# Topic 151 – Drawing program

## 1. Personal info

Title: Drawing program
Student name: Tu Tran Le
Student number: 401311
Study program: Service Design and Engineering
Year of enrollment: 2013
Submission date: 13.05.2014

## 2. Overall description

The application is a simple, graphical, mouse driven drawing program. The project targets to fulfill the average difficulty level. It was made based on Tkinter library of Python. Tkinter is a platform framework used for GUI programming. It provides a set of UI elements which can be used to draw the user interface and handle user interactions. Tkinter also provides an UI element called Canvas which can draw certain object shapes and keep track of the created shapes.

The target requirements are at Average level with following requirements:

- Create a new drawing
- Choose color
- Drawing a line with mouse
- Drawing empty circles, ellipses and boxes
- Clearing the drawing
- Undo
- Saving to a file
- Loading from a file
- The drawing canvas has support for scrolling

## 3. User instructions

The program entry point is defined in file “app.py” and it can be started from command line of python. In order to start the application, simplify specify the script path into python executable. For instance:

```
python.exe app.py
```

The user interacts with the program using mouse. The interactions are done using mouse click and mouse move, which is similar to the common built-in Paint program of the OS (e.g. Microsoft Paint). The program also supports saving/loading data using the file system (using bitmap file format). Users can start a new command by clicking on the corresponding button on the GUI. The program can accept following commands:

1. Create new drawing: The whole drawing will be cleared
2. Open existing drawing: Open an existing bitmap picture file and load the image into the drawing canvas.
3. Save the current drawing: Save the drawing canvas into Bitmap image file.
4. Select the line color: Define the shape color to be used to draw the shape.
5. Draw a line: Draw a straight line between two points. First, users click on the drawing canvas to define the starting point. Afterwards, the users can drag the mouse to draw a straight line. The line will be added to the canvas after the users release the mouse.
6. Draw circle: Draw a circle. First, users click on the drawing canvas to define the top-left corner of the circle. Afterwards, users can drag the mouse to define the shape of the circle. The circle will be added to the canvas after the users release the mouse. The mouse location when the users release the mouse will define the bottom-right corner of the circle's bounding rectangle.
7. Draw eclipse: It is similar to the functionality of drawing a circle, however an eclipse is created instead.
8. Draw box (rectangle): It is similar to drawing a circle. However, the bounding rectangle is created instead.
9. Clear the drawing: The whole drawing is cleared.
10. Undo / Redo: Undo/redo the previous task on the command stack.

The program layout is illustrated as follow:

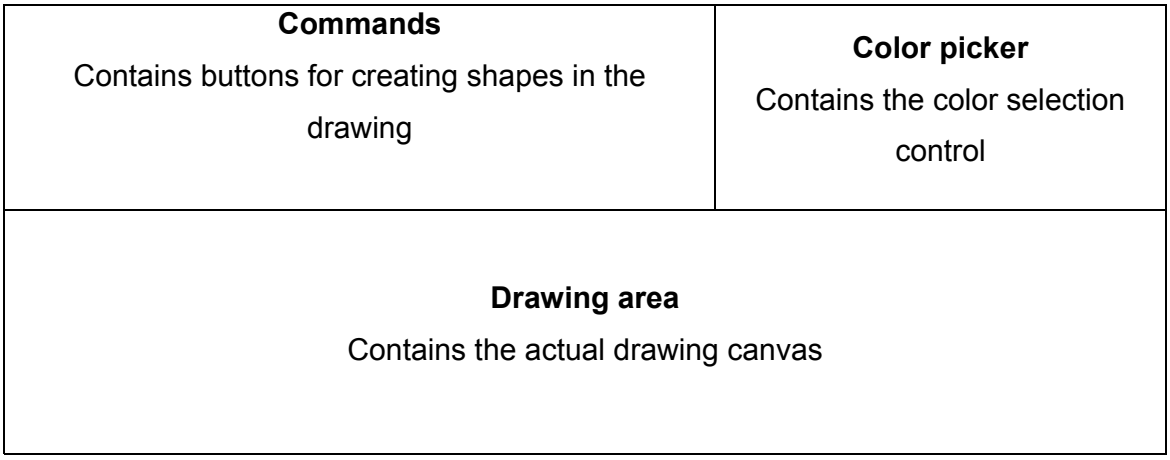


Figure 1: Program layout

### 4. Program Structure

The program will be developed iteratively, thus the class listings may be changed significantly. The following program structure is preliminary and is subject to change as the project proceeds. Basically, the program will have the GUI, thus it can make use of Tkinter library. The main structure will follow the MVC pattern, which consists of 3 main parts:

- Business model: Contains the actual data handling logic. It may contain following subparts:
    - Drawing: Contains objects for handling the drawing operations, including:
        - Drawing canvas class: Responsible for the drawing area on the screen. This class contains data for the actual drawing, including the pixels, metadata and it is associated with a filename. It inherited from the DrawingArea class in Tkinter library
        - Drawing command class: Responsible for certain drawing commands, such as drawing line, circle, rectangle, etc. The command can be activated/deactivated when users select the tool on the screen. They will attach/detach certain mouse event handlers to the drawing canvas. Their main task is to draw the shapes on the canvas and provides support for undoing/redoing. This class will keep reference to an instance of color picker(s) in order to determine the selected line color. For now, it will need

only one color picker since there is no support for drawing background color. This class is an interface/abstract class. The actual commands will inherit this class. For example, “draw line” command will inherit this class. It will override the “activate” method, so that it will draw a line on the drawing canvas when users drag and drop the mouse between two points. Then the command is deactivated, it will remove all the attached mouse handlers. The new active command will behave similarly.

        - Color picker class: Responsible for selecting a color. This class also acts as a custom control which provides color selection for the user
    - IO: Contains objects for handling IO operations, including:
        - File handler class: Responsible for reading/writing bitmap files and interact with the drawing canvas
    - Command stack class: Responsible for keep track of the previously done commands (for Undo/Redo)
- The graphical user interface (GUI) class: Displays the user interface, including toolbars, buttons and drawing canvas. This class makes use of the business model objects in order to handle user interactions. It provides methods for adding commands to the toolbar as well as the color picker.
    - Resource manager class: Contain the icons for all commands. This class is needed in order to support for theming in the future. This class is an interface/abstract class. There will be one default implementation which provides the icons for all commands. It is possible to create a new icon set simply by creating a new inherited class
- Drawing controller class: Responsible for connecting the drawing models and the GUI parts. This class will create an object for each supported drawing command.

So when the program starts, the pseudo code for the program initialization will be as simple as follow:

```
var root = new Tk();
var gui = new GUI(root);
```

```
var controller = new DrawController(gui);
root.mainloop();
```

All supported features will be initialized by the drawing controller as follow:

```
var canvas = new Canvas(picker);
var cmdStack = new CommandStack();
canvas.bindMouseEvents(controllerActions);

var ioCmds = new[] {
       NewDrawCommand,
       OpenDrawCommand,
       SaveDrawCommand,
       UndoDrawCommand,
       RedoDrawCommand
};

var toolCmdTypes = new[] {
       LineCommand,
       CircleCommand,
       EclipseCommand,
       RectangleCommand
};

var gui = new PaintGUI();
gui.SetCanvas(canvas);
gui.AddTools(toolCmds);
gui.AddIOs(ioCmds);

var controller = new Controller();
controller.BindCommands(gui);
gui.Show();
```

## 5. Algorithms

There is not really any algorithm needed in the program. The most complicated parts are to:

- Design the event-driven modules/classes which contain classes and their interaction using the observer pattern, which is already described in section 2. The target is to reduce the dependencies between classes and make the program components as simple as possible
- Implement the binary data handler for the Bitmap file format according to its specification. This can be simplified a lot by using a 3rd party library (if allowed)
- Implement the mechanism for undoing/redoing. One feasible solution would

be to keep track of the affected pixels instead of storing the snapshot of the whole drawing

## 6. Data Structures

The program will only work with bitmap data, and the data can be handled already by the DrawingArea class provider by the graphics library. The array can be used for static variables (such as array of supported commands). A stack is needed in order to implement the command stack (Undo stack).

However, it is necessary to have a data structure which stores specific pixel data, including the pixel location and its color. This data structure is needed in order to implement undo/redo and change certain pixels. It can be simply an array of tuple since it is accessed sequentially and there is no need to modify its data. The tuple contains a pair of pixel location and pixel color.

## 7. Files

The program only works with Bitmap file format. It is also possible to import and export Tkinter drawing canvas to a file. However, it only provides native support for PostScript file format. Thus, it is necessary to convert between PostScript and Bitmap file format, which is why Pillow library was used. In order for the conversion to work, it is necessary to have GhostScript binaries installed and have its location set in the global PATH. A bitmap image file can also be created using any simple image editing software included with Unix distros and Windows.

## 8. Testing

The program was tested so that it can fulfill all the requirements. As it was difficult to automate the testing of user interactions and the screen output, thus the testing was done by actually using the program and partially by unit tests. The basic operations to be tested:

- File manipulation (opening/saving)
- Modify & clear the drawing (involves creating shapes and undoing)

Unit tests were made for most classes in the project. However, complete unit test coverage could not be achieved since there are certain operations which require user interactions (such as specifying file name for opening/saving). Besides, Tkinter drawing canvas did not not support pixel-to-pixel data extraction, which prevented unit testing using pixel-to-pixel comparison. Besides, there are base abstract classes which contain empty methods, which also lowers the unit test coverage percentage.

Manual testing was also done during the project. The target is to verify that all basic features are functioning properly. Some of the preliminary use cases are as follow:

11. Create a new drawing from scratch
12. Open existing drawing for modification
13. Save the current drawing

The drawing could be modified by using different commands in random orders:

14. Select the line color: All new shapes created afterwards should have this line color
15. Draw a shape on blank or existing drawing (line, circle, eclipse, box)
16. Clear the drawing: The drawing canvas should be emptied
17. Undo / Redo: The command stack should work repeat the corresponding commands in a logical manner

The order of testing was to test all common use cases first. Afterwards, the program was tested for unhandled exceptions by trying out erroneous and extreme cases. This followed quite closely to the original plans (except that the workflow has been simplified).

## 9. Known defects and missing features in the program

The program has been tested against the planned feature requirements. However, the features specified in Hard difficulty were not implemented, including drawing text, background, different line shape and direct manipulation of existing shape objects (moving, deleting, copying and cutting). During the testing process, no defect was detected. This may be due to the fact that the existing functionalities are quite simple and straightforward. There were more problems with installation and setting up the needed libraries in order for file opening/saving to work.

## 10. The three best and three worst parts of the code

The code was organized according to MVC model and implemented quite well according to the design pattern. The best parts of the code consists of:

- Abstract base classes which define the interfaces for all model objects: This is not very meaningful in type-free programming language such as Python. However, it would make things easier to develop new features later on (since there is a constraint on methods which need to be implemented)
- Separated UI layer from business layer, which makes the code well-organized: The code is clear and class's responsibilities are isolated. It becomes easier to recognize the functionalities of the classes
- Code is reusable and there is no major repeated code: The amount of code is minimized and it is easier to implement new features by inheriting from other classes

However, there were also certain drawbacks in the code, including:

- Tkinter was used heavily and there was no wrapper around it. Thus, it would be difficult to change to another UI framework later on (especially when Tkinter has become obsolete due to its limited functionalities)
- Some methods take too many parameters. There could be a class which contains all the arguments
- Some objects were referred to in multiple classes. The code could be refactored so that the object will be isolated and there is only 1 reference to an object

## 11. Diversions from the original plan

The project was carried out according to the plan. However, there were certain deviation from the original plan:

- Tkinter library was used instead of GTK+: The main reason is because Tkinter is already built-in with Python and it provides the same set of functionalities for the project. However, there was a drawback with using Tkinter since it does not provide native support for processing bitmap image file. Thus, Pillow library and GhostScript binaries were used instead for handling bitmap image file.
- GUI was simplified: The user interface was simplified compared to the original plan. The main reason was because the set of feature is quite small, thus it would be more reasonable to have a simpler set of UI elements.

Time-wise, the project was carried out with a shorter timeline. The main reasons were that the implemented features and UI were simplified. Moreover, the project could reuse existing libraries (Tkinter), thus the amount of work was quite smaller than anticipated.

The implementation order was done according to the plan and the project flow went very smoothly. This was a key learning point. The chosen implementation order fit very well with the program design pattern (MVC model). The main program can be implemented independently and additional features can be implemented in any order. The features consisted of the supported draw commands, IO commands (clear, open and save drawings).

## 12. Actual order of activities

The project consists of the following phases:

1. Make the planning: Define the requirements and software specifications. Make general and technical.
2. Evaluate the alternative libraries which can serve the purpose of the program.
3. Start the program implementation accordingly in the planned order.
4. Implement unit tests and starts system testing.
5. Makes documentation

The program features implementation was done in a shorter period than the planned

timeline due to the explained reasons in the previous section.

### 13. Your own assessment of the project

The program can satisfy the Average difficulty level of the course. All the basic functionalities have been tested to be fully functional. Thanks to MVC design and clear separation of different application layers, it is quite simple to add new features. However, since all planned features only involved mouse operations (clicking on the canvas, drag and drop), the event handlers were attached to the canvas by the controller in order to simplify the application architecture. This approach would not work for more complicated tasks such as creating text (or moving shape objects). It would be more flexible if the event handlers are attached by the command itself.

**References**


1. tkinter — Python interface to Tcl/Tk: https://docs.python.org/3/library/tkinter.html
2. GUI Programming with Python: Canvas Widget: http://www.python-course.eu/tkinter_canvas.php
3. Course assignment 2 (Undo/Redo Manager)

## Appendices

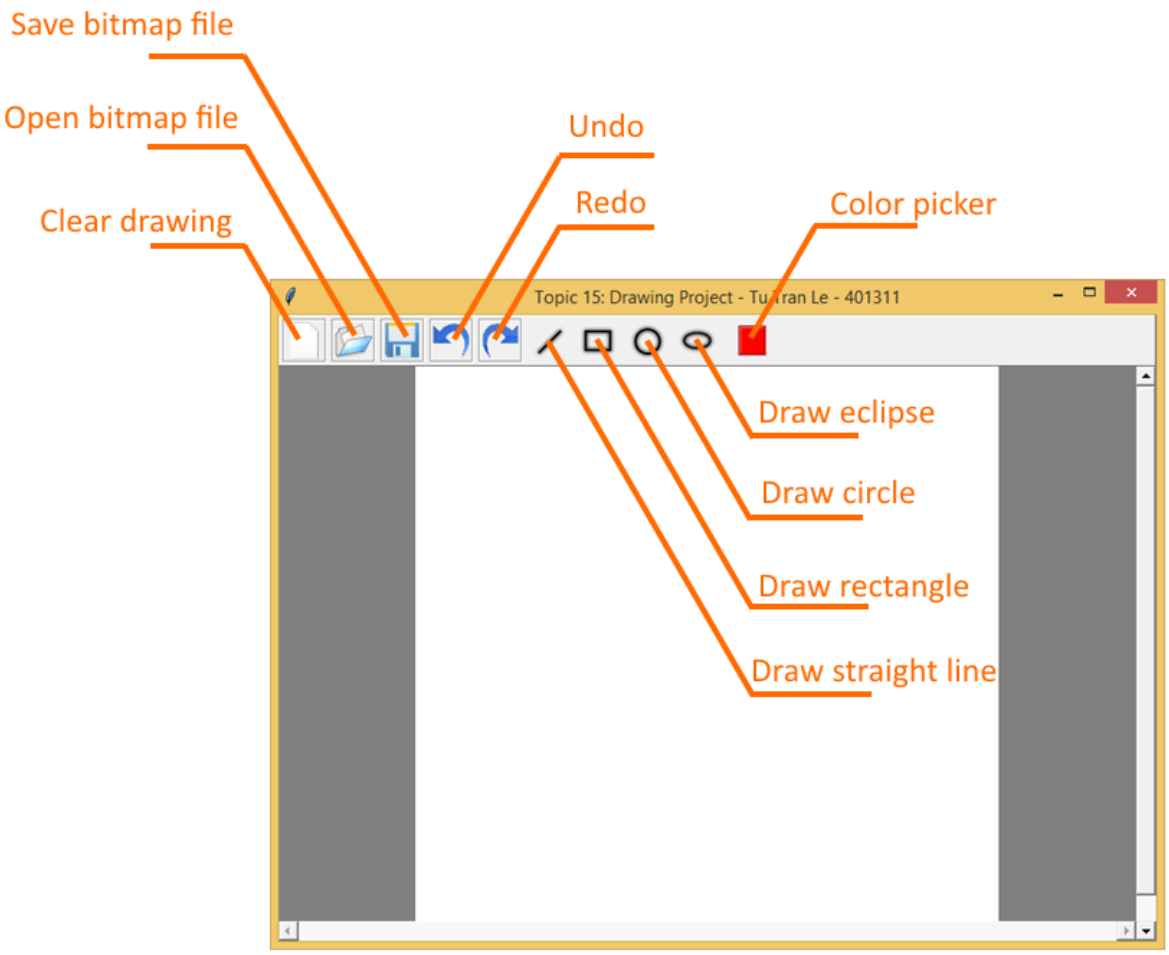


Figure 2. The main application view on startup.

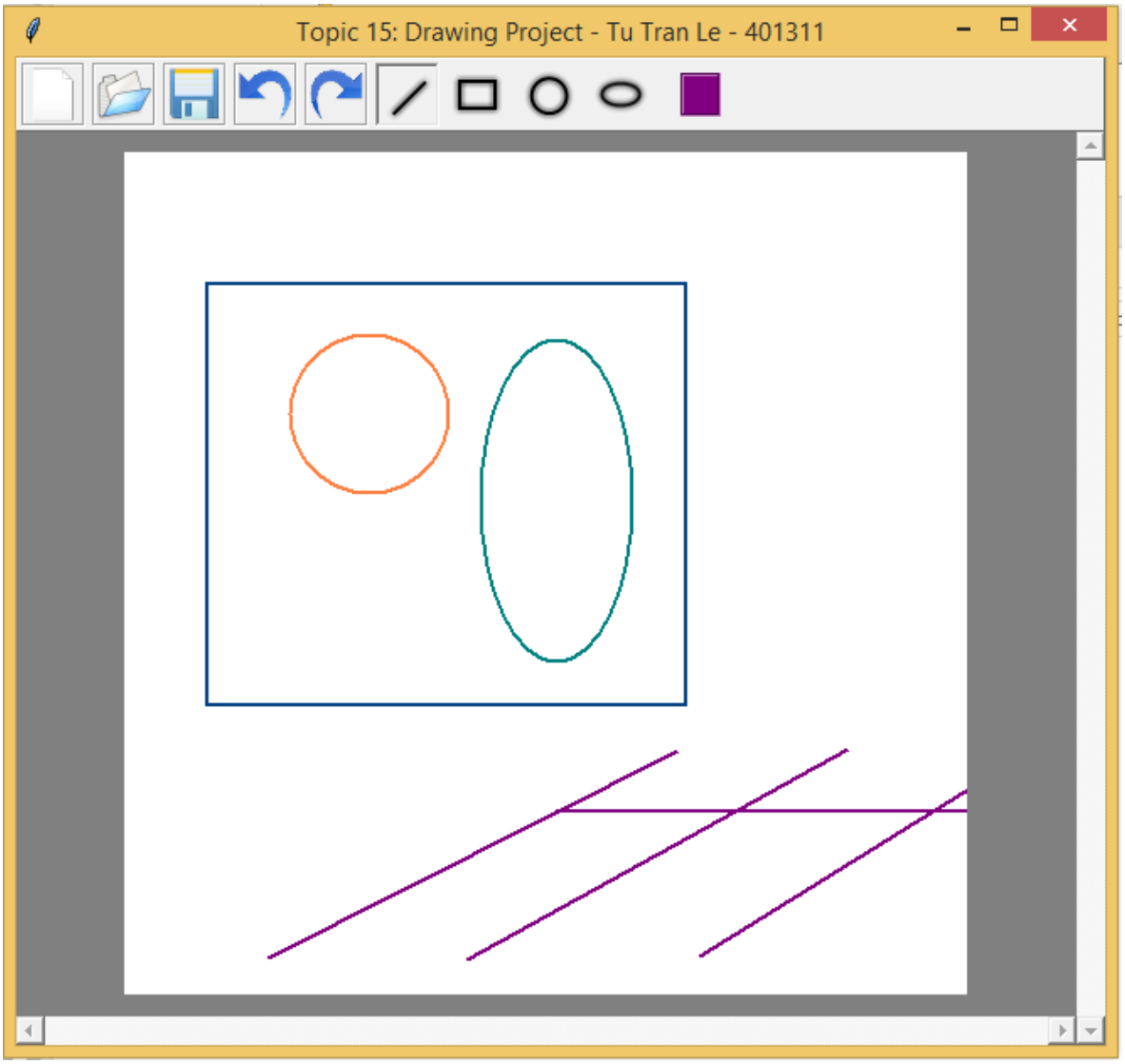


Figure 3. Drawing shapes on the drawing canvas.

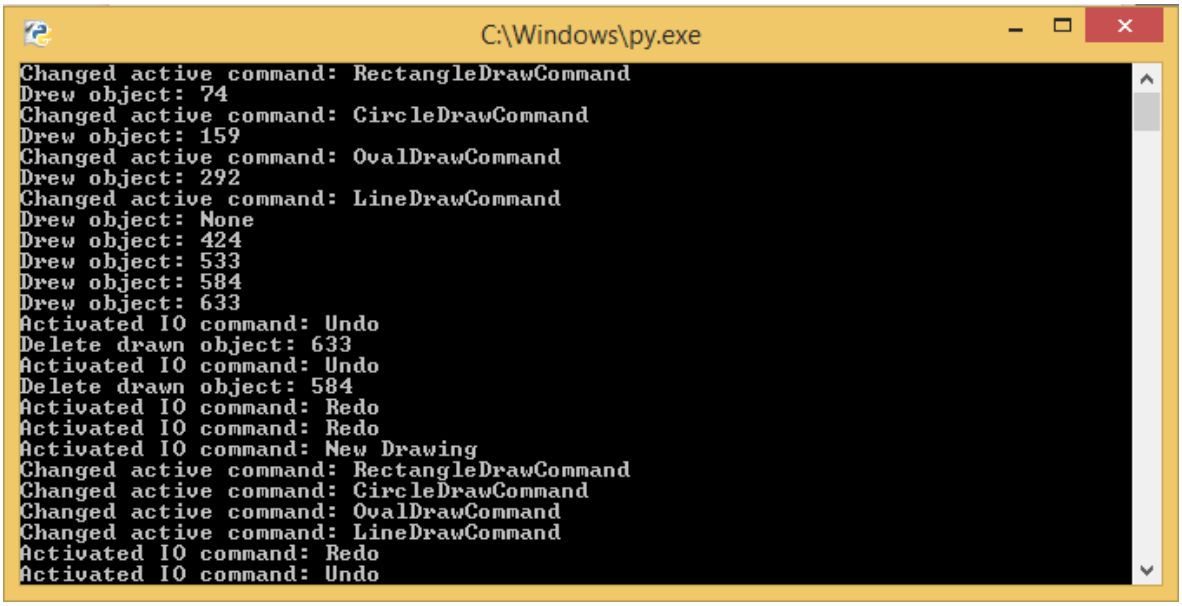


Figure 4. Console view displays the logs.