Technical plan

Topic 151 – Drawing program

Student name: Tu Tran Le

Student number: 401311

Study program: Service Design and Engineering

1. 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 GTK (PyGObject) or 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 will probably inherit from the DrawingArea class in GTK/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.

Some important methods and classes are summarized in the following table:

Class name	**Method**	**Description**
Command	Undo()	Undo the previous command
(Base abstract class for	Redo()	Redo the current command

all command types)	SetActive(newStatus)	Activate or deactivate the command
DrawCommand (Base abstract class for all drawing commands, such as draw line, draw rectangle, draw circle)	_init_(canvas)	Initialize a new instance of the draw command. The constructor also takes the drawing canvas. The command object needs to know the canvas to which the mouse handlers can be attached
This class inherits Command class	GetColor()	Get the selected color
ColorPicker	_init_(cmdTypes)	Add the supported command types into the command stack (command types must inherit from DrawCommand)
CommandStack	SetActiveCommand(cmd)	Set the new active command (e.g. draw line, draw rectangle). This will also deactivate the current active command if any
	Undo()	Undo the last command
	Redo()	Redo the previous command
	Show()	Draw the GUI to the screen
PaintGUI	SetCanvas(canvas)	Set the drawing canvas
	SetStack(cmdStack)	Set the command stack
	AddIOs(commands)	Add the IO commands array into the toolbar (e.g. new file, open file, save file)
	BindCommands(gui)	Initialize all supported commands with the GUI
	Clear()	Clear the whole drawing

DrawingController	Initialize(data)	Initialize the data (e.g. for existing drawing or replacing with the new drawing)
DrawingCanvas	GetData()	Get the current drawing data (e.g. pixels and metadata)
	SetColorPicker(picker)	Set the active color picker used for drawing.
	GetColorPicker()	Get the active color picker used for drawing. This is needed in order to get the selected line color
FileIO	Open(file)	Open a drawing from a Bitmap file and returns the drawing data
	Save(canvas, file)	Save a drawing to a Bitmap file

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

```
var io = new FileIO();
var picker = new ColorPicker();
var canvas = new Canvas(picker);
var cmdStack = new CommandStack();

var ioCmds = new[] {
       {new NewCommand(canvas) },
       {new OpenCommand(canvas, io) },
       {new SaveCommand(canvas, io) } }

var toolCmdTypes = new[] {
       typeof(LineCommand),
       typeof(CircleCommand),
       typeof(EclipseCommand),
       typeof(RectangleCommand),
       typeof(ClearCommand),
       typeof(UndoCommand),
       typeof(RedoCommand) };
```

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

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

2. Use Cases

This section describes the usage scenario when users try to create a new drawing which contains several shapes:

- Users start the application: An empty drawing canvas is shown
  - The program will initialize all needed objects and show the GUI
- Users select the line color using the color picker
  - The color picker data will be updated with the newly selected color
- Users select a drawing tool called “draw line”
  - The active command of the command stack is set to “draw line”
  - The command stack creates a new instance of “draw line” and activate it
  - The command stack adds an event handler to the command “draw line” so that it will be notified when the event is deactivated
  - The “draw line” command will attach mouse event handlers to the drawing canvas. It will observe the mouse click and mouse move events
  - The command stack is updated and keeps track of the new active command
- Users left-click at one location on the drawing canvas which defines the starting point of the line
  - The “draw line” command will be notified. It will then store the location of this first selected point as the start point of the line
- Users move the mouse to another location, the preview line is drawn from the starting point to the current mouse location
  - The “draw line” command will be notified. It will then draw a line preview on the canvas

- Users left-click at another location, the line will be drawn on the canvas
    - The “draw line” command will be notified. It will then draw a line starting from the start point to the current mouse location on the canvas
    - The command will deactivate itself and detach all mouse event handlers from the drawing canvas
    - The command stack will be notified by the event handler. The current command will be added to the Undo stack
    - The command stack will repeat the process of creating and activating the current active command (which is “draw line” type)
- Users save the drawing: A popup window will appear and ask for file name
    - The “save” command will be activated and handle the logic

3. 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

4. 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.

5. Schedule

Date	Task description	Workload
14.04.2014	General planning	4h
15.04.2014-18.04.2014	Technical planning	12h
20.04.2014-21.04.2014	Find out how GUI programming works with Python	6h
22.04.2014	Plan demo	0.5h
20.04.2014-24.04.2014	Implement base classes/interfaces and program skeleton	24h
25.04.2014-27.04.2014	Implement GUI layout and integrate with the skeleton. Implement Clear drawing command (the simplest command) to test the architecture	16h
28.04.2014-02.05.2014	Implement Line drawing command	14h
03.05-2014-05.05.2014	Implement Rectangle drawing command	12h
07.05-2014-09.05.2014	Implement Circle drawing command	10h
10.05-2014-12.05.2014	Implement Eclipse drawing command	8h
15.05.2014-17.05.2014	Test the whole program and bug fixing	16h
17.05.2014-19.05.2014	Documentation	14h
26.05.2014-30.05.2014	Final demo	1h

The class will be implement in the following orders:

- Abstract class/interface
- Higher-level components: GUI layout

- Lower-level components: Command stack
- Major GUI components: Drawing canvas
- Major feature components: Commands
    - Draw line
    - Draw rectangle
    - Draw circle
    - Draw eclipse
- Independent components: Color picker

6. Unit testing plan

Only major parts of the program will be unit-tested due to the time constraint. It is also very difficult to test the actual user interaction and the displayed output. The following components will be unit-tested:
-GUI initialization: Validate that the components are added to the GUI
-Command stack: Validate that the command stack behaves correctly when commands are added/undone
-Drawing commands: Validate that the shape is created correctly in the drawing. This consists of pixel location and color
-Undo commands: Validate that the drawing canvas will be restored to the state before the command is executed. This applies to all commands

It is better to have full unit test coverage, however this is not feasible in this project.

7. References

Python documentation: https://docs.python.org/3/
Tkinter: https://wiki.python.org/moin/TkInter
GTK+: http://www.pygtk.org/
painthon: https://code.google.com/p/painthon/
Google (and stackoverflow.com)