Technical Report
COMP1100 Assignment 1

Jacob Bos
ANU u7469354

April 10, 2022

Lab: Tuesday 11am

Tutor: Abhaas Goyal

Word-count beyond cover page at ≤ 1000 words

Contents

Introduction

This report documents the assignment solution's structure and provide analysis of design choices and testing. The program takes user inputs to produce a picture onscreen using the Haskell CodeWorld package.

1 Documentation

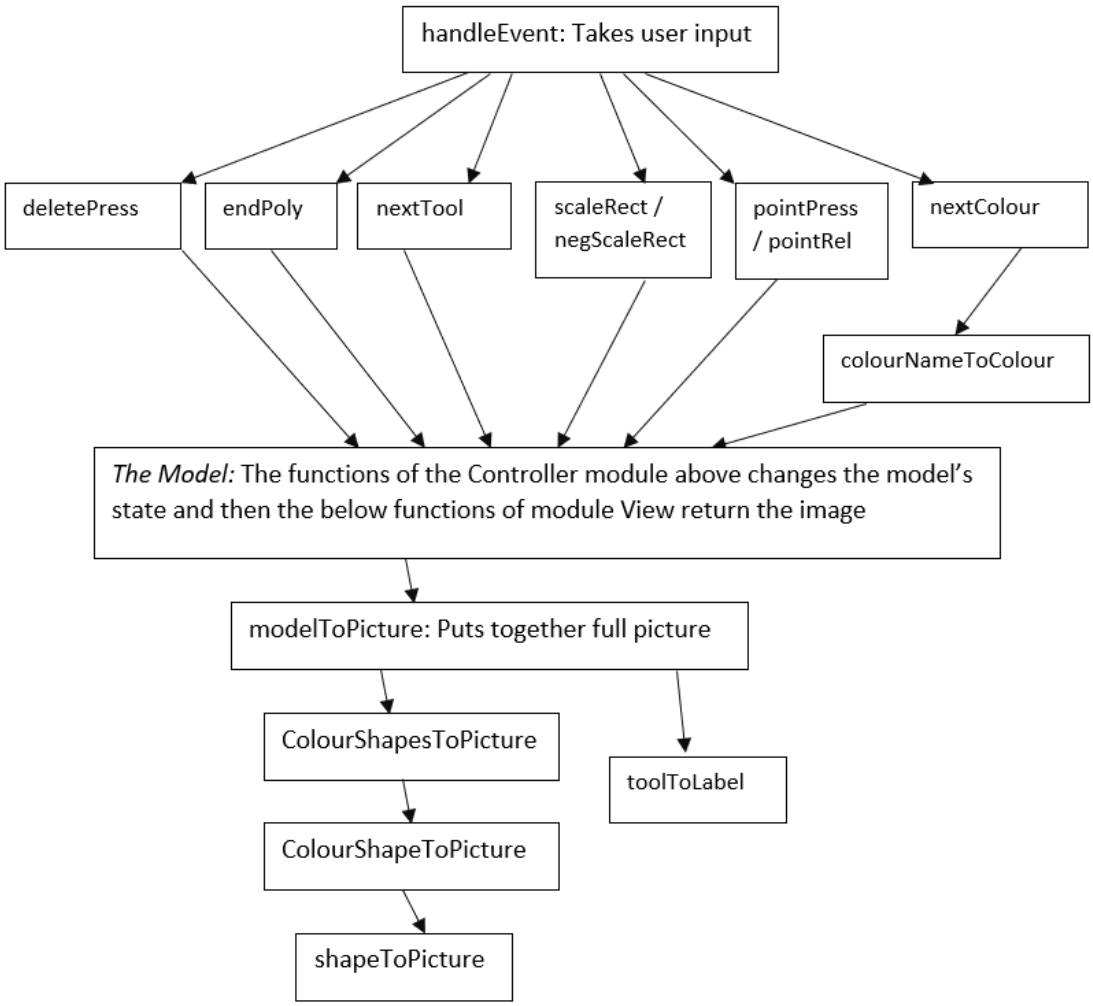
1.1 Design

Part 1 Has three functions, all case matching to cycle through model states. The functions `nextColor` and `nextTool` both cycle through a sequence of elements of their type. `nextTool` cycles through empty tools, using a wildcard to return the input when tools are non-empty.

Part 2 Contains four functions, firstly, `colourNameToColour` case matches `ColourName`s to return an equivalent CodeWorld `Colour`. Secondly, `shapeToPicture` converts an input type `Shape` to CodeWorld's `Picture`. Most inputs were case matched to directly equivalent CodeWorld functions. However, rectangle's specifications are converted to a `solidPolygon` rather than a `solidRectangle`. Cap is a combination of the functions `clipped` and `circle` with nested translations guarded to just return a circle in the case the cut-off is below the circle. Thirdly, `colourShapeToPicture` casematches inputs of type `colourShape` to return a coloured CodeWorld `Picture`. The helpers `distance` and `otherTriPoint` were used to calculate circle radii and the third isosceles triangle point respectively. Finally, `colourShapesToPicture` recurses through lists of type `[ColourShape]`, returning a composite `Picture`.

Part 3 contains the main function `handleEvent` and helpers. `handleEvent` cases on user inputs, changing the Model's state. Backspace and delete inputs call `deletePress` which removes the head of the list of shapes, deleting the image of the most recently drawn shape. The spacebar input calls `endPoly` that takes any list stored in `PolygonTool` and adds a coloured polygon the model. Key inputs of + or - call `scaleRect` and `negScaleRect` respectively that increment the scaling factor stored in `RectangleTool`. Mouse presses call `pointPress` that stores the point pressed in the tool. Mouse releases call `pointRel` which generally completes a shape, adding it to the colourshape list and returning an empty tool however there are two cases on `CapTool` determining if it is storing the circumference point or the cutoff level.

1.2 Structure



Function dependencies.

The top functions are from `module Controller`, allowing user control of the model whilst the functions within `module View` produce an image from the model.

1.3 Technical Decisions

Part 1 used case statements for all three functions as they were all injective with no need for guards. **Part 2** Whilst `colourNameToColour` uses simple case matching, `shapeToPicture` is more complicated, case matching on the tool it returns a CodeWorld `Picture`. For both triangle and rectangle tools the `solidPolygon` function was used to create the associated picture due to the specifications of the input not aligning well to a specific 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 the other two points are a translation of the first two points by a degree dictated by the scaling

factor. Necessitated by the clip window and translations used, cap required a guard to determine if the cutoff was below the circle or not. If so it just returns a circle, otherwise the desired cap will be produced. `colourShapeToPicture` used both prior part 2 functions and the CodeWorld `coloured` function to return a coloured picture. Finally it was necessary for `colourShapesToPicture`to recurse through the list of colourshapes 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 case exhaustion errors most helpers had a wildcard case.

1.4 Assumptions

A specification gap for `handleEvent` necessitated assuming that `pointRel` should leave the scale factor of the rectangle tool unchanged upon the completion of a rectangle rather than re-initialise. This is hoped to reduce the amount a user has to change the scale factor to sequentially draw similar rectangles. For `colourShapesToPicture` it was assumed that in case of an empty shapes list it should return a blank picture, and thus used the CodeWorld function `blank`.

2 Testing

Part 1 composed of three functions was tested using and passed the provided black-box test under `cabal v2-test` 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 compilation errors or warnings. 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 case key 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. Key input responses were also tested when mid-drawing of a shape. Further, the functions in part 2 were subjected to a number of supplementary Black-Box doctests

all of which passed. Consequently as no errors could be found and everything held to specifications the program can be deemed correct.

3 Reflection

Due to the simple nature of and efficacy of the program the author does see any impetus to build the program differently. Further they did not run into any notable development issues or any strain on their technical skills. Due to its simplicity the code is easily interpreted and well documented.