Computational Art Graphical User Interface

Long Ho

July 27, 2009

1 Abstract

The usefulness of Graphical User Interface (GUI) has been widely recognized in various softwares, both open-source and commercial. However, despite its very unique and client-specific nature, GUI nowadays seems to share some common patterns that was developed and tested in well-known pieces of softwares. Some of these patterns apply to all groups of users and some very specific. The purpose of this paper is to study the basic principals of designing GUI demonstrated through an application for the Computational Art project. The type of client to be studied is art students with little knowledge of and exposure to computer science. The GUI aims to explain very basic concepts of recursion, variables and methods through a different set of user-friendly vocabularies, icons, images and navigations.

2 Introduction

2.1 Computational Art Project

The Computational Art Project was initiated during summer 2008 at Lafayette College. It has always been intended to be an interdisciplinary project to bring Computer Science students and Art students together. The project itself comprises of several different components: the Metaphor system, the Contexts and the Grammars.

Metaphor: a system of parsing text-based input files into drawing methods. The generated string, or output, from this system is a list of available methods along with their corresponding parameters. The set of methods are pre-defined in the Contexts source code while the input files are called human-readable text called Grammars. This system, though functions well on its own for technical users, lacks input-editing and non-technical documentations. Thus, it is very likely for non-technical users to make input errors, which discourages them from using the system.

Context: a set of pre-defined methods written in Python language. It uses an external program called Persistence Of Vision Raytracer (a.k.a POV-ray) to generate 3D images and Python Image Library (a.k.a PIL) to generate 2D images. Its main functionality is supply users with simple and straightforward methods to manipulate and draw picture, instead of letting them interacting with POV-ray or PIL by themselves. Similar to Metaphor, the system well serves the purpose of generating virtual graphics for Computer Science students, yet lacks a variety of supports for Art students.

Grammars: a text-based input in a separate non-traditional format. A sample would be as follow:

```
grammar Sierpinski
        Axiom A

        Production A => B right A right B
        Production B => A left B left A

        Map A => forward[1]
        Map B => forward[1]
        Map right => right[60]
        Map left => left[60]
```

The grammar is given a name, in this case Sierpenski, for easy identification. Its main purpose is to translate recursive evolution into different drawing techniques. Thus, we use a Variable system with each Variable having its own Production and Mapping rules. In this case, `A` is called an Axiom, or a Variable that gets called first before anything. A is mapped to drawing `forward` 1 pixel, same with B while `right` is mapped to drawing `right` 60 pixels and `left` is mapped to drawing `left` 60 pixels.
Aside from the mapping, the evolution rules convert `A` into `B right A right B` and `B` into `A left B left A`, producing the following pattern: Generation 1: A Generation 2: B right A right B Generation 3: (A left B left A) right (B right A right B) right (A left B left A) Its complexity, though regarded as a basic concept for Computer Science, turns out to

be of great difficulty to Art students. Therefore, there is definitely a need for an intuitive Graphical User Interface.

2.2 Computational Art GUI

GUI itself is very user-specific despite its common back-end languages and engines. It is a combination of computer science and art in its own being. Most GUIs aim to guide non-technical users to take control of and navigate through complicated pieces of softwares.
The history of GUIs has bring forth great resources of design inspiration and technical solutions for nowadays GUIs. In addition, the design choices of well-known GUIs have somehow affect the way people interact with softwares in general. People tend to relate to a common course of action if they see a common set of icons, such as the 💾 for "Save" or a ❓ for "Help" or "What's this".
Therefore, such expectations have build up a common ground for non-technical users and GUI designers to easily interact with each other and this case study software was also built on such basis.
The same principles were the very same foundation for building this specific GUI. Client habits and software usage were analyzed in order to discover working patterns most intuitive to Art major students.

3 Front-end Design

3.1 Requirements Analysis

The back-end engine of the GUI, *metaphor*, a system of mapping patterns to drawing methods. *Metaphor* uses the *grammar*, which is a text input of how patterns evolve over a certain number of generations. The evolution is basically a recursion of different variables, each of which is mapped to one or multiple methods. One example would be the Lindenmeyer system in a more complicated 3D world with custom light source and camera positions. However, the *metaphor* system itself doesn't come with predefined methods and patterns, but rather translate *grammar* into a string of methods, which will be drawn using different *contexts*, such as *diffusion-limited aggregation* techniques.
In addition, the *grammar* file itself is fairly technical and prone to user errors. Every time a user wants to add map a variable to a method, he/she has to directly look into the source file of the context. Thus, the GUI serves the

purpose of layering on top of such system a user-friendly interactive experience along with user-editing and feedbacks.
As indicated before, the target users would be non-technical art major students. However, the nature of art major familiarize such users with a variety of image editing softwares, most well-known being Adobe Photoshop and Adobe Illustrator. Therefore, using a similar layout and design to such softwares would result in an easier transition and more intuitive user experience.

3.2 GUI Layout

Other than the basic I/O components such as New/Save/Load, image editing softwares tend to share a common layout as **Figure 1**
In the center of attention, where users interact the most would be the

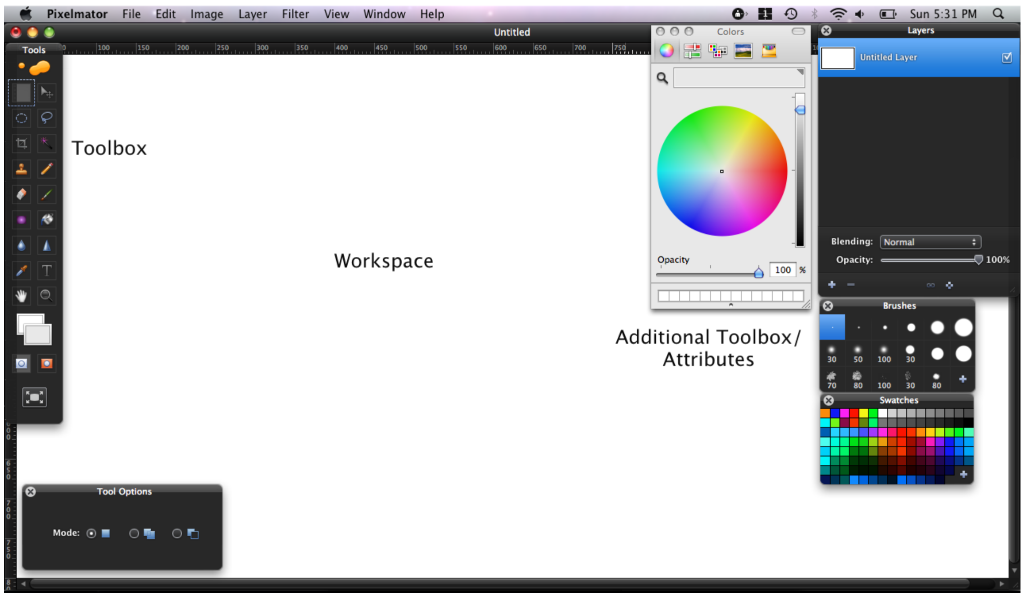


Figure 1: GUI Layout - Sample from Pixelmator

workspace, typically being a blank working area with separate color tone from other components. The toolbox always holds necessary tools that can be access anytime. Thus, it is normally separated from the workspace and have a background color that stands out. The basic I/O functionalities are placed on top just as the usual GUI conventions. With that said, this software, named Computational Art Program (**CAP**), applies the very same principle in its layout, as indicated **Figure 2**.
However, the main difference is that the toolbox is blank, acting as a placeholder for *variables*. It also indicates the freedom upon users being

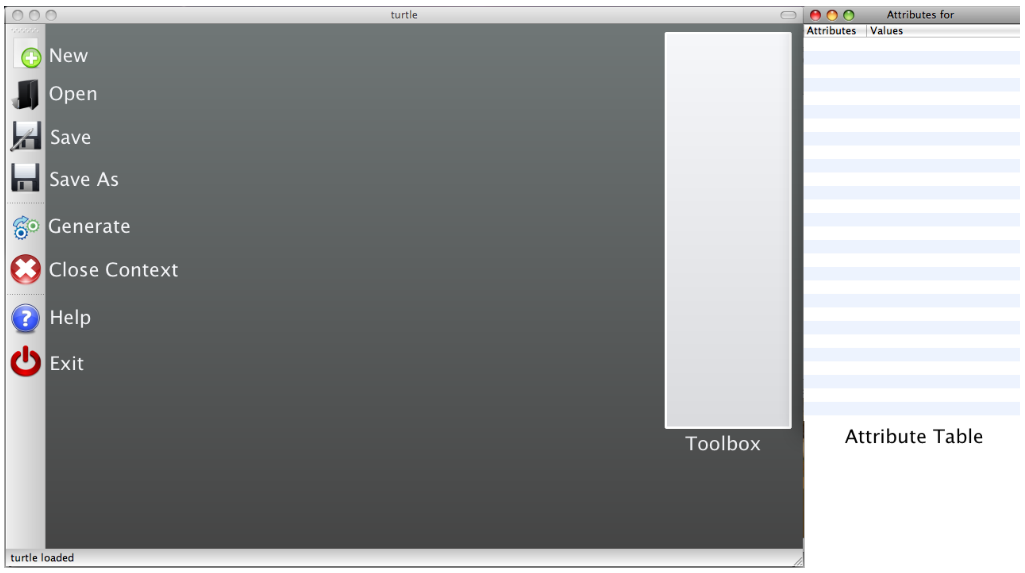


Figure 2: Computational Art Program GUI Layout

able to create their own tools. The toolbox comes with a context-menu (**Figure 3**) allowing users to add/remove and manipulate variables in their own ways. The color choice also follows the same conventions. Darker color for workspace and lighter colors for toolbox for attention.

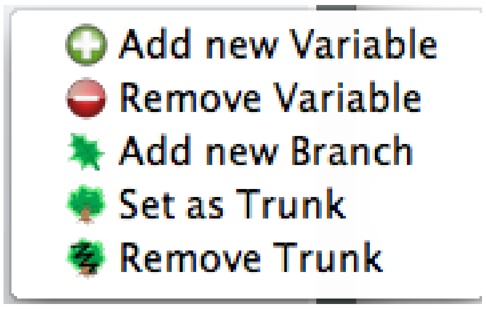


Figure 3: Toolbox Context Menu

3.3 Metaphor Functionality Integration

CAP was brainstormed using the simplest example of metaphor grammar, such as the KochCurve:

```
grammar KochCurve
        Axiom F
```

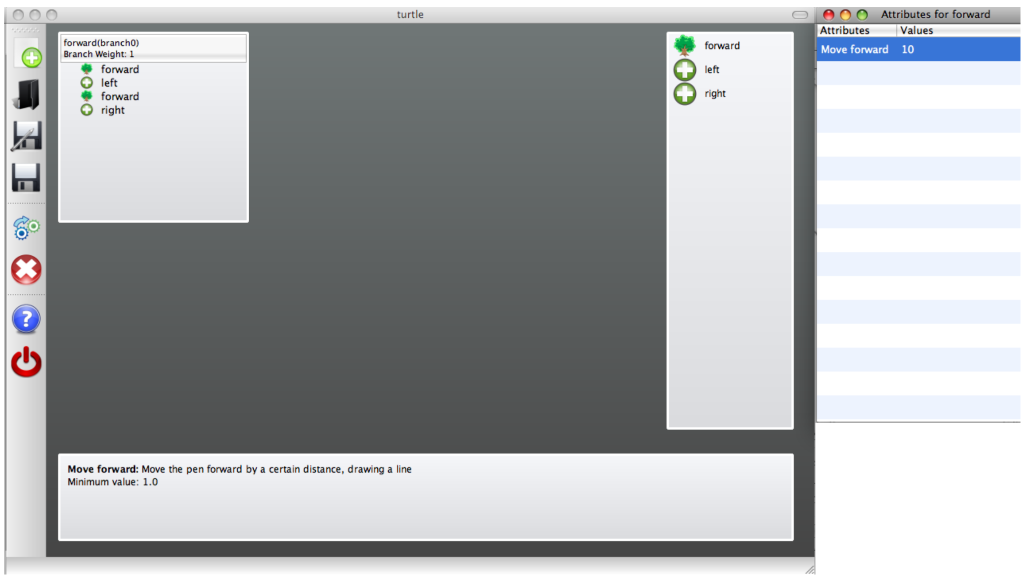
```
Production F => F left F right F right F left F

Map F => forward[10]
Map left => left[90]
Map right => right[90]
```

The structure of this Production is recursive and very similar to a tree structure: after one generation, the starting point (named *Axiom* in this grammar and *Trunk* in the GUI) `F` is converted into `F left F right F right F left F` and the recursions keeps going on. Besides, `F` ,`left` and `right` are mapped to `forward`, `left` and `right` methods respectively. Therefore, these variables are the center of attention in this software, thus deserve a toolbox.
On the other hand, the Production manipulates how the image looks like, which fits the role of a workspace. Each Production has its own small window, named *Branch* in the UI, indicating one option of evolving.
Each mapped method, such as `left` or `forward` is called *Attribute* and has its own place in the attribute table. The attribute table is also integrated with input editing for specific value types such as integer, float and string. Each input box is enforced with predefined minimum and maximum values, as well as available decimal points and steps.
This layout puts the Variables/Trunks in the center as it should be while having Branches on the left and Attributes on the right as **Figure 4**
Another necessary feature would be "Help" in various forms. CAP uses a wide variety of helps including a separate help window, a small pop-up window with brief method description (**Figure 4**) and also tooltips.

4 Back-end Design

4.1 Software Specifications

CAP GUI was written in Python using Python SDK 2.6 along with PyQt4 library and Qt 4.5.2. Python is very flexible, lightweight and straightforward, which turns out to be a great language for the project. Qt has been well-known in the industry for a while with its detailed documentations and a great set of widgets. Most of the GUI components were an extension of Qt Widgets, giving them the ability to take full use of Qt features while developing their own custom behaviors such and re-ordering widgets and custom signals and slots.
CAP also follows a traditional Model - View - Controller design with the Model being in charge of input editing and validation, the View manipulates



forward(branch0)
Branch Weight: 1

Attributes	Values
Move forward	10

Move forward: Move the pen forward by a certain distance, drawing a line
Minimum value: 1.0

Figure 4: KochCurve represented in the UI

widgets' positions and formatting data and the Controller is in charge of signals, slots and data transmission between View and Model.
The I/O format of CAP is designed to be human-readable and editable. CAP uses **XML** to read methods and rules from the metaphor system along with writing savefiles. This design choice is not only because of XML's popularity, but also to take full use of the Python library `xml.sax`, which reduces a great amount of time for writing a custom parser for a custom format. A sample descriptor of "turtle" context would be:

```
<context name="turtle">

<method name="right" canonical="Move right">
<param type="integer" name="amount" canonical="Distance" min="1"
description="">
</param>
<description>Move the pen to the right by a certain distance, drawing
a line</description>
</method>
...
</context>
```

In addition, most CAP documentations are made in HTML, separate from the system source code, providing CAP with the ability to publish its online tutorials and easier to update or make changes.

4.2 Class Design

The simple version of the class design is indicated as **Figure 5**.

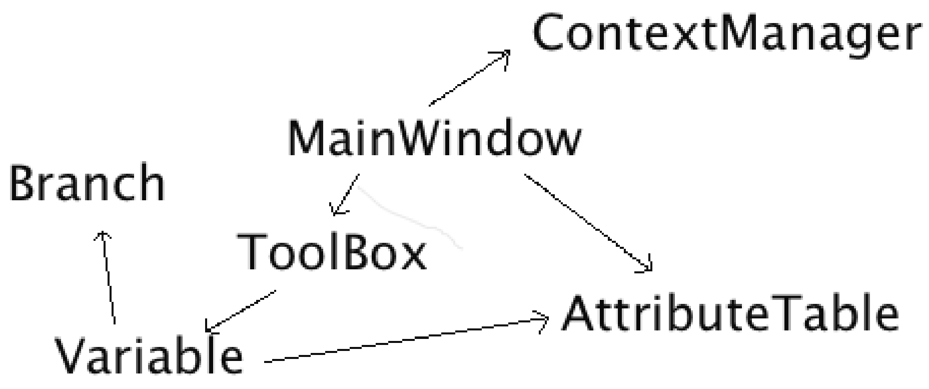


Figure 5: Simplified Version of Class Design

ContextManager is in charge of interacting with the Metaphor system and parse each context's descriptor, producing a Method Database. This database is then passed down to MainWindow and displayed as available attributes in the AttributeTable. The Toolbox is a list of user-created variables. Each variable stores its own branches and methods, which are displayed in Branch window and AttributeTable window respectively.
Besides, in order to make this program user customizable, the configurations are stored separately as static variables. They are written in `compart.cfg` which is parsed every time the program starts. Python also provides CAP with a module called ConfigParser, which gives CAP the ability to be configured as administrators see fit. A sample of `compart.cfg` would be:

```
[folders]
contexts = ../contexts/
docs = docs/
samples = samples/
metaphor = ../metaphor/

[params]
```

```
program_name = Computational Art
recent_file_list = recent.txt
recent_file_limit = 8
float_step = 0.001
```

5 Conclusions

Building a user-friendly GUI is a challenging process of combining aesthetic inspirations and software engineering skills. The process of making CAP involves not only following a set of pre-defined patterns but also brainstorming new ideas as well as interpreting requirements in a user-specific way. Although CAP was built specifically for art students, it has been designed for flexibility and maintainability across different platforms. It has also brought forth a new experience to art design from a perspective of a Computer Science student.

6 References

Dayton, Kramer, McFarland, Heidelberg. "Participatory GUI design from task models". Conference on Human Factors in Computing Systems. New York, NY, 1996.
"Designing Interfaces". 1st Edition. O'Reilly, 2005.
Sade, Battarbee. "Bridge for buttons - a GUI design methodology applied in non-GUI consumer product design". Designing Interactive Systems. New York, NY, 2000.
Johnson. "Joining the GUI design team: a case study". ACM Special Interest Group for Design of Communication. Banff, Alberta, Canada, 1994.
Wilding. "Practical GUI screen design: making it usable". Conference on Human Factors in Computing Systems. Los Angeles, LA, 1998.