Technical Report
COMP1100 Assignment 1

Jacob Bos
ANU u7469354

April 4, 2022

Lab: Tuesday 11am

Tutor: Abhaas Goyal

Contents

1 Introduction **1**

2 Design Documentation **1**

3 Testing **2**

4 Reflection **3**

4.1 Technical Decisions 3

4.2 Assumptions 4

4.3 Reflection 4

1 Introduction

This Technical report documents the structure of the assignment solution and offers a reflective analysis of design choices made and the results and structure of the testing regime. The program is designed to take user inputs and produce a picture onscreen using the Haskell CodeWorld package.

2 Design Documentation

Part 1 consists of three functions, all use case matching to produce the specified outputs for given inputs. The two functions `nextColor` and `nextTool` both cycle through a sequence of values depending on the given value. `nextTool` only accepts empty tools and so via a wildcard case will return the input when this is not the case.

Part 2 Contains four functions the first of which, `colourNameToColour` in `module View` case matches elements of type `ColourName` and returns the same information in the typef `Colour` which CodeWorld uses. The second function `shapeToPicture` takes the information kept within type `Shape` and converts it to type `Picture` to be printed to the display. Where most inputs were case matched to equivalent CodeWorld functions the specifications of `Rectangle` were through some linear algebra converted to a `solidPolygon` of four points. `Cap` is a combination of the functions `clipped` and `circle` translated as the tool specifies. The third function `colourShapeToPicture` casematches inputs of type `colourShape`, returning a coloured shape of type `Picture`. The helper functions `distance` and `otherTriPoint` assisted implementation, the former calculating circle radii and the latter the third point of the isosceles triangle. Finally, the function `colourShapesToPicture` recursively runs through an input of type `[ColourShape]` and returns each member as a composite `Picture`.

Part 3 consists of one key function and six helper functions. The main function `handleEvent` cases on keystrokeand mouse key inputs to produce the intended picture. Presses of backspace and delete calls the function `deletePress` which removes the head of the list of shapes to remove the most recently added shape from the image. The spacebar input calls the function `endPoly` that takes any list stored in type `Tool` and then adds a polygon to the list of colourshapes. Key inputs of + or - call the functions `scaleRect` and `negScaleRect` respectively that increament the scaling factor stored in `rectangleTool`. Mouse presses call `pointPress` that cases on the shape tool being used to then store the pressed point in the tool. Further the helper `pointRel` is called upon mouse releases generally to complete a shape adding it to the list of shapes and returning an empty tool. There are two cases on `CapTool` to determine if it is

storing the second point or the y coordinate of the cutoff point.

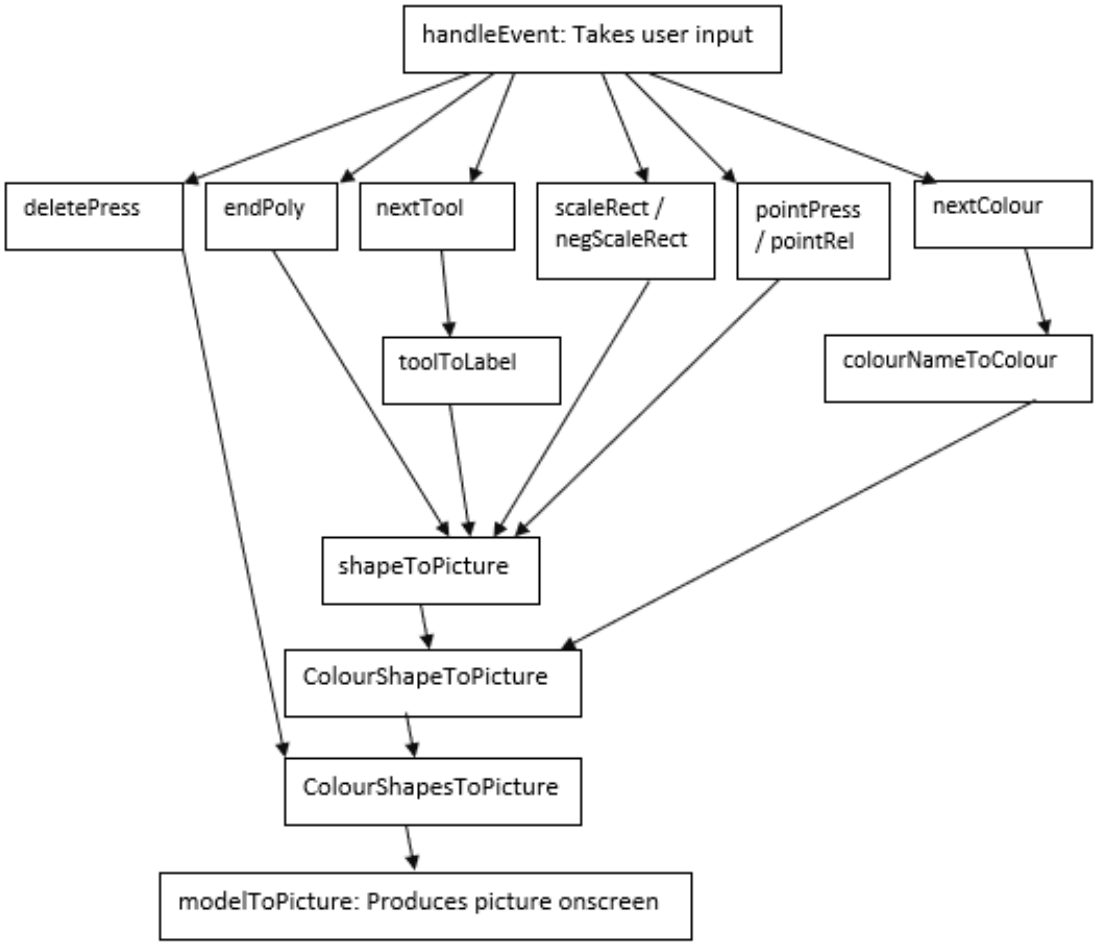

Figure 1: Function dependency graph - Arrows show direction of function calls

3 Testing

Part 1 composed of the functions `toolToLabel` `nextTool` and `nextColour` was tested using the provided black-box test file under the command `cabal v2-test`. It passed `1 of 1 test suites` indicating correctness. Further simple white-box tests were conducted within development calling functions with edge-case inputs in the terminal to ensure the case matching was error tolerant, eventually any such errors were deemed eliminated.

Parts 2 & 3 were tested firstly by removing all errors or warnings delivered by program compilation. As the Part 3 functions are designed to call the functions in both parts 1 and 2 it was decided it would be possible to test the functionality of parts 2 and 3 just through rigorous black-box testing of program

GUI response. Each shape tool was tested in as many input configurations as possible including colour. Further all key inputs were tested to ensure they produced the desired response. The program passed both testing regimes. Finally, testing concluded with some white-box tests for edge cases in inputs to check for crashes or specification violations. Firstly the delete command was tested on a blank canvas and did not have any unintended consequences. Next, various variations of capTool, the most complex tool, were tested in all four coordinate quadrants and various sizes. The tool returned the appropriate outputs for cut lines above and below the defined circle and also above and below the centre of but within the circle. Further the rectangle tool was tested that it was coded as clockwise which it was. Consequently as no errors could be found and everything held to specifications the program can be deemed correct.

4 Reflection

4.1 Technical Decisions

Part 1 used a case statement for all three functions as they were all nessisarially injective. Consequent to this there was no need to use guards. Part 2 Whilst `colourNameToColour` was very simple case matching, `shapeToPicture` is more complicated. It case matches on the tool used. For both the triangle and rectangle inputs the `solidPolygon` function was used to create the associated picture due to the specifications of the input not aligning well to a unique CodeWorld function. For the triangle the points used were the two given points and a third given by the function `otherTriPoint` that calculated the other isosceles point. For rectangles some vector maths is used in the definition to define the two other points as a translation of the first two points of a degree dictated by the scaling factor. The specifications for producing a cap required another nested case to determine if the cutoff was below the circle or not. if it was it would just return a circle, otherwise the desired cap would be produced. This was necessitated by the particular clip window and translations used. `colourShapeToPicture` applied both prior part 2 functions to return a coloured picture using the appropriate CodeWorld function. Finally it was necessary to recurse through the list of colourshapes in the `colourShapesToPicture` function as the list could be of any length. Part 3 was a simple implementation. The main function `handleEvent` cased on different inputs and would, instead of nesting cases, call appropriate helper function(s) which could case on the required part of the input to produce the desired output . To reduce the risk of errors most helpers were guarded by wildcards.

4.2 Assumptions

A gap in specifications for `handleEvent` necessitated the assumption that the function `pointRel` should re-initialise the scale factor of the rectangle tool at `1.0` upon the completion of a rectangle on release of cursor. This is hoped to enhance functionality when a user has used an extreme value of the scale factor and wants to quickly return to a reasonable scale factor for the next rectangle input. For `colourShapesToPicture` it was assumed that in case of an empty shapes list it should return a blank canvas, thus the prespecified CodeWorld function `blank` was used.

4.3 Reflection

Due to the simple nature of and efficacy of the program the author does see any impetus to buold the program differently. Further they did not run into any notable issues during development and did not find any significant strain on their technical skills. Due to its simplicity the program is very easy to read.