Object Oriented Programming with Java

Coursework 6

Graphics

Nikolaos Pappas

Profesor:
Dr Ian Holyer

May 11, 2016

Introduction

I found this project to be an opportunity to dig deep into the Java built it Graphics library. My idea was to combine this project with the research I am currently conducting for the Final Project in this course. The research is about Artificial Neural Networks and Pattern Recognition in imagery. To be more specific I have undertaken the task of building a Convolutional Neural Network, thus it is crucial for me to research on Image Processing before even I can get started with the ANNs.

In this project and up to this point I have used simple drawing on a JPanel, Polygon drawing, Java Image manipulation, mouseListened transformations on drawnShapes, JButtons, JComboBoxes, JSliders, Animation Timer, live video manipulation with OpenCV, and the slightly out of context sound Manager.

Software Engineering and Class Dependencies

All slides extend Slide class All slides are animation ready with methods like *tick()*, *start()* and *stop()*
The animation is acheived by a runer class with the swing timer swing utilities. the Slide sequence is independent of the gui and can be played without gui. There was no testing framework involved as everything could be put into trial with a simple glance.

Slide

Slide is a class where all the common attributes and functionality of the different scenes are accumulated. All classes use a common pallete and typography so there one can find the colours and fonts used and public methods so that any Class in this program can obtain them.

Intro

This is just an introductory slide for the user to get orientated on what is about to follow.

Light

This slide's aim is to introduce the user the concept of the picture. In a monitor for example we begin with a form of electric energy that creates white light and then by filtering it in various ways we get the colour we want. If one rotates the prism the light from the flashlight gets divided into the spectrum of colours. To achieve this I used the AffineTransform class.

ColourCombination

This slide simply explains how combining different colous of a base of colours, in this case the base of red, green, blue, one can get all the colour space. In this slide we see the use of JSliders

ChannelCombination

After we see how colours are made then the next step is to compose a picture out of pixels, that are a combination of the above three channels. A picture is made out of rectangle with the superposition of the colour channels. This functionality was added with three arrays of JComboBoxes.

Video Processing

This slide shows how video can be processed like an image, as a video is compiled from individual pictures, called frames. The user can use JButtons to chose between the given effects. The image processing has been achieved by the class VideoProcessing, which is a combination of code taken from the internet and methods developed by me, inspired from the sources cited.

Pong

This slide is the final one, and the one that shows the user, the added value of all this. This is just a Pong game, or is it? It is not, because it is controlled by the position of the face detected by the camera. Computer Vision is not all about processing images to get artwork out of them, usefull data can be obtained from the images as well. In this case it is not the most usefull application in the world, it lets the user though make that leap, and think further, that the paddle movement actually comes from the face detection.

Explain

Explain is the class that controls all the Slides. It is the frame that is projected in the GUI class. As the original idea was to create an interactive animation, initially the Slides would play until their end, and at the moment of their completion the next scene in the sequence would be triggered. This type of sequence resembles a video animation and it was hard to calculate how much time a user needs for each scene. To tackle this issue a GUI was added so that the user can go anywhere he wants at any time.

GUI

The GUI class is the one that adds the browsing feature in between the scenes, so that the user can chose what to do, in any sequence

Features

In this section some of the most important Java Graphics and general features used to develop this project.

SVG Shape Input

This application has the ability to import closed polygons directly from SVG files, exported by Inkscape, as JavaFX files with the suffix ".fx". This idea came from the need to draw a polygon which was easier to draw in a Vector graphics with GUI, rather than calculating coordinates by hand. The SVGShapes class reads the ".fx" file and keeps only the coordinates of the Polygon, that is why at the moment the functionality is reduced to closed polygon shapes. I thought it was a usefull adittion, and one that could, in future projects, enable colaboration with people without programming background. Can be easily used with the PolygonExt class which is an enhanced functionality etension of the java Polygon class.

Sound Manager

This aproach to the sound effects addition was probably an overkill, as it is used only for a couple of sound effects, but in my opinion these small sounds add a first impresion of a vivid interactive application. The sound manager class is implemented as found on the cited inside the file web site, with no change at all. Appart from the few sound effects, the real addition of the sound manager is the lesson one gets when they try to read, understand and include someone else's code into their own application.

Jar Ready

I have since the first Java assignement tried to program, having in mind that the resulting application should be compatible and ready to be packed in an executable jar file. This was a way to get the most out of a multi-platform programming language with enhanced compatibility, which enables the programming of platform-independent application with a few simple steps. In this case I wasn't completely succesful in this task, as the use of the openCV library, a computer vision library written in C++, impeded its inclusion to the jar file because of its size. The inclusion of the library inside the jar file would result in an executable larger than 1Gb.

Custom JComponent Style

The Java built0in libraries provide the programmer with modules, such as JComponents, so that a modulaTr aproach to the design is possible. This is a great way of designing, there is though a small drawback, these components still have the look and feel of the previous millenium and make the application look like something created in the past.

Design

This application has educative aims so the graphics design approach was orientated towards clean neutral surfaces. This design aims to be eligible for teaching in class or in other educative platforms. As one might observe in such cases the most commonly used medium is a tablet. The big buttons have been design that way exactly because their are intended to be used with touch-screens that is friendly for children. Neutral grey colored pallete is chosen because of the nature of the slides that explain colour and their edge cases in black or white would seem too hard and might get devalued with other combinations of colours.

Frameworks, Compiling and Running

I started developing this project with a simple text editor and raw javac, as I did always before with java applications. When I saw that this would become a complex program and I would need some framework to keep everything tidy I started using gradle. I was happy with gradle until I found it impossible to compile including the opencv libraries. This problem made me turn to ant. Ant was of an extream help even though I had to restructure the folder for it to find the correct resources path, I found ant remarkable.

OpenCV compilation tutorial

1. Download opencv 3.1.0 source code for Mac/Linux from:
   https://github.com/Itseez/opencv/archive/3.1.0.zip

2. Decompress the the zip file to a folder of choice
   (Not necessarily the system folder)

3. Configure the opencv code with shared libraries turned off Using cmake
   with the commands below:
   mkdir build
   cd build
   cmake -DBUILD_SHARED_LIBS=OFF ..
   Make sure that ant and java dependencies are found.
   e.g *"– Java:*
   – ant: /bin/ant (ver 1.9.2)
   *– JNI: /usr/lib/jvm/java/include /usr/lib/jvm/**java**/include/linux /usr/lib/jvm/java/include*
   – Java wrappers: YES
   – Java tests: YES"
   And that the java libraries are about to be compiled:
   "OpenCV modules:
   *– To be built: core flann imgproc ml photo video imgcodecs shape videoio highgui objdetect superres ts features2d calib3d **java** stitching videostab python2"*

4. Compile the code bu running:
   make -j8
   This will create the files needed for the compilation of the project in:
   (ocvJarDir)folderWhereOpenvWasExtracted/build/bin/opencv-310.jar and
   (ocvLibDir) folderWhereOpenvWasExtracted/build/lib

5. Compile java project with:
   ant -DocvJarDir=/home/msc15/np15685/linux/Downloads/SOFTWARe/opencv-3.1.0/build/bin/ -DocvLibDir=/home/msc15/np15685/linux/Downloads/SOFTWARe/opencv-3.1.0/build/lib

   ** Replace the italics with the build folder you created at the start of this tutorial.