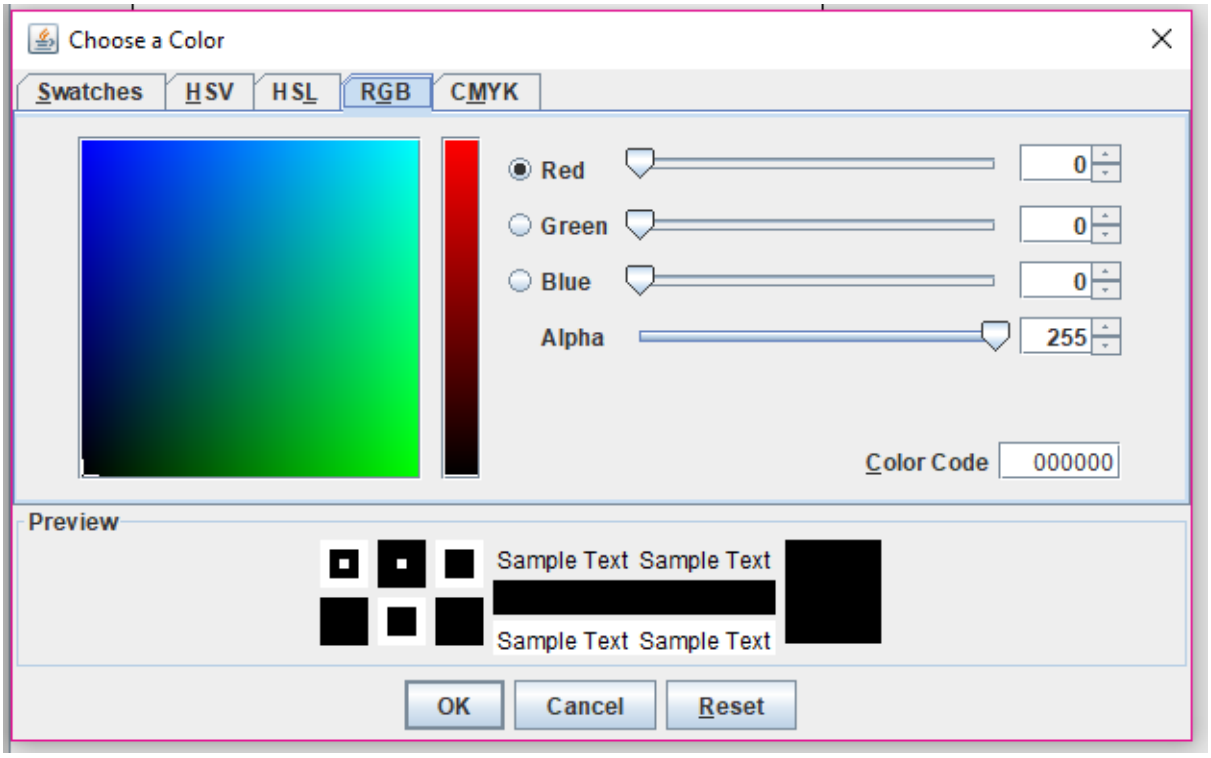
Swatches Tab



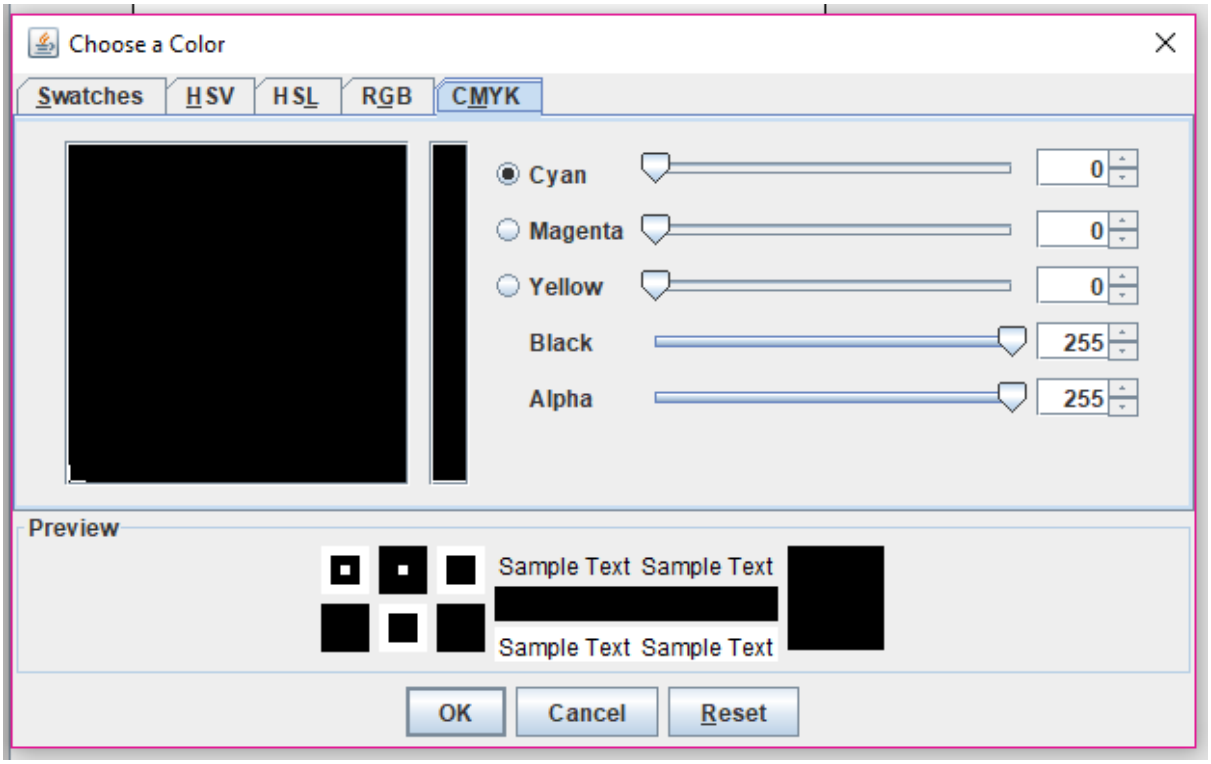
HSV Tab



HSL Tab

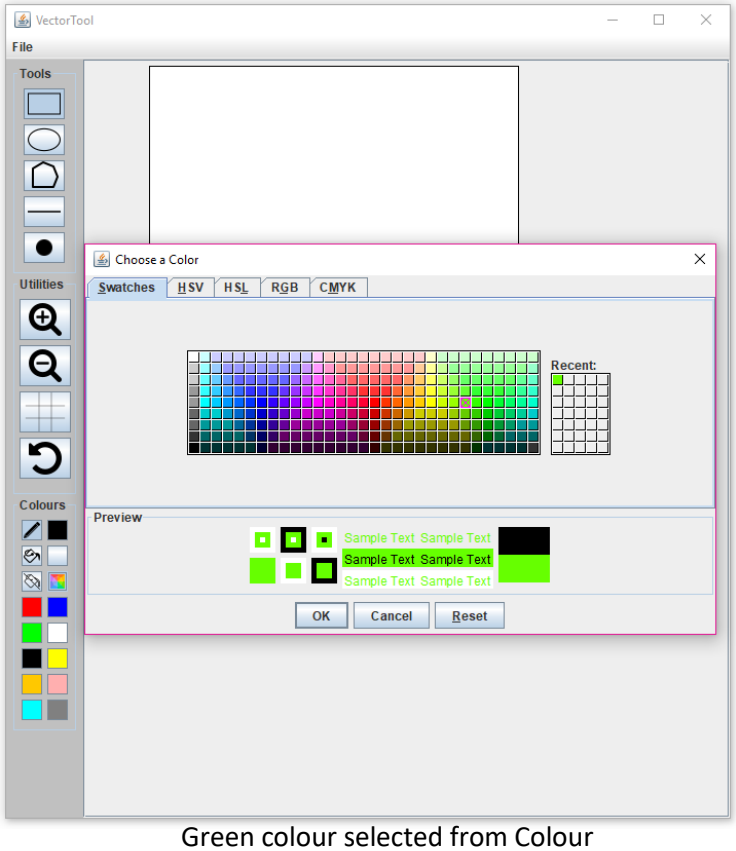


RGB Tab

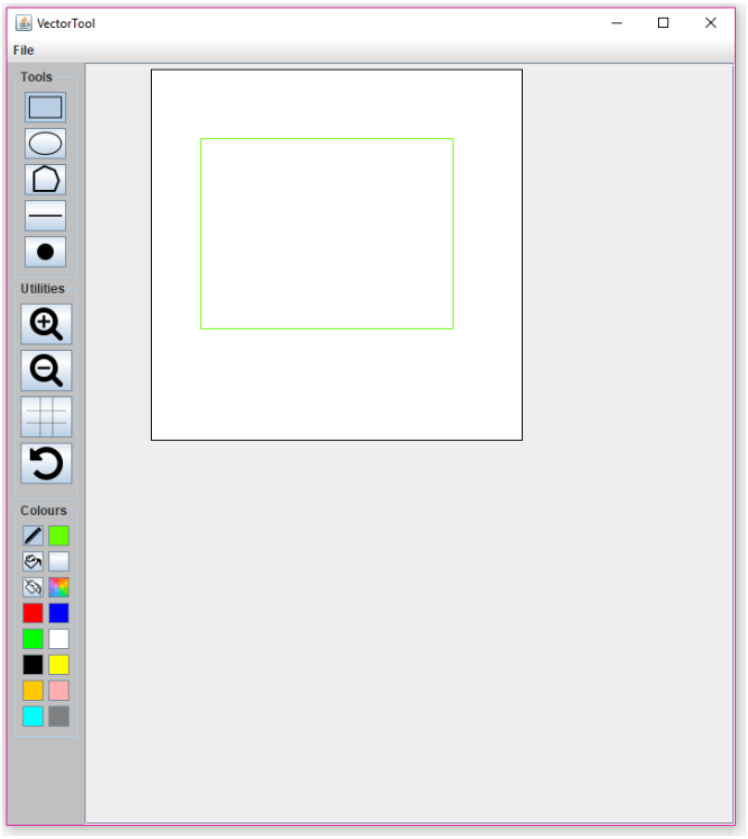


CMYK Tab

OK button to select colour

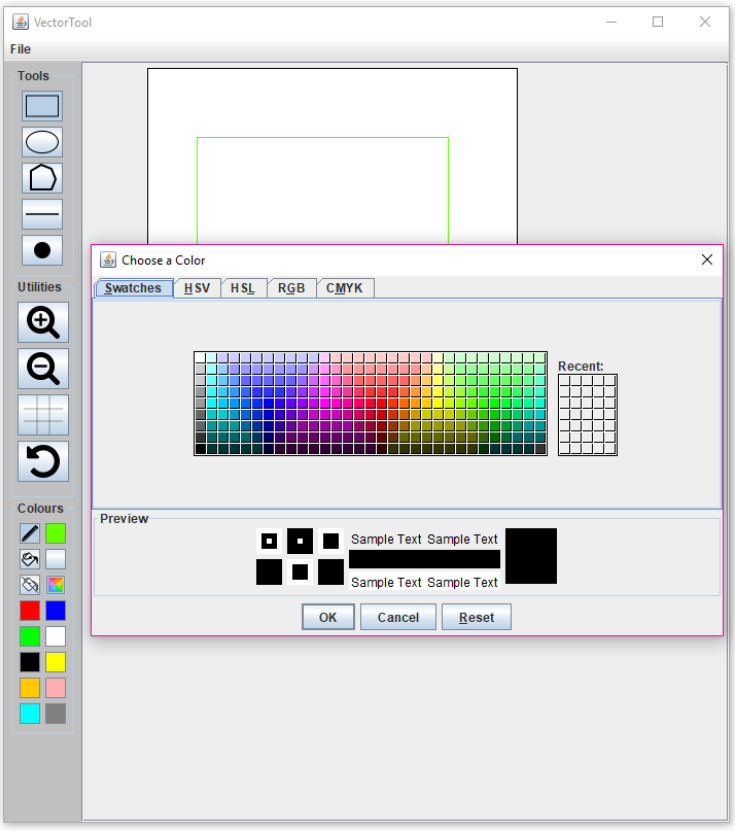


Green colour selected from Colour

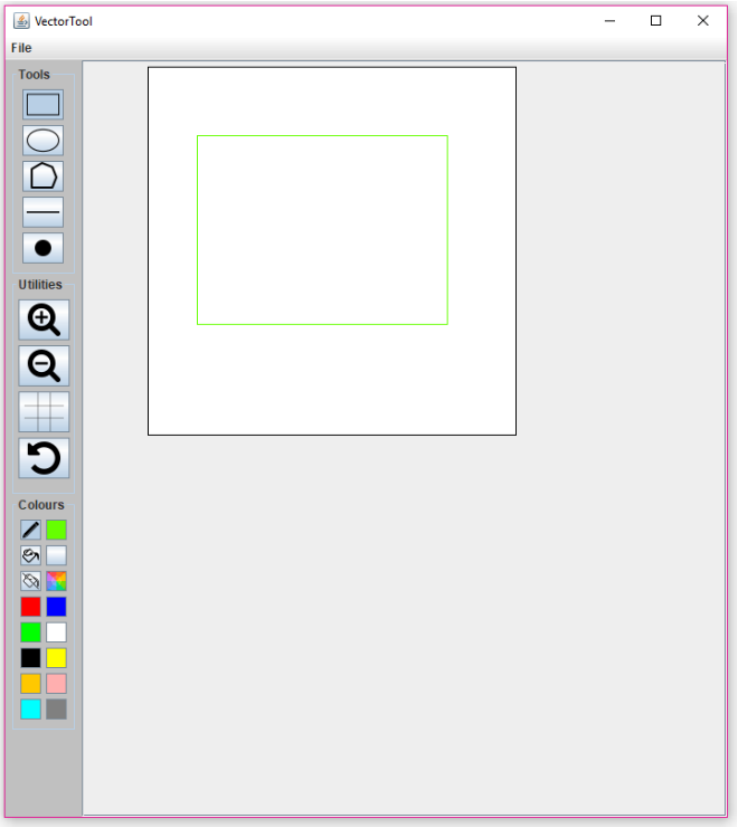


Rectangle Tool used to draw shape

Cancel button to exit Colour Picker

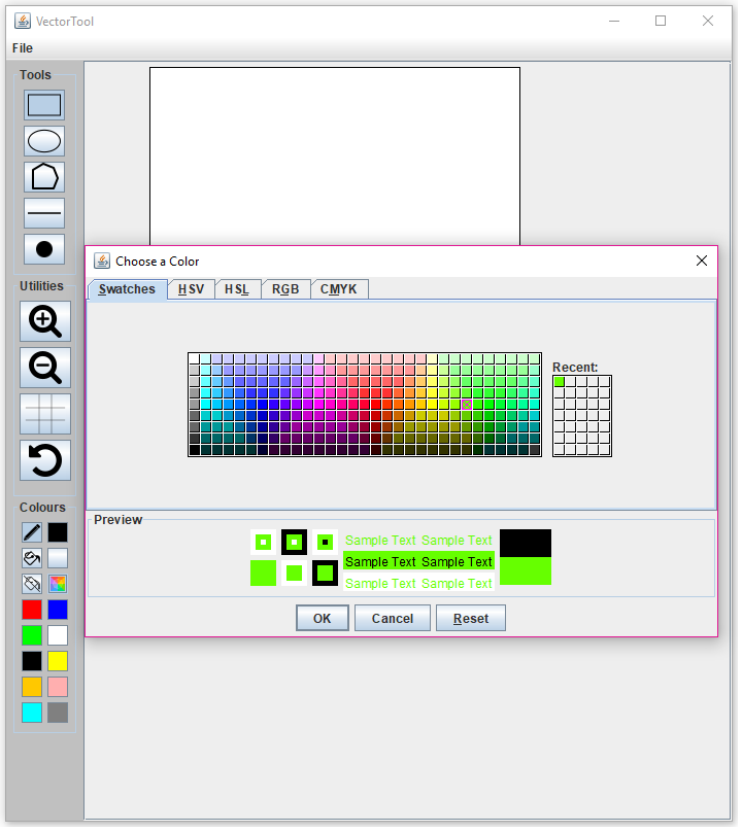


Colour Picker open

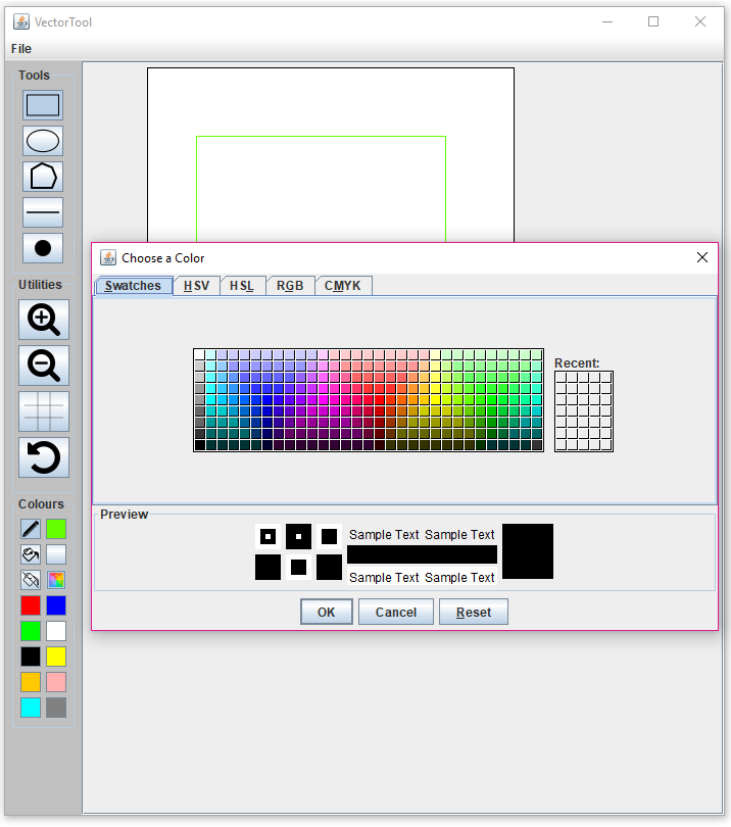


Colour Picker closed with same Pen colour

Reset button to return to default colour

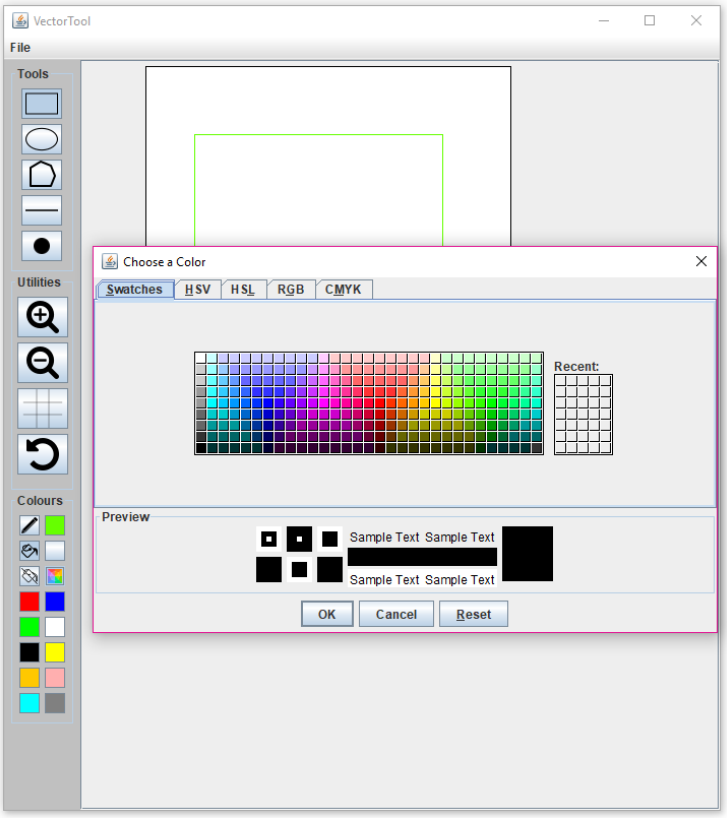


Colour green selected



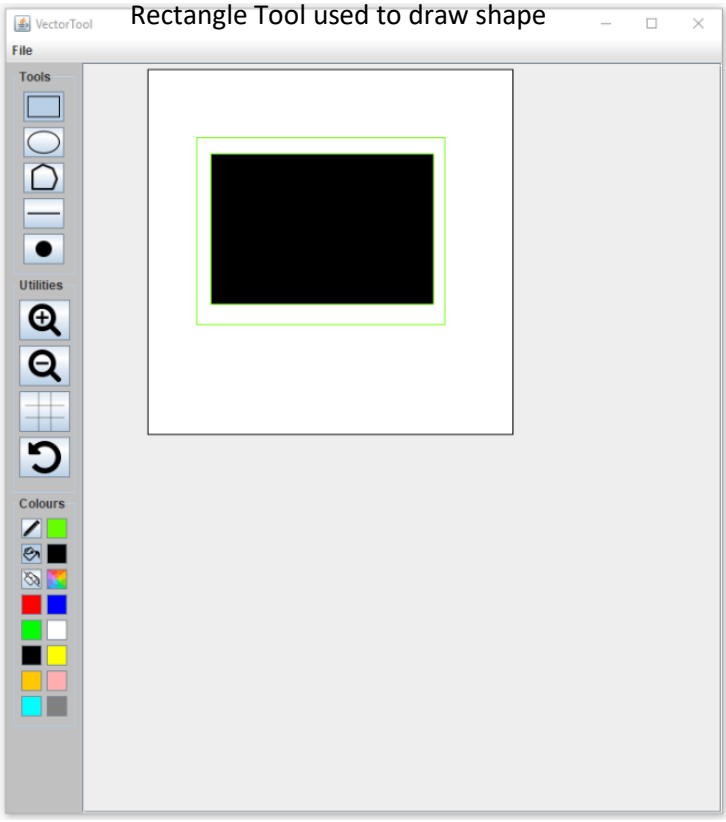
Colour set to default colour

Colour Picker used to change the Fill colour and Rectangle Tool used to draw shape



Select colour *black* from Colour Picker

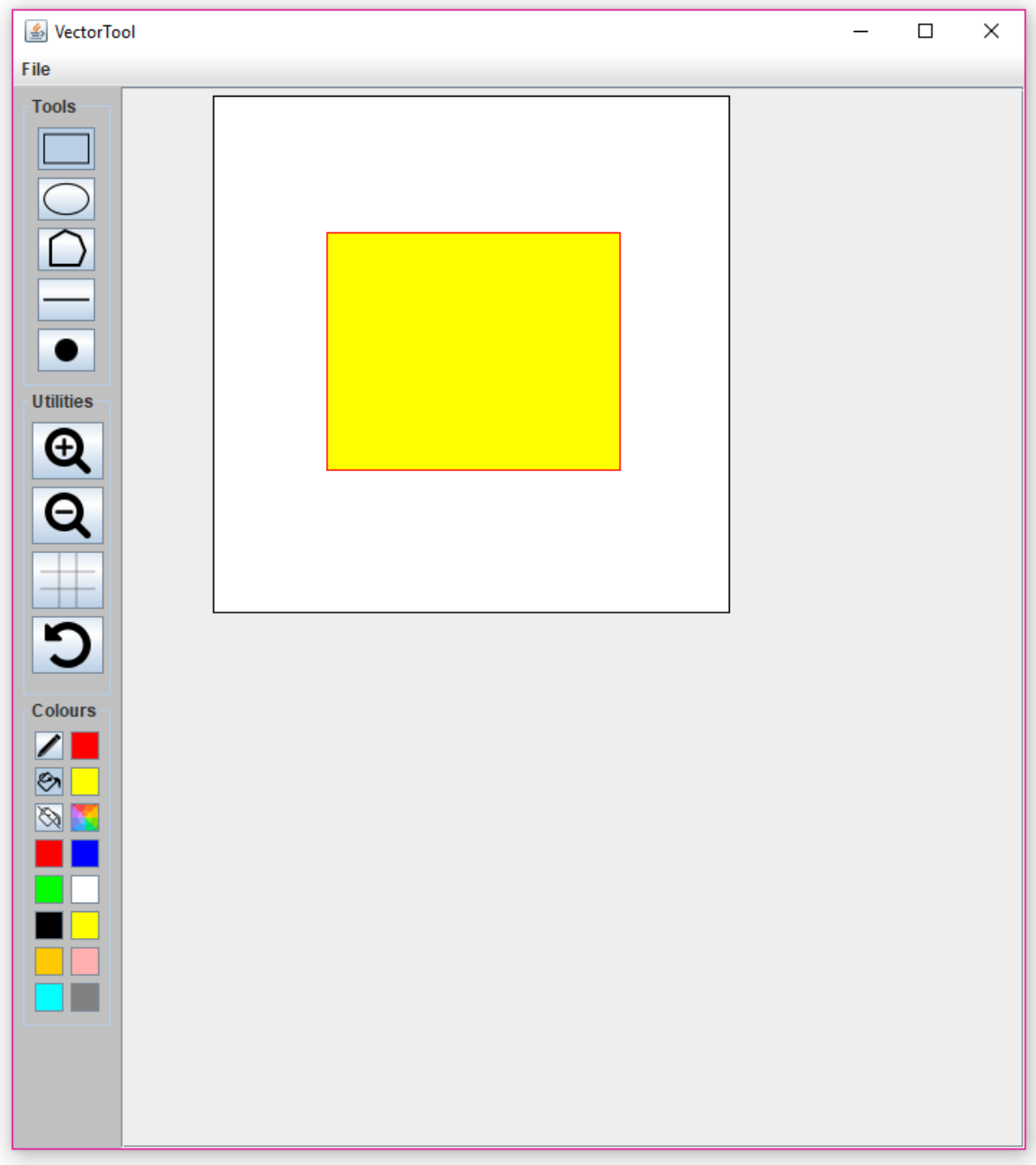
Rectangle Tool used to draw shape



Quick Select Colours

Clicking on one of these buttons will set the colour for the current Tool selected. The colour of the button will be the same colour being set. This will change the colour of the Pen or Fill Colour to the colour being set.

The image below show the Quick Select Colours being used to change the Pen and Fill Tool colours.

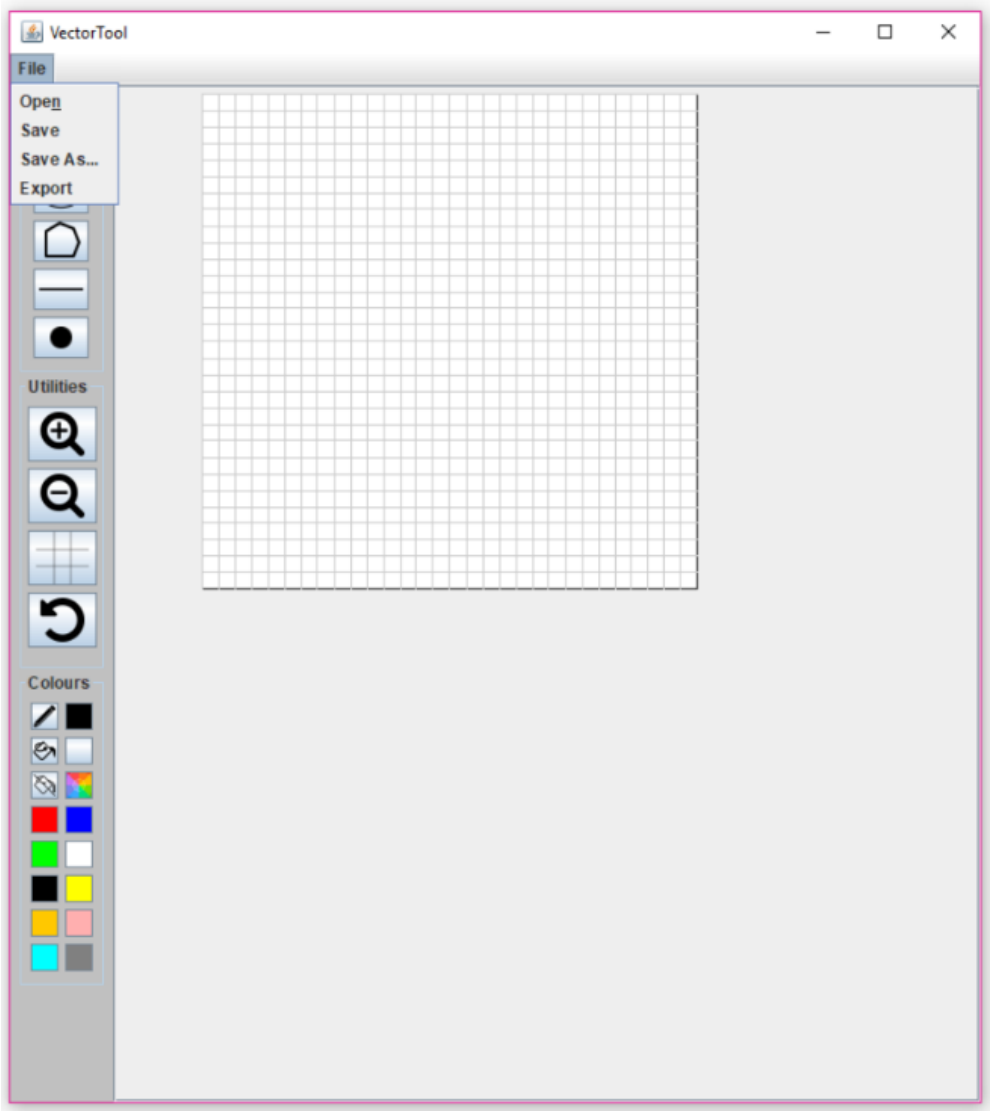


How use to File operations

The File tab on the Menu Bar is used for file operations. The tab contains five menu items,

- Open – a file system window is displayed and used to open .VEC files
- Save – save state of canvas
- Save As… - a file system window is displayed where the location and name of the file can be determined
- Export – operates like the Save As… function but the file format is Bitmap

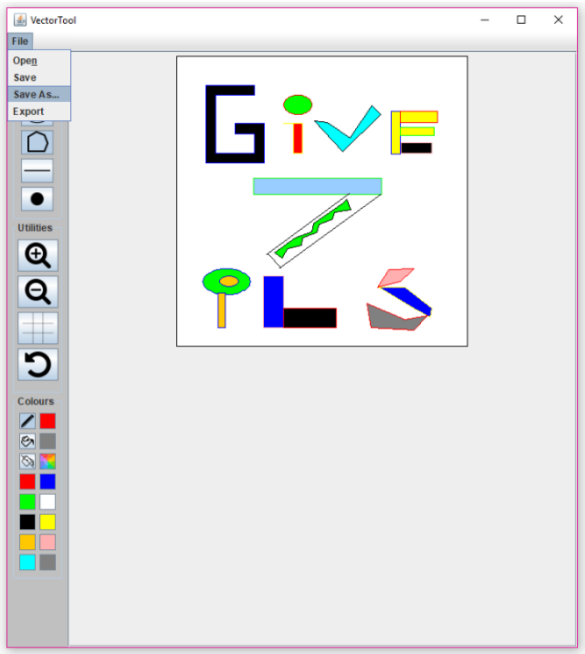
The sections below detail instructions on how to use the file operations.



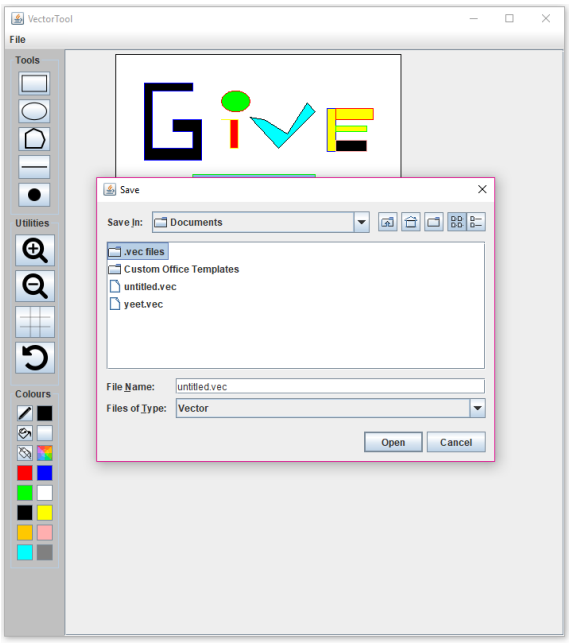
Contents of File Tab

Using Save As…

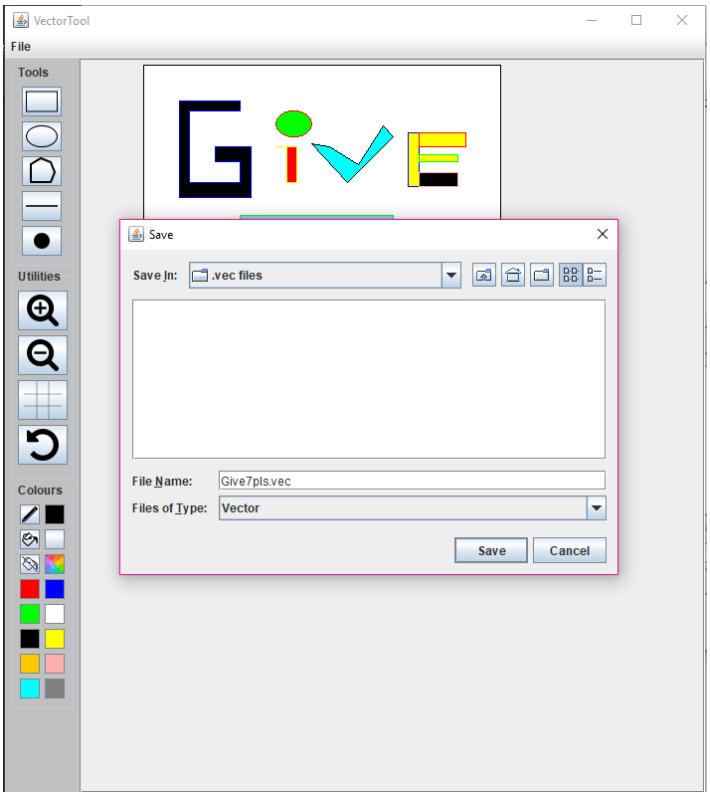
1. Click *Save As…* in File tab
2. Choose location to save file in File System
3. Choose name of file to be saved and click *Save*



Select Save As…



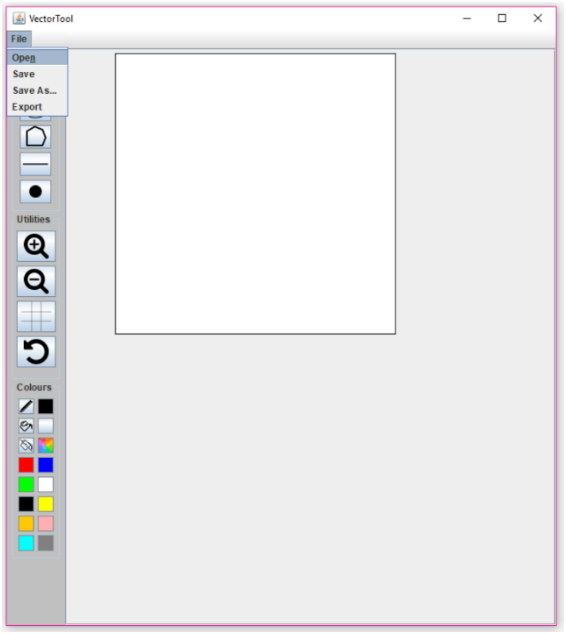
Contents of File Tab



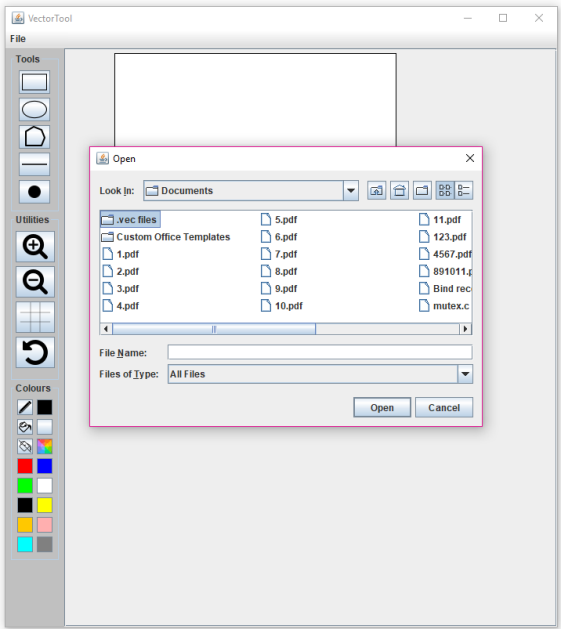
Click *Save* to compete Save procedure

Using Open

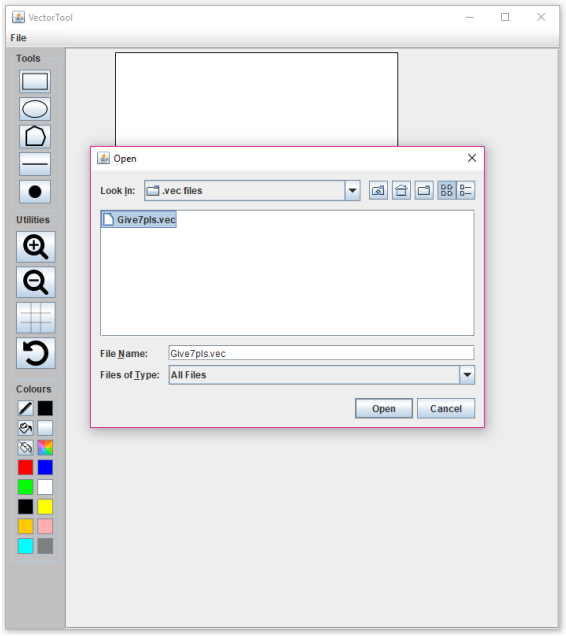
1. Click *Open* in File tab
2. Navigate File System to file to be opened
3. Select file and click *Open*



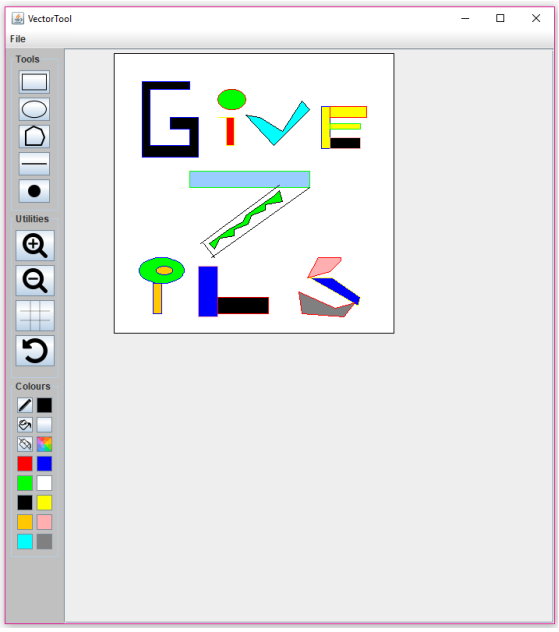
Click *Open*



Folder where file is saved



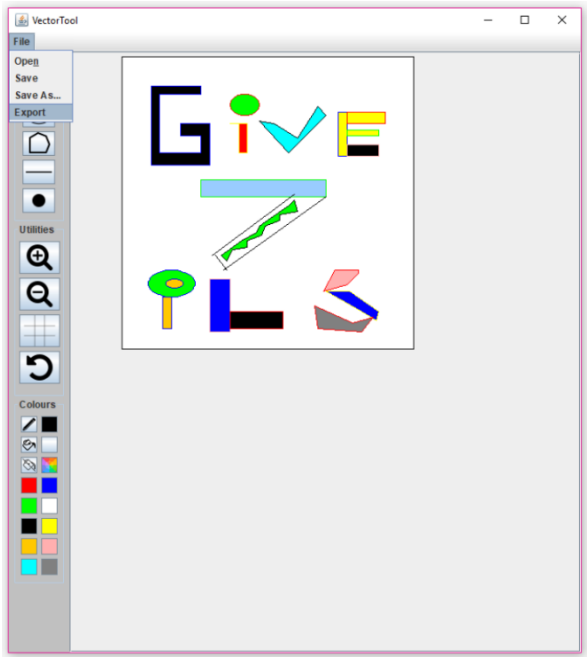
Select file and click *Open*



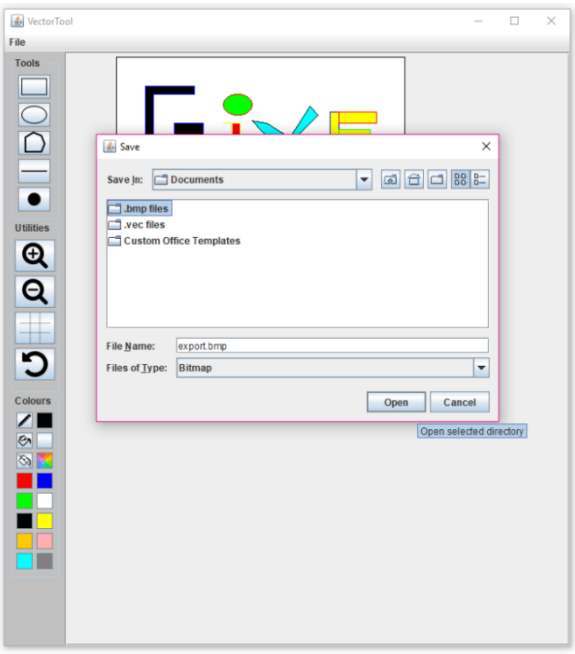
Opened File

Using Export

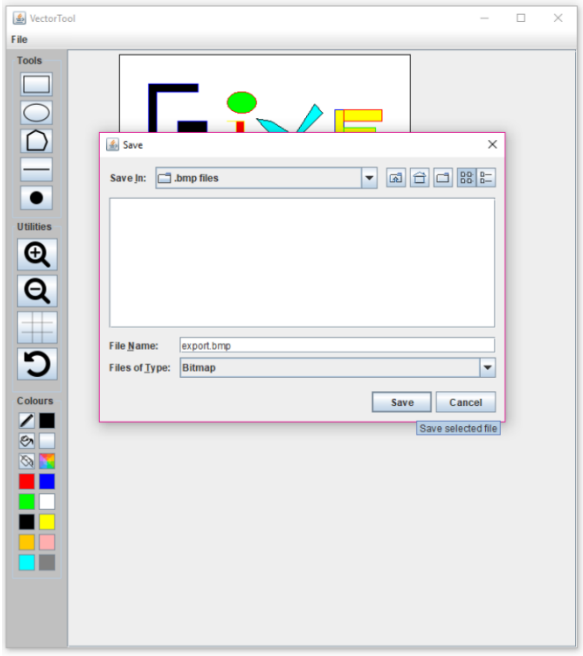
1. Click *Export* in File tab
2. Navigate File System to location to save file
3. Name file and click *Save*
4. Set side width of BMP and click *OK*



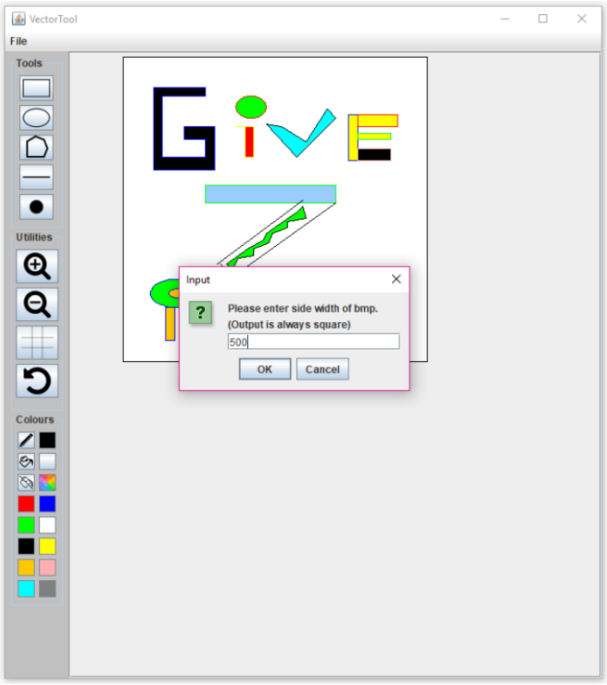
Export button



Folder to save export



Name file and save



Set side width

Using Save

1. Click *Save* in File tab