Assignment 2g

[Niels Dreesen, Mustafa Hekmat Al Abdelamir, Jakup Højgaard Lützen]

24. september 2022

1 Report for assignment 2G

This assignment involved making a program exclusively with predetermined functions, and documenting it with XML. The assignment was an exercise in working with restrictions regarding the inputs and outputs of functions and the amount and variety of them.

Assignment $2g0$

In this section we are given 3 function description:

(a) addition of vectors

$$add : vec- > vec- > vec$$

(b) multiplication of a vector and a constant

$$mul : vec- > float- > vec$$

(c) rotation of a vector

$$rot : vec- > float- > vec$$

For function (a) we decide to split the vec type into 2 separate variables for each vector input. We do this in the function definition line for tidiness. We then write out an arithmetic expression wrapped into a tuple of form float*float, in other words a vec type.

```
let add (x1:float, y1:float) (x2:float, y2:float) =
  (x1 + x2, y1 + y2)
```

for function (b) and (c) we had similar considerations and used the same structure

Assignment $2g1$

This section is similar to $2g0$: We were tasked with making a function that converts a float tuple (vec) to an int tuple (int*int). We use the same structure again, except this time, we need to round the float. Instead of using the a function for this, we opted to use the more compact method of where you add 0.5 before typecasting, thus correctly rounding to the nearest int.

```
let toInt (v1:float, v2:float):vec : int*int =
  (int (v1 + 0.5), int (v2 + 0.5))
```

For function (b) we are supposed to define the input vector as a line originating from a pos position. The tricky part about this function is the canvas module which is used to define the lines takes integer tuples as inputs. This means that in addition to the appropriate calculation to place the line, we must convert the values to integers. We have decided to use multiple lines of code, for better readability:

```
let setVector (c:canvas) (Col:color) (vec:vec) (pos:vec) =
  let start = toInt pos
  let tempFloatVector = add vec pos
  let destination = toInt(tempFloatVector)
  do setLine c black start destination
```

For function (c) we are supposed to create a function which adds 36 spokes around the center. We separated the function in two parts. The first part is a draw function which defines the position and length of our vectors. We use v to define the first pair of coordinates at (200.0,0.0) and c as the second part, which is in the center ((height/2), (width/2)).

For the recursive part of (c) we have decided to split it into two cases (with pattern matching) One case is where we have drawn all the spokes and the function exits, and he other case is for when its supposed to draw a line. When supposed to draw a line, it does so by rotating the original vector v by $n * \frac{2\pi}{36}$ and defines a new vector from it, where n is the number of the spoke. In addition it will call the recursive part again with n-1 (to draw the next spoke in line). $n = 36$ at initialization and the recursive function exists when $n < 1$ as to iterate through all values n from 1 to 36. We are basically cutting 36 pizza slices and each line number n has n pizza slices of angle. The draw function draws the defined lines after the recursive function finishes and then exits with the canvas C as an output.

```
let draw (height:int) (width:int) (s:state) =
  let C = create (width:int) (height:int)
  let v = (300.0,0.0)
  let center= (float(height/2), float(width/2))
  let spokes = 36.0

  let rec drawSpokes (amount:float) =
    match amount with
      | n when 1.0 <= n ->
        setVector C black (vecRotate (s + amount*2.0*System.Math.PI/36.0) v) center
        drawSpokes (amount - 1.0)
      | _ -> ()
  do drawSpokes spokes
  C
```

For part (d) we are simply using runApp as well as defining a state (float) which acts as an angle offset to our vector definition in draw.

Figure 2

A screenshot of our Canvas for 2g1.