2015 ICTSA Programming Challenge

COOLLAGE

Version 0.999

Dr Christian Colombo, Dr Jean-Paul Ebejer,
Prof. Gordon J. Pace, Dr Chris Porter

November 20, 2015

This document outlines the programming challenge task to be solved in the 2015 ICTSA Programming Challenge (sometimes referred to as *Game of Codes*). The competition opens on Friday, 20th November 2015 at 20:00 and closes on Sunday, 22nd November 2015 at 20:00 ZST[1]. This document also specifies the criteria that will be used to judge the competition's entries.

1 Background

2029 A.D, the world is now a dysfunctional dystopia worth of a Cormac McCarthy novel or a Stanley Kubric film. Computer and AI overlords have taken over, fulfilling the 2015 prophecy of Elon Musk and Stephen Hawking[2]. All hope is lost, faith in human capability is at a historic low, and individual resolution broken. Machines have targeted and successfully obliterated all forms of human expression, including art and facebook statuses. Monets, Picassos, Dalis, Pollocks have been sequestered by robots and incinerated in a frenzied mechanical dance.

New earth is bleak, but to ease the pain induced by our memories of old earth (and perhaps in accordance with the First Law of Robotics), the Machine has made sure that humankind does not remember any of that. Except that you are the only one left on earth who does. You cannot understand

[1] Zeus Standard Time – the time as set on UNIX server zeus, based at the Department of Computer Science at the University of Malta. This time can be seen on the Programming Challenge website at `http://www.cs.um.edu.mt/svrg/56ayl4778abtds377/Game_of_Codes/`

[2] Amongst others. `http://futureoflife.org/AI/open_letter_autonomous_weapons`

why this is so, but can assume it is due to a subtle bug in the *Human Mind Conditioning and Memory Management* module in the Machine.

You are the only human left alive who can remember scenes from old Earth, and the only one who can remember the art works of a previous time. With every passing day you remember less and less, but you hit on a solution. You write a program to process your thoughts and rebuild images from your memory using bits and pieces of images from bleak new Earth. By giving hope to the people you plan to overthrow the Machine. Starting now ...

Good luck *programmer*!, or should I say *revolutionary*?

2 The Task

The task is to write a program which given a target image, a list of composing images and the number of rectangles (n) to use, will copy&paste (at most) n rectangles from the composing images to approximate the original, target, image. We refer to the final set of all rectangular pastes as the *collage*.

For example, if you are given *La Gioconda* as a target image, and the list of composing images contains an image of deep space, you can use parts of this to approximate the dark side of *La Gioconda*.

2.1 Input

Your program executable is to be called `the_wise_man_of_babylon`[3]. We will execute `the_wise_man_of_babylon`, which is to take the following parameters as input (in this order):

1. **`target_file -`** The path to the target image, in `.cjp` format. This format is described in Section 2.1.1.
2. **`composition_images -`** A path to the text file (typically named `composition_images.txt`) containing the **absolute** filenames of the composing images from which rectangles will be copied to form the collage. Each of these composing image files is also in `.cjp` format. Only one `.cjp` image per line is allowed in the `composition_images.txt` file.
3. **`rectangle_count -`** The maximum number (integer) of rectangles allowed for pasting.
4. **`collage_output_file -`** The path to the output file, in `.clg` format which contains the list of regions to paste. This format is described in Section 2.3.

Your program will be called from a batch file you will also supply. This file will be named either `the_wise_man_of_babylon.bat` or

[3]There is method to our madness

`the_wise_man_of_babylon.sh` file depending on your OS. In this file you should make sure to set any extra parameters you may need (e.g. setting `Xmx` or `Xms` sizes for the Java virtual machine).

2.1.1 `.cjp` Image File Format

Lines in a `.cjp` image file are defined as follows:

1. An integer (w), representing the width (number of pixels) of the image followed by a new line.
2. An integer (h), representing the height (number of pixels) of the image followed by a new line.
3. h lines each of which containing:
   a) A space delimited string of w colours (one for each pixel), where each colour is defined as a six character hexadecimal color code (two each for the red, green, blue components, in that order). This line ends with a newline character.

An example of a red plus sign (+) on a white background is shown in Listing 1.

Listing 1: Anatomy of a `.cjp` file.

```
8
9
ffffff ffffff ffffff ff0000 ff0000 ffffff ffffff ffffff
ffffff ffffff ffffff ff0000 ff0000 ffffff ffffff ffffff
ffffff ffffff ffffff ff0000 ff0000 ffffff ffffff ffffff
ff0000 ff0000 ff0000 ff0000 ff0000 ff0000 ff0000 ff0000
ff0000 ff0000 ff0000 ff0000 ff0000 ff0000 ff0000 ff0000
ff0000 ff0000 ff0000 ff0000 ff0000 ff0000 ff0000 ff0000
ffffff ffffff ffffff ff0000 ff0000 ffffff ffffff ffffff
ffffff ffffff ffffff ff0000 ff0000 ffffff ffffff ffffff
ffffff ffffff ffffff ff0000 ff0000 ffffff ffffff ffffff
```

2.2 Output

The output file (described earlier as `collage_output_file`) is a text file with `.clg` extension which format is described in detail below.

2.3 `.clg` Collage Output File

The collage file contains a list of lines, each of which having the following format:

composing_image_id,x_{comp},y_{comp},width,height,x_{target},y_{target}<newline>

where:

1. **composing_image_id -** The integer identifier which points to a line in the input `composition_file`. This is a zero-based index.
2. $\mathbf{x_{comp}}$ **-** The zero-based x coordinate of the top-left corner of the copy&paste rectangle on the **composing** image.
3. $\mathbf{y_{comp}}$ **-** The zero-based y coordinate of the top-left corner of the copy&paste rectangle on the **composing** image.
4. **width -** The width (in number of pixels) of the copied rectangle on the **composing** image.
5. **height -** The height (in number of pixels) of the copied rectangle on the **composing** image.
6. $\mathbf{x_{target}}$ **-** The zero-based x coordinate of the top-left corner on the **target** image where the rectangle should be pasted.
7. $\mathbf{y_{target}}$ **-** The zero-based y coordinate of the top-left corner on the **target** image where the rectangle should be pasted.

An example of a `.clg` file is show in Listing 2.

Listing 2: Anatomy of a `.clg` file.

```
1,105,15,60,95,20,30
6,11,110,53,54,15,132
6,100,10,60,94,81,15
4,20,3,30,6,1,76
8,75,15,60,80,22,40
```

Note that throughout this text the origin (0,0) of an image is at the top-left corner of the image.

Only rectangular copy&pastes are allowed. The order of rectangles in the `collage_output_file` is important as it defines the order in which pasting will occur. Rectangles in the `collage_output_file` may be overlapping, but the current rectangle in the output file always covers any previous rectangle

pastes. Portions of rectangles exceeding the border of the target image are ignored. Also, portions of rectangles exceeding the border of the composing image are ignored. Note that the number of lines in the `.clg` file **cannot** exceed the input parameter `rectangle_count` (described in Section 2.1). If `rectangle_count` is exceeded, your run is disqualified for that particular judging problem. You can copy rectangles from any number of composing images (i.e. you can use a single composing image more than once, or not at all). You can also copy from the same area in a composition image multiple times. The list of composition images used by the judges during evaluation is secret, but will be published when the competition is over.

2.4 Scoring Function

A black background with the size of the current target image is assumed (i.e. there are no "holes" in the final collage). The `collage_output_file` is read line-by-line and the rectangles are copied from the specified composition image and pasted on the target image.

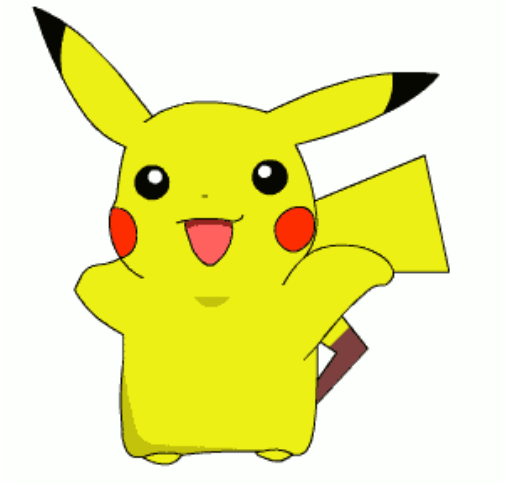
We then use Equation 1 to determine the score for a collage image.

$$Score = \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \sqrt{(t_{x,y}^{red} - c_{x,y}^{red})^2 + (t_{x,y}^{green} - c_{x,y}^{green})^2 + (t_{x,y}^{blue} - c_{x,y}^{blue})^2} \quad (1)$$

where w is the width in pixels of the target image; h is the height in pixels of the target image; $t_{x,y}^{red}$, $t_{x,y}^{green}$, $t_{x,y}^{blue}$ are respectively the red, green and blue colours of the target pixel at position (x, y) and $c_{x,y}^{red}$, $c_{x,y}^{green}$, $c_{x,y}^{blue}$ are respectively the red, green and blue colours of the collage pixel at position (x, y). The perfect score is zero. The smaller the score the better (i.e. the closer the collage is to the target image).

2.5 Sample Problem

This section describes an example of what your program should output. Figure 1 (a) is the **target** image representing Pikachu (perhaps the most annoying Pokémon). This is the image which you will need to approximate using a set of composition images in **CJP** format. A pictorial representation of the output **collage** is shown in Figure 1 (b). This is composed of eleven different rectangular copy&pastes. One white rectangle for the background, two black ones for the eyes, three red ones for the cheeks and mouth and five yellow ones for the body.



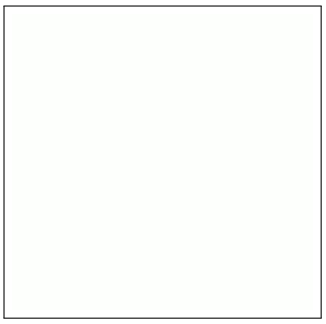
(a) The target image.

(b) A collage using 11 rectangles pasted from three composition images.

Figure 1: Pikachu and its approximate sibling are overjoyed to see you participate in the Programming Competition!

(a) Red and yellow composition image, used for body and cheeks.

(b) Black composition image used for eyes.



(c) White composition image with added black frame (for visibility). Used for background.

Figure 2: The composing images used to create the collage in Figure 1 (b). Note that these images are scaled down from their actual sizes for presentation reasons.

The output of your program should be a collage file (`.clg`), and in the Pikachu sample a possible valid solution is given in Listing 3.

Listing 3: `pikachu.clg` file.

```
1,0,0,300,300,0,0
0,0,0,130,115,60,60
0,0,0,45,78,45,22
0,10,10,70,35,160,55
0,0,0,70,50,190,120
0,0,0,105,110,75,165
0,0,210,18,30,60,128
0,0,210,35,25,105,140
0,0,210,21,30,165,130
2,0,0,20,25,80,105
2,0,0,18,22,150,100
```

The first column of `pikachu.clg` file is a zero-based index of the `composition_images.txt` file which points to the composing image from where the rectangle should be copied. The contents of the composition images file is shown in Listing 4. Note that this file contains absolute paths to the individual composition `.cjp` files, in this particular instance on a Linux system (hence the forward slash). These composition images refer to the images shown in Figure 2.

Listing 4: `composition_images.txt` file.

```
/programming_competition_2015/sample/yellow_red.cjp
/programming_competition_2015/sample/white.cjp
/programming_competition_2015/sample/black.cjp
```

This sample problem is clearly a simplified example. Here, the composing images are either bands of different colours (e.g. `yellow_red.cjp`) or monochromatic. In practice, there is **no** such limitation on the composition images employed.

2.6 Supplied Tools

The following software is supplied as-is, without warranty, guarantee or support. This document offers enough information to build the below tools yourself. These are just offered so you can concentrate on the actual task, and not on the peripheral functionality (e.g. viewing a `.cjp` file). Should you require any help with these tools, or wish to report bugs please contact The Judges on the Google group. Also, read the terms and conditions in the supplied `LICENSE.txt file.`

The following is a list of supplied tools. These programs run on the Java Virtual Machine (version 1.8 and later). The tools will be supplied via a `.zip` file for download[4] which needs to be unzipped and placed in the classpath. **None of the following tools support transparent `.png` files.**

2.6.1 Scoring Tool

The scoring tool, `CollageScore`, is run in the following manner:

```
java tools.CollageScore 100 target_image.cjp collage.clg compos_imgs.txt
```

Where the first parameter, `100`, is the number of maximum of collage pieces you can paste, `target_image.cjp` is the target image file, `collage.clg` is your program's output file containing the collage, and `compos_imgs.txt` is the file containing the list of composition images (one per line). In this case the number of lines in `collage.clg` cannot exceed 100.

This program outputs the score of your solution (smaller numbers indicate better scores; zero indicates perfect match/score).

2.6.2 Composition File Generator

Given a directory the composition file generator tool will create a `composition_images.txt` file which contains a list of `.cjp` files in that directory. `Composition File Generator`, is run in the following manner:

```
java tools.CompositionFileGenerator /my/directory
```

Where the first (and only) parameter, `/my/directory`, is the directory which you want to generate a composition file list for. The generated file is called `composition_images.txt`. This file will be created in the specified directory (e.g. `/my/directory`).

[4] From `http://www.cs.um.edu.mt/svrg/56ayl4778abtds377/Game_of_Codes/`

2.6.3 `cjp2png` Tool

Converts a `.cjp` image text file to a `.png` binary file you can view in a common image viewer utility (e.g. Gimp or Photoshop). This program is run in the following manner:

```
java tools.cjp2png input.cjp output.png
```

Where the first parameter `input.cjp` is the image text file (i.e. the target file or the composition image files), and `output.png` is the output binary file in `.png` format. The file `input.cjp` must exist.

2.6.4 `png2cjp` Tool

Converts a `.png` binary image file to a `.cjp` image text file. This program is run in the following manner:

```
java tools.png2cjp input.png output.cjp
```

Where the first parameter `input.png` is the input binary `.png` image file, and `output.png` is the output image text file in `.cjp` format. The file `input.png` must exist.

2.6.5 `collage2png` Tool

Converts a collage text file (`.clg`) to a `.png` image binary file. This program is run in the following manner:

```
java tools.collage2png input.clg compos_imgs.txt width height output.png
```

Where the first parameter `input.clg` is the input collage file in `.clg` format (as described in Section 2.3), `compos_imgs.txt` file containing the list of composition images (one per line), `width` is the width of target image (in pixels), `height` is the height of target image (in pixels) and, `output.png` is the output image binary file (`png` format).

Note: We are aware that the programs `cjp2png`, `png2cjp`, and `collage2png` do not follow standard Java coding convention (class names should start with an uppercase letter).

3 Judging Criteria

The judging panel will run all submissions on a number of target image files and a list of images to be used as the composition set. Each program will be given up to 1 minute to generate an output collage file. Programs not terminating on a particular problem within 1 minute, or producing wrong

output (e.g. exceeding allowed number of lines in collage files) will get no points for that problem. For each problem instance, the programs will be ranked (in ascending order) according to the computed score. A score is then given to each team according to their ranking: best 50 points, second 30 points, third 20 points, fourth 10 points and fifth 5 points. In the case of ties, the fastest program runtime execution is used as a tie breaker. The judging panel reserves the right to resolve any further tie in as fair a way as possible using a tie breaker.

4 Submission Rules

All submissions have to be in the form of a batch file (i.e. `.bat`) or shell script (i.e. `.sh`) which calls your program. This batch file will call your program executable (i.e. `the_wise_man_of_babylon`) has to accept the four parameters described in Section 2.1.

No new windows or graphical user interfaces are to be opened by your program. The program will be executed on an Intel-based PC and having 4GB of RAM. The supported 64-bit operating systems will be Microsoft Windows 8.1 and Ubuntu 14.04.3 LTS. Microsoft Windows 8.1 will have the latest version of the .NET Framework installed and both OSes will have the latest Java version installed. Specific language runtime libraries which are required have to be free and available for download. Any submissions that fail to execute on the mentioned environment or any submission that takes longer than 1 minute to output a result **and terminate** will not be considered by the judging panel.

Each registered team is allowed to submit as many solutions as it wishes. The last submission on Sunday, 22nd November 2015 at 20:00 ZST will be considered for the awards. After this deadline no corrections or resubmissions will be allowed. Submissions can only be done through the programming challenge website – note that the program must finish uploading before the aforementioned time.