ASCII Art
Homework Assignment 1

Rostislav Horčík

February 28, 2021

1 Introduction

The goal of this homework assignment is to practice applications of higher-order functions for processing lists of elements. You are supposed to implement a function generating an ascii art from input images following the specification in Section 2. Examples of such a transformation are depicted in Figures 1 and 2.

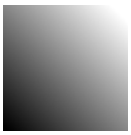

Figure 1: Example of a gradient image (left) and its transformation (right).

Figure 2: Another example

2 Specification

2.1 General setting

The implementation has to be done in the programming language `racket`. In order to work with images, it has to import the library `2htdp/image`. All your code is required to be in a single file called `hw1.rkt`. The file should behave as a module providing two functions `img->mat` and `ascii-art`. Thus your file should start with the following lines:

```
#lang racket
(require 2htdp/image)
```

```
(provide img->mat
         ascii-art)
```

To test your implementation you need to work with images. The images can be either loaded from a file for example as follows:

```
(define img (bitmap "grad.png"))
```

or can be created by the function provided by the library `2htdp/image`. For instance

```
(define img (circle 20 "solid" "blue"))
```

defines a blue disc of radius 20. For further details see `https://docs.racket-lang.org/teachpack/2htdpimage.html`.

2.2 Functions to be implemented

The first function `(img->mat img)` is a helper function transforming color images into matrices of intensities. To do that you need a couple of functions. First, the function `(image-width img)` returns the width of the image in pixels. Second, the function `(image->color-list img)` transforms the image into a list of pixels colors, reading from left to right, top to bottom. For instance,

```
(define img (triangle 5 "solid" "violet"))
(image->color-list img) =>
(#(struct:color 255 255 255 0)
 #(struct:color 255 255 255 1)
 #(struct:color 238 130 238 149)
 #(struct:color 255 128 255 2)
 #(struct:color 255 255 255 0)
 ...
 )
```

The colors should be converted into a grayscale intensity by the following function:

```
(define (RGB->grayscale color)
  (+ (* 0.3 (color-red color))
     (* 0.59 (color-green color))
     (* 0.11 (color-blue color)))
  )
```

Now applying the function `RGB->grayscale` to the list of colors, we get a list of intesities:

```
(map RGB->grayscale (image->color-list img)) =>
(255.0 255.0 174.28 180.07 255.0 255.0 174.69 174.28 174.1 255.0
171.46 174.28 174.28 174.28 172.69 174.28 174.28 174.28 174.28 173.69)
```

The function `img->mat` should return a matrix of intensities. So your task is to take the above list of intensities and transform it into a matrix, i.e., a list of rows where each row represents a horizontal line of pixels in the image. As the width of the above triangle image is 5, the output should look like

```
(img->mat img) =>
((255.0 255.0 174.28 180.07 255.0)
 (255.0 174.69 174.28 174.1 255.0)
 (171.46 174.28 174.28 174.28 172.69)
 (174.28 174.28 174.28 174.28 173.69))
```

As a second function, you are supposed to implement the function

```
(ascii-art width height chars)
```

returning a function of a single argument `img`. The returned function takes an image `img` and transforms it into a string approximating the input image. Thus the function is going to be called e.g. as follows:

```
(define chars " .,:;ox%#@")
((ascii-art 5 8 chars) (bitmap "grad.png"))
```

The `chars` argument is a string of characters we want to use to approximate intervals of intensities. Let d be the length of `chars`. An intensity i should be represented by a character whose index k in `chars` is computed by the following formula:

$$k = \left\lfloor \frac{d(255 - \lfloor i \rfloor)}{256} \right\rfloor \tag{1}$$

The notation $\lfloor x \rfloor$ denotes maximum integer number below x which can be computed in racket by the function `floor`. Thus for the highest intensity 255 we have $k = 0$ and for the lowest intesity 0 we have $k = d - 1$. The reason there is the inner floor function $\lfloor i \rfloor$ is that i might be slightly larger than 255 due to rounding errors.

The function returned by `ascii-art` should split the matrix of intensities into blocks. The size of blocks is given by the arguments `width` and `height`. Separation of the matrix into blocks is depicted in Figure 3. In case that width (resp. height) of the matrix is not divisible by `width` (resp. `height`) the incomplete blocks have to be removed from the matrix as is illustrated by the red area in Figure 3.

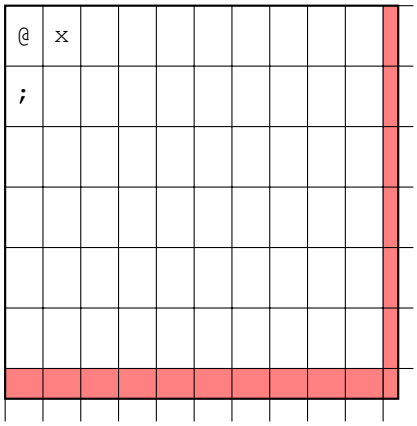

Figure 3: Separation of the matrix of intensities into blocks.

Once the matrix is separated into blocks, intensities in each blocks have to be averaged. The average intensities are then transformed into the corresponding characters by the formula (1), i.e., applying the function `list-ref` to `chars` and the index computed by (1). Finally, the resulting matrix of characters is transformed into a string composed out of the characters in the matrix each row followed by the newline character `"\n"`.

3 Example

Consider the following simple image composed of four gray rectangles of different intensities. The function `(make-color r g b)` creates a color object with respective RGB components. The function `above` places its image arguments on top of each other and analogously `beside` next to each other.

```
(define example
    (above
        (beside (rectangle 2 1 "solid" (make-color 0 0 0))
                (rectangle 3 1 "solid" (make-color 75 75 75)))
        (beside (rectangle 2 3 "solid" (make-color 180 180 180))
                (rectangle 3 3 "solid" (make-color 225 225 225)))
    ))
```

Evaluating the function `img->mat` on this image results in

After splitting into blocks of size 2×2 (the last column is omitted as the matrix width is 5) and computing average intensities we obtain

```
((90.0 150.0) (180.0 225.0))
```

Assuming that our characters approximating intensities are

```
(define chars " .,:;ox%#@")
```

the above intensities are represented by the following characters

```
((#\x #\;)
 (#\, #\.))
```

After transforming the above matrix of characters into a string, the function returned by `ascii-art` produces the following output:

```
((ascii-art 2 2 chars) example) => "x;\n,.\n"
```

To see the result better, you can use the function `display` as follows:

```
(display ((ascii-art 2 2 chars) example)) =>

x;
,.
```