Multiple Choice Student Quiz Systems

Aditya Sinha

A thesis submitted for the degree of
Bachelor of Advanced Computing (Honours)
Supervised by: Dr. Eric McCreath
The Australian National University
October 2018

Except where otherwise indicated, this thesis is my own original work.

Aditya Sinha
October 2018

Acknowledgements

I would like to thank my research supervisor, Dr. Eric McCreath for his invaluable help, advice, support, and feedback over the course of one year. This project would not have been possible without his guidance.

I would also like to thank my friends and family, for always being there for me, for providing the much needed moral support in low times.

Contents

Index of Appendices

Abstract

Institutes of higher education have used periodic quizzes as a part of their continuous assessment schemes for quite some time. Although many private platforms now offer the tools to create online quizzes, there aren't any readily available ones that design and assess traditional, paper-based multiple-choice quizzes. And given that institutes still rely on paper-based examinations to assess students, this is still an important area.

This report discusses the making of one such software package. We go through the design and components of an all-in-one, functional quiz generation and assessment system. The first part of the report goes over the formats, libraries and tools required to generate a pdf quiz sheet from a yaml file. The second part gets into the knitty-gritty of the computer vision techniques that help extract and mark answers from scanned quiz sheets. All this is done in the context of a Java-based system. The system is validated on a set of six uniquely filled scans. The results provide concrete insights on what tools and approaches would be suitable for designing an effective system of this kind, for larger scale real-world application.

Keywords: PDF, Yaml, Java, GUI, Swing, Computer vision

1. Introduction

1.1 Motivation

Student assessments have been an integral part of most long-term educational courses. At worst, they serve as a tool to audit the student's level of familiarity with the coursework. At best, they catalyse more effective learning, beyond just testing.

Although numerous products and websites offer the tools to create online quizzes, these quizzes are limited in flexibility and format. There seem to be no open source/easily available offline softwares that help design traditional, on-paper quizzes with enough flexibility and control. The biggest motivation for this project is the potential for this learning, to be applied in making it easier for teachers and professors to administer tests.

Practical usage aside, the other major motivation for this project is its wide scope, and the range of learning opportunities resulting from it. In order to create a working software package that achieves all objectives, one must learn about:

1) Yaml format
2) PDF format
3) Java, and effective Object-Oriented Programming
4) PDF Libraries and their usage through Java
5) GUIs and Swings in Java
6) 2D Graphics in Java (BuffferedImage class and AffineTransforms)
7) Image processing
8) Computer vision techniques (Edge detection, Transforms, shape detection)

1.2 Objectives

The primary objective is to learn about how effective, unified quiz systems can be made. The project aims to make an easy-to-use Java based GUI that generates pdf quiz sheets from questions stored as yaml data. The system also makes assessment easier by processing the attempted scans, and semi-validates them by finding attempted answers.

Another objective of the project is to engage research and problem solving skills to learn about PDF libraries, Yaml packages, Image Processing, and Computer Vision and what they mean in the context of java, and learning about how all these different components can work synergistically in one GUI-integrated system.

The thesis hopes to explore how effective, unified quiz softwares can be developed. It goes over the concepts, methods, tools and techniques that are required to do so and demonstrates how to use them in tandem, while also discussing alternatives and future possibilities. This report weighs the merits and demerits of the various approaches and covers the difficulties and unexpected challenges that one might encounter in the process.

To this end, my contributions to this project include mapping out the components of the system, coming up with a design for it, researching and studying the available options, coming up with a plan for implementation, writing java code, making numerous libraries work together, and preparing a set of scans for testing, employing computer vision techniques for semi-validation and so on. All in all, my role was researching the possibilities, and developing the system. The system works, although it has its limitations. The report goes over all this and more in the sections that follow.

1.3 Outline

Section 1: Provides a brief gist of the project and goes over the motivation, and objectives of this project, while providing a high-level outline for this report.

Section 2: Goes over the alternatives currently in use for similar purposes. It also explains the key concepts used over the rest of the report, and needed to understand the working of this project such as the Yaml format, PDF libraries, and computer vision techniques.

Section 3: Covers the general design of how the system works for the user, and the design used from the developer's end and the reasoning behind it.

Section 4: Gets into the knitty-gritty of the development process. One by one goes over how each component was made, and how all of them fit together to form the system.

Section 5: We discuss what we observed on testing the system with a set of inputs.

Section 6: We discuss what can be concluded from the results, and what that would mean for future work along the same lines.

Section 7: Provides links and citations to all pieces of data referenced in this report.

2. Background

2.1. Multiple Choice Quizzes

In the past, oral examinations between teachers and students were considered to be a good means to judge the students' learning. Ancient philosophers like Socrates and Confucius would quiz their disciples with puzzling questions and challenging riddles. Formal exams were administered to candidates as far back as 210 B.C. in China, to select for civil services [1].

As time passed, the number of people interested in pursuing higher education increased, and so arose the need for quantitative evaluation, a type of testing that allowed objectively testing a students' learning (as opposed to qualitative testing which was more subjective). Later, E.L. Thorndike with his assistant, came up with the concept of multiple choice questions around early 20th century [2]. With the advent and popularisation of computers, scanners and printers, multiple choice questions became a preferred method of assessment, as large number of candidates could be tested and evaluated in short periods of time.

Advantages: This form of testing reduces teachers' bias by being more objective. It facilitates quick evaluation, and can be very effective if the questions are framed well.

Disadvantages: Only limited kinds of learning lend themselves to this format. Some fields of study are better suited to evaluation through short answers and essays. And sometimes with MCQs, a student's random attempt can still turn out to be correct.

2.2. Related Work

First we consider the options available to us in the form of other products/services on the market that perform the same tasks. There are numerous quiz softwares and websites that help create and conduct quizzes. Some of the most popular ones are Survey Monkey, Quiz Star, Easy Test Maker, Class Marker, ProProfs, and GoConqr [3].

The mentioned products help design and dispatch quizzes, but are very web-centric [4]. Quizzes are created in the form of web pages, and are attempted online. A good number of them rely on designing quizzes by selecting and combining from a set of pre-built templates. What we're looking for is a flexible, paper-based solution. Some of these services offer an option that exports quizzes to pdf that can be printed and distributed. But even with those, there are no features that help validate scans of these attempted quiz sheets in an automated fashion. The only systems that validate attempted quizzes do so for online attempts.

Moreover, most of these services tend to be freemium. Which means, they offer a small portion of their features for free but charge a monthly subscription price for full usability. The other ones available as downloadable software need purchase after the trial period is over.

OMRs: Optical Mark Readers have been conventionally used to detect answers on a multiple choice answer sheet. OMRs work by bouncing off a beam of light on the attempted page, and finding filled choices by detecting differences in reflectivity [5]. Usually, OMRs work on specialised forms. These forms are carefully designed and structured to optimise accuracy. This is where OMR systems can be a bit rigid. There are both open source and commercial OMR applications, although there aren't many that offer a

unified functionality, i.e. quiz design, generation, processing and validation in one.

All in all, there are not many viable alternatives that offer the degree of control and flexibility that we seek, in one comprehensive package.

2.3. The Yaml format

Yaml (Yaml Ain't Markup Language) is a human-readable data serialization language. Unlike JSON and XML which use explicitly marked blocks, Yaml uses line and whitespace indentations [6]. The difference in structure is illustrated below:

Yaml

```
Student:
  - firstName: John
  - lastName: Wick
```

JSON

```
{
  "firstName": "John",
  "lastName": "Wick",
}
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<student>
    <firstName>John</firstName>
    <lastName>Wick</lastName>
<student>
```

This project uses the YamlBeans java library to work with the yaml format.

```
import net.sourceforge.yamlbeans.YamlReader;
```

Yaml is easy to write, easy to read, export and process, lightweight in usage (saved in plain text format), and it structures information well. For these reasons, Yaml was used as the input format for question papers in this project.

2.4. PDF Libraries

This first part of the project involves creating a pdf quiz sheet and hence required a java package that can help process PDFs. Amongst the options available (i.e. iText, gnu.jpdf, pdfbox, FOP etc.), Apache's PDFBox offered the features desired to achieve the task at hand, was open source, and had a good deal of handy documentation available online. So it was selected for this project.

2.5. Computer Vision

Computer vision is the field of study that helps computers see things, recognise what they're seeing and process/categorise things in useful ways, in essence seeking to mimic the human visual system [7]. The second part of the project involves semi-automating the marking process for the attempted quiz sheets. This is where computer vision comes in. Computer vision techniques can be employed in detecting page borders, straightening a skewed scan, cropping it appropriately, and most importantly, in finding the attempted answers on the page. This will be discussed in more detail in the following sections.

Hough Transform

Some of the most important applications of computer vision involve being able to detect lines and shapes on an image. Hough Transform is a technique that helps with that. Usually, some type of edge detection algorithm is run on the input before the hough transform is applied. Having marked the edges, Hough Transform helps identify which edge pixels form lines or shapes together.

Let's consider two points on a line in the x-y plane

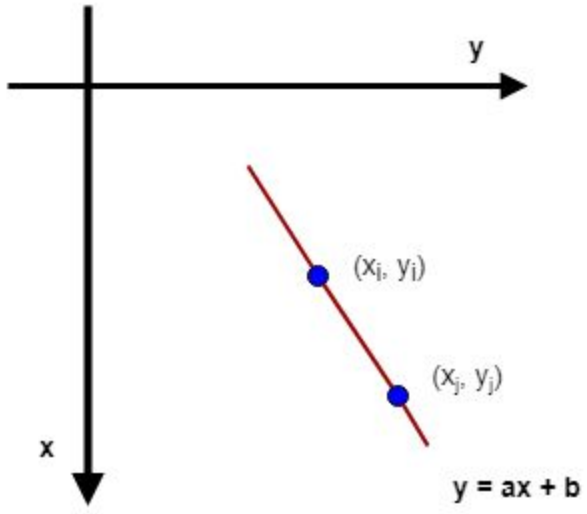


Figure 1.a) A line in the x-y plane and two points on it

The x-y plane represents a range of values of x and y, and which combination of those values lie on line y = ax + b given fixed parameters a and b. If we invert this, each point on the line y = ax + b in the x-y plane, is a line on the a-b plane (also called the feature space or parameter space) [8]. So each of the two points illustrated above in the x-y plane, is represented by a line in the a-b plane as shown below.

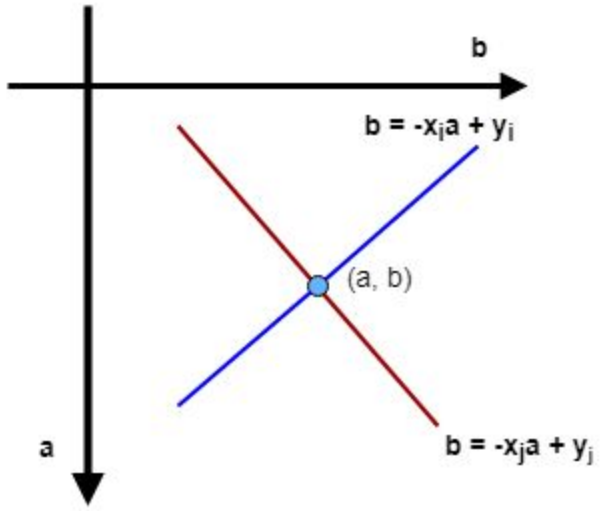


Figure 1.b) The two points from figure 1.a) mapped onto the feature space as two lines.

Conversely, the line y = ax + b that passes through both those points in the x-y plane, is represented by a point (a,b) in the a-b plane.

So given that the edge detection algorithm finds a bunch of pixels on the x-y plane, applying Hough transform will translate all these points into lines in the a-b plane. The intersection of all these lines will be a point, which when mapped back in the x-y plane will be a shape that passes through (or connects) all those points.

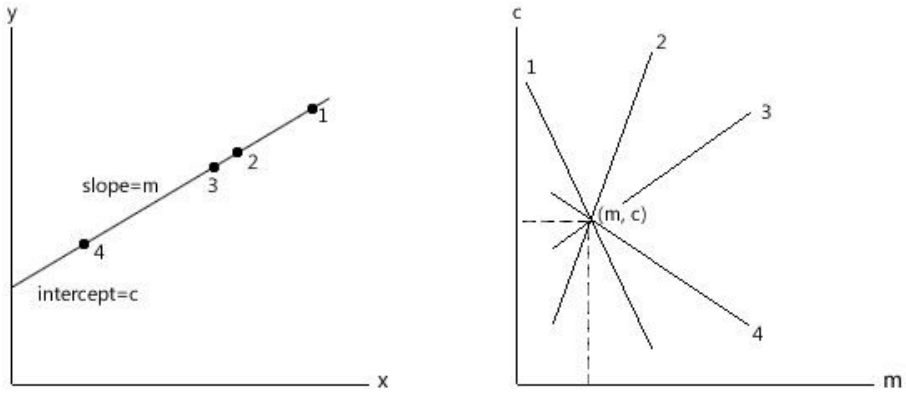


Figure 2. Points on the same edge map onto parameter space as lines which intersect [8].

The above example only aims to demonstrate the concept of Hough transforms and not an implementation. In practical application, Polar coordinates are used (r and θ), as the value of slope tends to be infinite for a vertical line in cartesian coordinates. The same procedure is followed, but instead of the parameter space axes being m and c (or a and b), they're r and θ, i.e. the distance and the angle.

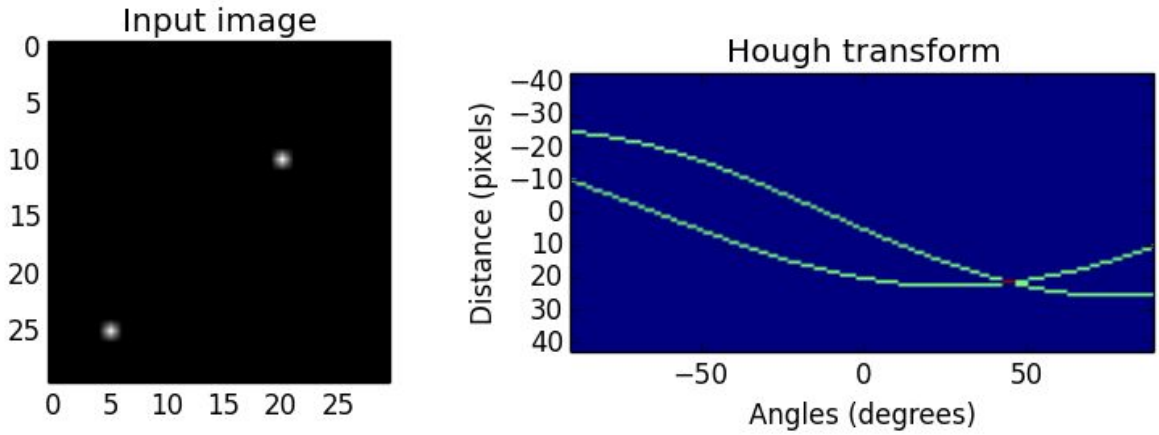


Figure 3. Two points on the cartesian plane mapped in the polar parameter space have their curves intersect at the angle and distance of the line that connects them [9].

Summing everything up, the algorithm goes as follows:

1) Process image: This involves cleaning the image up, noise reduction (using Gaussian or Median Blurring), making everything sharper and easier to detect by binary thresholding and so on.

2) Find edge pixels: Some sort of edge detection algorithm (sobel or canny for example) would help detect edge pixels. This is discussed in more detail in later sections.

3) Parametrization: The detected edge pixels are translated from cartesian space onto the parameter space after calculating a suitable range for r and θ.

4) Prepare an accumulator matrix: The accumulator matrix is a 2D matrix where each row corresponds to discretized intervals of the distance parameter (r) and each column corresponds to a finite range of θ values.

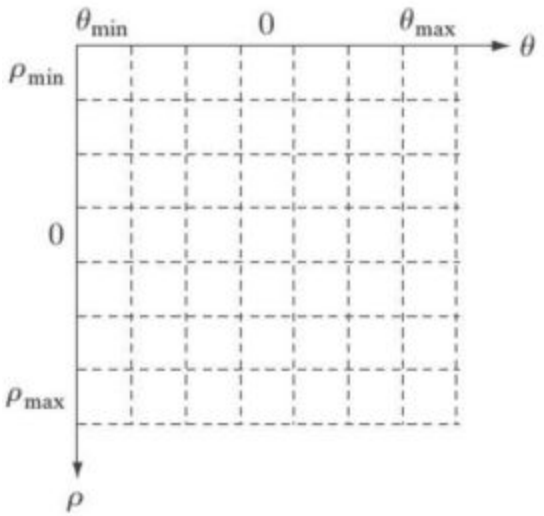


Figure 4. An accumulator matrix is used for the voting process [10].

5) Start voting: Looping through edge pixels, generate the sinusoidal curve for each, and increment the value (vote) at the corresponding cell in the accumulator matrix.

6) Peak finding: By the time the above step is finished, the maximum values in the matrix will correspond to the parameters of the most prominent lines.

For circles, Hough transform works similarly.

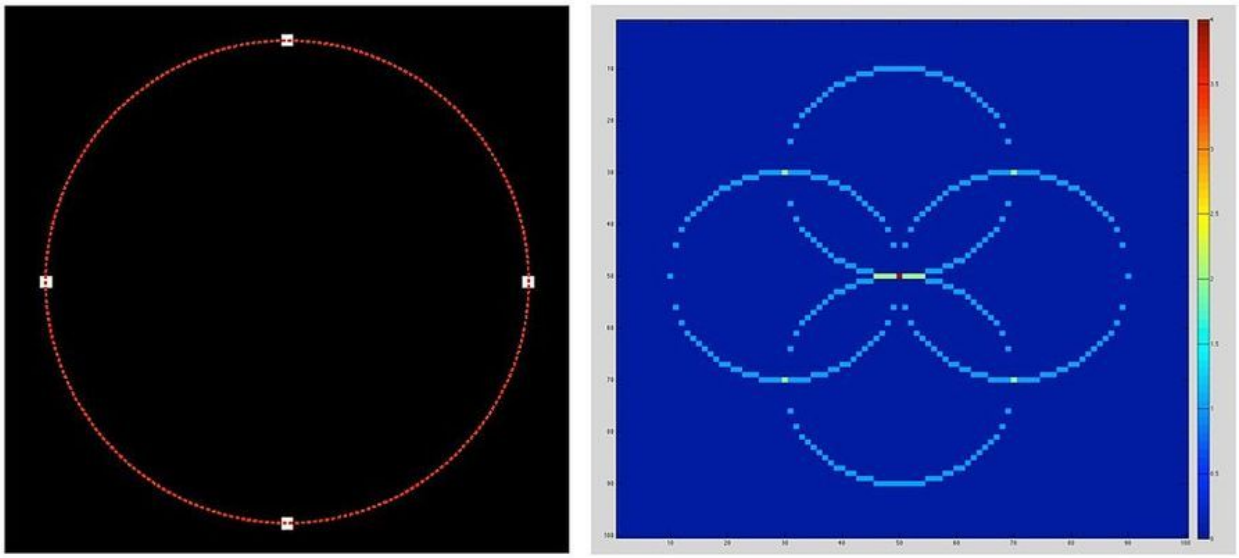


Figure 5: Four points on the edge of a circle plotted as four circles in parameter space [11].

After the initial processing and edge detection, a suitable accumulator matrix is created. The edge pixels are one by one plotted (as circles) in the parameter space, and all cells corresponding to the points lying on the circles are incremented in the matrix by one. Once this is done, searching for the local maxima in the accumulator space yields the circles that the edge points lie on [12].

Because the required input for hough transform is an image with detected edges, let's briefly discuss edge detection.

Edge Detection

Edge detection involves finding regions in an image where there are changes in intensity or colour. If a change is steep (or sharp) enough, we know that there's an edge there. There are many ways to look for edges, but edge detection techniques are usually of one of two types [13]:

Gradient-based methods: These methods look for sharp changes in the first derivative of the intensity graph of the image.

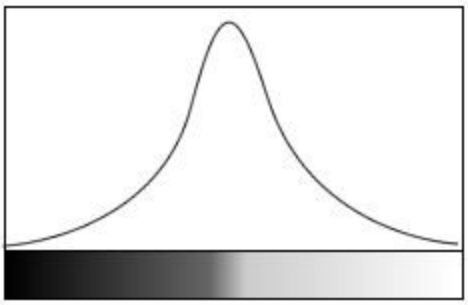


Figure 6: A first derivative plot showing a sharp peak [14].

Laplacian methods: These methods search for points of zero crossing in the second derivative plots of the image.

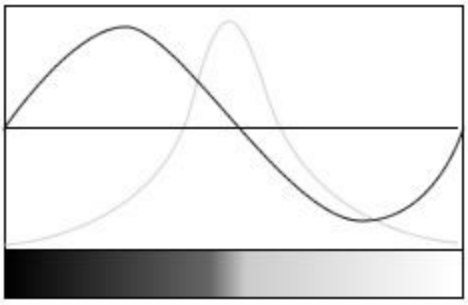


Figure 7: A second derivative graph with a crossing
in the middle (indicating the presence of an edge) [14].

A commonly used gradient-based method is the Sobel edge detector. It's an operator that approximates first derivatives by convolving two different kernels, one along the x-axis and the other along the y-axis the image.

-1	0	1
-2	0	2
-1	0	1

Horizontal

-1	-2	-1
0	0	0
-1	-2	-1

Vertical

Figure 8: Kernels used for Sobel edge detection [14].

Note: Convolution is the treatment/processing of one matrix using another in which the elements of one matrix are added with their neighbours, weighed according to the second matrix (or the kernel) [15].

2.6. BufferedImage

For a significant portion of the image work, this project used the java.awt.image.BufferedImage class. It's a class that facilitates easy access to the data of the image through its objects, making possible pixel-level control and manipulation.

BufferedImage is made up of two components:

The Raster: Contains the data that determines the colour at each pixel. Every pixel has one or more values allotted to it in the raster, called Sample(s). The raster is the collection of all the samples for all pixels in the image [16]. Within the raster, the data buffer contains the actual information about the samples for each pixel in primitive data types like byte and int. While the Sample model helps extract this information.

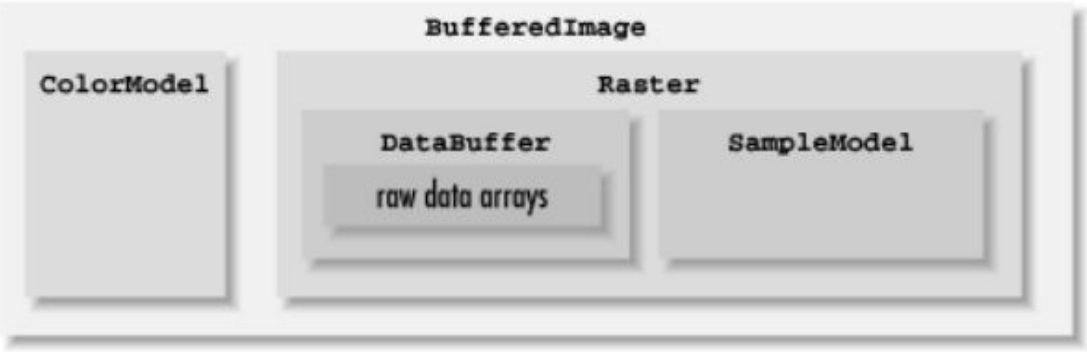


Figure 9: The components of BufferedImage[16].

The ColorModel: The task of the ColorModel is to interpret the samples obtained from the Raster as colour.

3. Design

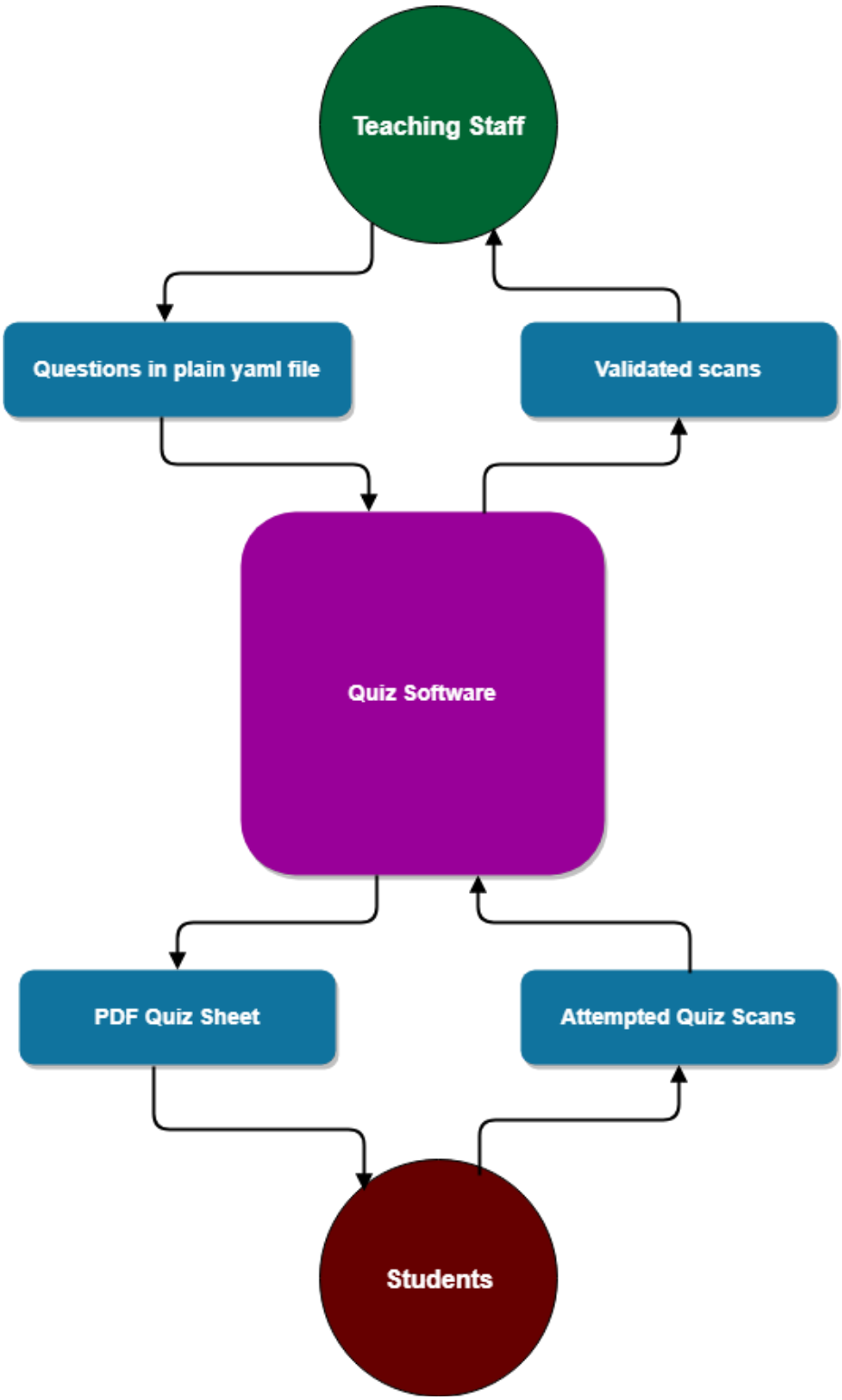


Figure 10: A high level overview of the working of the system

Figure 10 illustrates how the system works. The teaching staff frames a set of questions that they deem suitable for evaluating their students. These questions are put in a yaml file. The user can select the yaml file using our quiz system, and generate a well formatted pdf quiz. They can then print out this pdf and distribute it amongst students. Once the students are done, the user can have the attempted quizzes scanned. These scans can then be validated by the system.

Talking class-wise, the design is organised in three components.

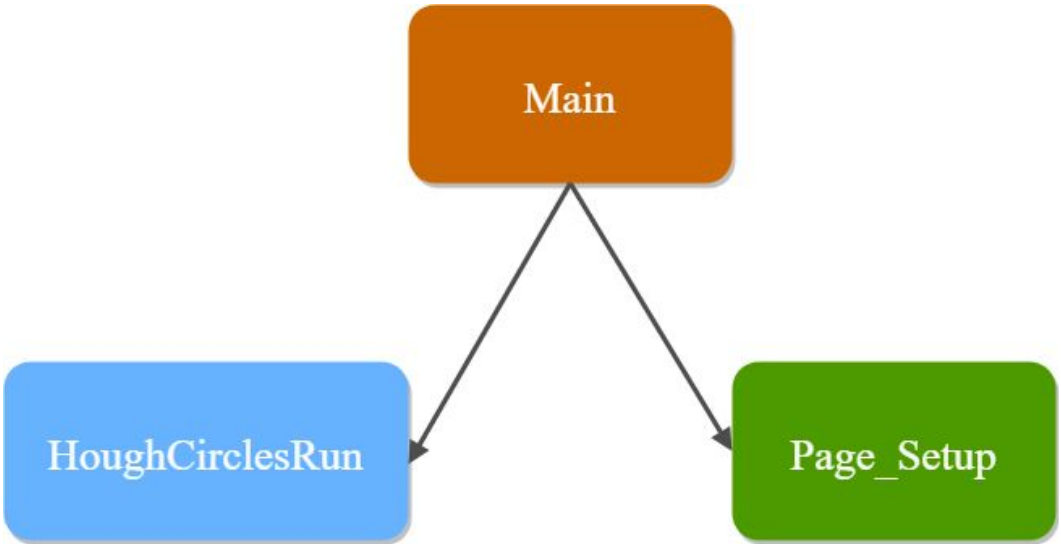


Figure 11: The system was modularised in three parts

The first part is the class **Main**. It serves as the entity that houses all GUI related methods and attributes. It invokes a Swing-based app that provides the user a choice between *Generating a quiz sheet* and *Marking a scanned quiz*. So this is the bit that acts as an interface between the user and the rest of the system, and thus first in order of execution. GUI aside the Main class also contains methods that process the input yaml file, extracting important information that can be printed onto the PDF file. Following is a class diagram for the Main class:

Main
frame : JFrame menuBar : JMenuBar fileMenu: JMenu editMenu : JMenu generate_new_quiz : JMenuItem mark_scanned_quiz : JMenuItem fc : JFileChooser panel : JPanel scan_img : JLabel yaml_file : File scan_file : File question_paper : String pdf_save_name : String answers : List read_scan : BufferedImage
+ run () : void + make_new_pdf (String, String) : void + create_png_from_pdf (File) : void + static main (String) : void - deskew_image (BufferedImage, int, int, int, int) : BufferedImage - crop_image (BufferedImage, int, int) : BufferedImage - remove_and_store_answers (String) : Temp_Container - format_Yaml (Object) : String

Figure 12: Class Main

Page_Setup deals with everything that has to do with the *Generating a Quiz* option. Having extracted all the necessary bits from the yaml file, the make_new_pdf method is called upon by the running thread. It is this

method that creates an object of Page_Setup type. This object contains all relevant measurements regarding the structure and formatting of the PDF file. It contains attributes that control paragraph spacing, line spacing, font size and so on. Here's a self-explanatory class diagram:

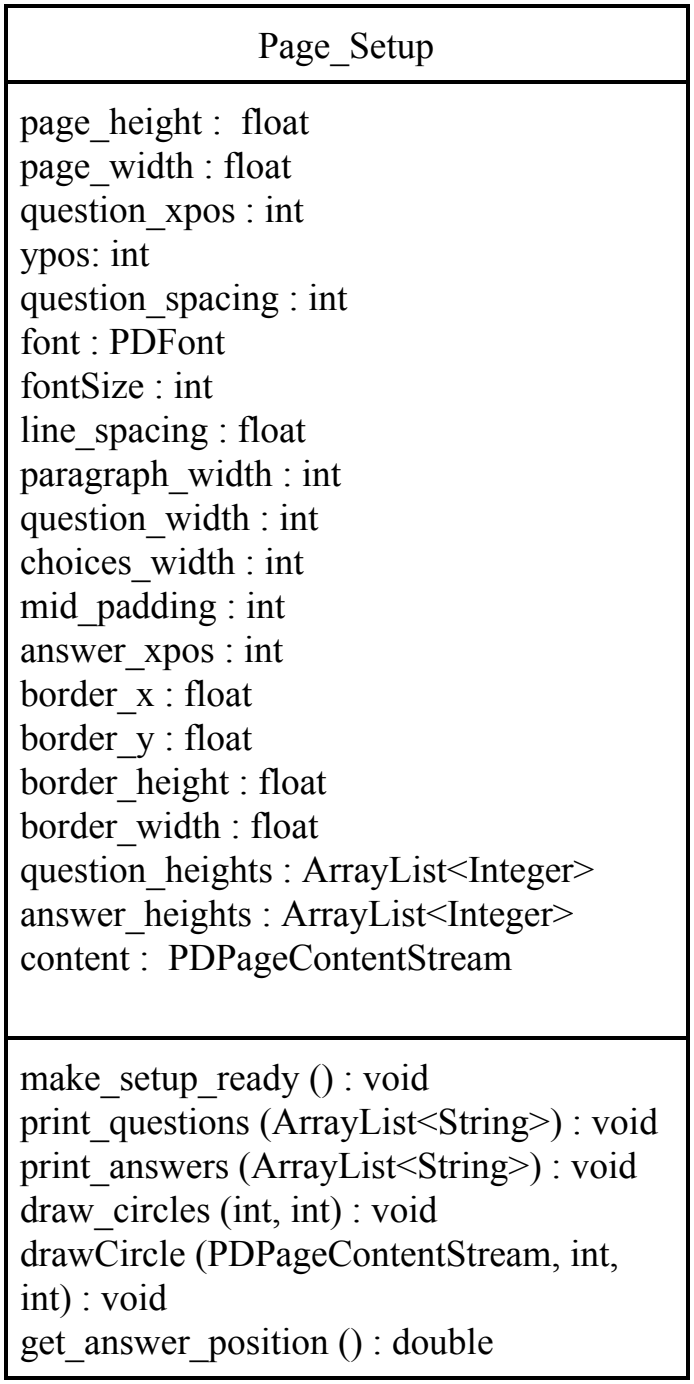


Figure 13: Class Page_Setup

It's when the user needs to validate the attempted quiz scans, that the third component comes to play. It only has the run() function, and utilizes Hough Transforms (covered in Background section) from the Java OpenCV library to detect filled circles on the attempted sheets.

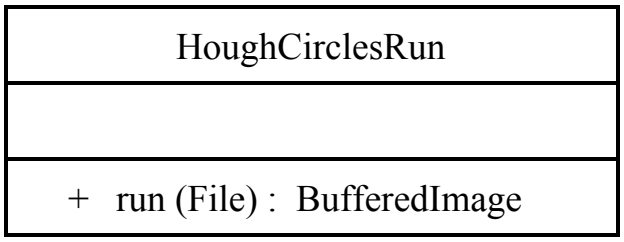


Figure 14: Class HoughCirclesRun

Note: These tables are not comprehensive. For brevity's sake, only the most important methods, attributes and classes have been listed.

4. Implementation

Because the system was meant to be a unified one, it contained multiple moving parts. To make sure everything worked, it was decided that the individual components be developed first, making sure each worked on their own. In the later stage, all components could be tied together, giving enough room to move things around if something did not fit in the first go. From starting to end, the software development process involved the following stages:

Stage One

- Accept the yaml file as input
- Extract the text from the yaml
- Format it to prepare for printing on the pdf
- Sort this text into questions and answers
- Print the questions and answers at on a blank page
- Draw answer circles and structure other details
- Save as a pdf file

Stage Two

- Accept scans of filled answer sheets
- Process scans to make them clearer
- Find answers

Stage Three

- Integrate all of the above components in a functional, intuitive GUI.
- Perform testing

4.1. Working with the yaml file

The first step in yaml processing was to deserialize the contents of the Yaml document to Java objects. YamlReader from the java YamlBeans library was

well equipped to handle this task, and the yaml file was read into an object of Object type.

At this stage, there was a choice between converting this Object into a Map type or a String type. Because most yaml documents can be read as lists of entries, converting to Map would have been a common route. But owing to potential difficulties in dealing with nested entries and considering ease of formatting, it was converted into a String object instead.

Note: Serialization is the process of converting an object into a sequence of bytes which can be stored to a disk or database or can be sent through streams. The reverse process of creating an object(s) from a sequence of bytes is called deserialization.

4.2. String processing

Although we now had the Yaml document in one string, it was just a string representation of the map object, and needed further processing. For example, the Yaml snippet (previously mentioned in section 2.3) would turn out as the following string:

```
{student=[{firstname=John}, {Lastname=Wick}]}
```

The relevant pieces of information in the above string are student, firstname, john, lastname and Wick. But they're obscured by extra symbols like square brackets and commas, and equals to signs. Although these symbols could have been helpful in navigating and retrieving information if this was a map, they're not much use in string form.

So they were removed using String processing methods. The format_Yaml method in the code does this by replacing unnecessary

symbols with blanks and new lines, thus making the string easier to read. (The String replace() function was used for this purpose)

Because both questions and answers are preceded by headings of their names in how the yaml file is written, the method remove_questions_and_answers() extracted them using further string processing, thus storing them separately.

4.3. Creating a PDF file

A new document was created using the Apache PDFBox library's PDDocument class. Then pages were added to this document as PDPage objects. Because PDFBox requires everything to be printed onto the pdf using content streams, we created and used a PDPageContentStream object.

As discussed in the Design section, the part that deals with the pdf has been encapsulated in a different module from the Main interface. To gain access to all those methods and attributes, a Page_Setup object was created. This Page_Setup object, in its attributes, contains important values such as question width, positioning, padding, border coordinates, thickness and so on. (For more detail, refer to Figure 13.)

At the start, the setup was made ready. This required adding a border of suitable thickness and size around the page. Deciding on where the beginning text would be printed, deciding on the font size, and line spacing, and making sure that the content stream was open. All these tasks were put in one method (make_setup_ready()), meant to be called after the setup was created.

The questions and choices previously extracted from the input string were then printed onto the pdf using the showText() method (from PDContentStream) and updating the offset to suit the desired layout.

One of the more troubling aspect of printing things onto the document was how to position and align everything. In case of questions and answers, a good time was spent experimenting with how they could be structured well on the page, using the least amount of time, number of variables and the least number of resets to the offset position.

In case of drawing circles, things were a little more complicated. The first issue was that off putting circular shapes onto paper. Although, PDFBox offered an easy way to draw rectangles, there did not seem to be a readily available equivalent of addRect, that could draw circles. The other way to go about it was getting ContentStream to print a small image of a hollow circle with a transparent background onto the page.

In time, it was found that that method was harder to implement than I expected, and an easier way was found amongst the answers on StackOverflow. The solution involved drawing circles using a combination of bezier curves. Drawing circles aside, the other issue was positioning them. Eventually a solution was found by using the distances at which the answers had been printed (from the edge), to find a starting position for printing the circles, and then iteratively adding answer spacing to draw the rest. The circles were drawn alongside the printed choices, each circle being a combination of 4 bezier curves drawn using the stroke() method. By the end of this process, the bordered page was ready with the questions, choices and circles laid out in a structured form. The content stream was closed and the document saved as a file with a ".pdf" extension.

4.4. Processing the scan

The second big component of the system was its ability to find answers in attempted quizzes. The scans of attempted quiz sheets were fed into the system. The first task was to ensure a consistent format for all scans. So if

the scan was a png file, it was left as it was. The png file was read into a BufferedImage object using the ImageIO.read() method.

However if the scan was in pdf format, it was converted to png. This was done using the PDFRenderer class from the PDFBox library. The pdf scan can be read as a File object, that can be further read as a PDDocument object. This PDDocument object was used to initialise the renderer object. The renderImageWithDPI() method of class PDFRenderer converts the document into a BufferedImage of the specified type.

Deskewing

More often than not, scans from printers are skewed. So our first step was to ensure that the skew was corrected. The approach used in our system uses a downward traversal of the image from its top-edge, pixel by pixel until it finds the black border. This traversal is done at two points (called anchor points ap_1 and ap_2), one on each side (right and left) of the top edge. As we traverse downwards we check the RGB value of each pixel on the path. This was fairly easily achieved using the getRGB() method of BufferedImage. In fact, this level of control and manipulation was a prime motivator for working with the scan as a BufferedImage.

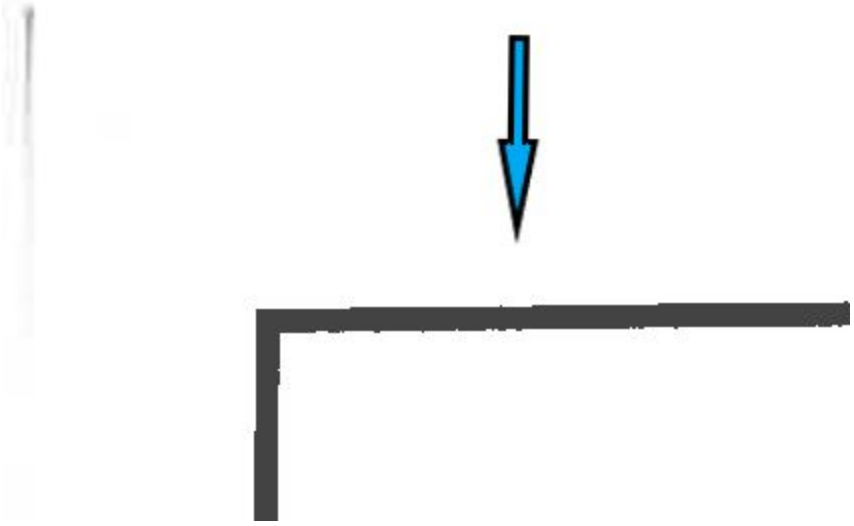


Figure 14: Reaching the border required moving downwards
pixel by pixel until the colour black was found.

Once the black border is reached, if the y-coordinate value of the two points (ap_1 and ap_2) is not the same, they are at two different distances from the top. Because the top edge of the border would ideally be horizontal, the points should ideally be at the same height. This difference indicates the presence of a skew. The degree of the skew was calculated by finding the angle of the line that connects the two points. Using AffineTransforms the image was rotated by the skew angle in the opposite direction, thus correcting it.

Cropping

Having the image deskewed, the next step was to crop out the unnecessary bits. The getSubImage() method can obtain a specified rectangular section of a BufferedImage, although it needs the coordinates of a corner, height and width. In order to find the border corner, we traversed downwards pixel by pixel from the left side of the top edge (just like deskewing). But instead of also doing the same on the right side of the top-edge, the second traversal was started from the left-edge towards the right.

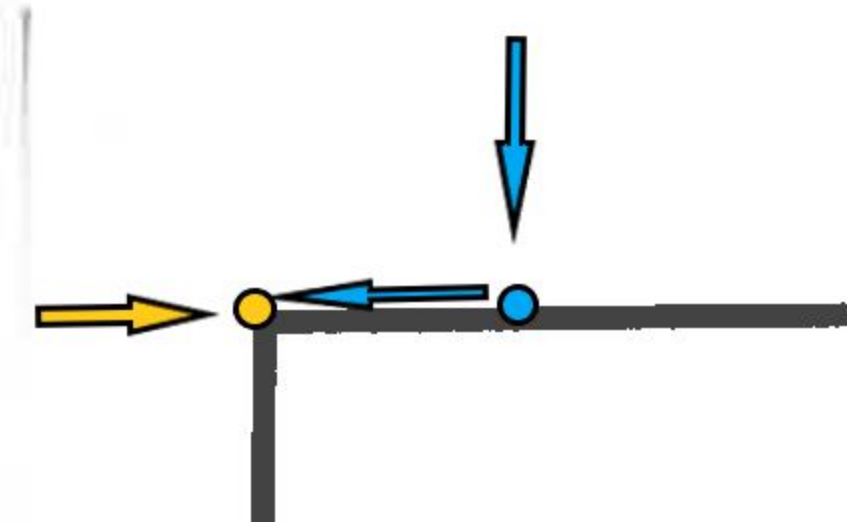


Figure 15: Cropping required finding the corner of the border.

Finding one anchor point (ap_1) on the top edge of the border and a second on the left edge of the border (ap_2), would allow us to traverse towards the left from the first anchor point (ap_1) and upwards from the second (ap_2). This would eventually lead us to the intersection of the two paths. This would be the corner of the black border.

Using similar traversal from the right edge of the scan towards the left would help us find the right side of the border. The difference between the x-coordinate of this pixel, and that of the previously reached corner pixel would yield the horizontal distance between the two points, i.e. the crop width. An appropriate crop_height could be found by subtracting twice the y-coordinate value from the height of the image. Having all required parameters, the getSubImage function obtained the most important portion for further use, and the rest was cropped out.

4.5. More processing to detect circles

With the scan deskewed and cropped, we could look for filled circles now. The first strategy was to pick a region on the image that was likely to contain the column of circles (for all choices from all questions), then scan downwards in a circle of pixels. The stride, size and positioning of this circle could be changed to our requirements, making this approach very malleable. The pixels within the circle would have their RGB values stored in an ArrayList. If the average value from the scanned circle of pixels was sufficiently close to black, it could be concluded (with some determinable degree of confidence) that the student had selected that option. But there were issues with this approach:

(Note: A tentative version of that code can be found in Appendix E.)

- It was difficult to determine the stride of this circle. It could end up scanning the top and/or bottom halves of all circles instead of actually overlapping with them.

- There seemed to be no clear way of accounting for what the student had done outside the circle. Even if the circle had an average RGB value corresponding to the colour black (meaning the circle was filled), there was no way of knowing if the filled circle was slashed outwards (meaning it had been later crossed out)

- There seemed to be no clear way of finding what would be a good Average Colour threshold to decide which circles were "filled enough" for the option to be considered selected

- With inconsistently filled circles, the average colour value might not have been the best metric to check if they were filled. We did not know how the value would reflect changes in consistency, for example, a fully filled circle, a lined circle, a dotted circle, a circle almost fully filled except a blank crescent on the side.

Considering all these problems with the first approach, it was decided that using circle detection through Hough Transform would be more suitable. It would not entirely solve all of these issues, but it would easily bypass a good number of them.

Some further processing of the image was required before Hough Transform could be applied. This was all done using the Java OpenCV library, it was the most comprehensive, easily accessible and compatible tool available at the time.

Note: A significant portion of the code in this section was an aptly modified version of some of the code available in the Open CV docs.

What follows was coded in the HoughCirclesRun class, as the third major component of the system. The File (the cropped deskewed image) was first loaded into a Mat object. In openCV, Mat represents an n dimensional array. The matrix was then converted into Grayscale to correct any accidental occurrences of colour, using the cvtColor() method from Imgproc. It was then converted into a binary image using thresholding. This was done to make sure all pixels stand out clear and sharp. As the final step, the MedianBlur filter was applied. The median filter traverses the image replacing the value at each pixel with the median of its neighbouring pixels. It's very effective for reducing noise while preserving edges.

OpenCV's HoughCircles() method was run over this processed image, storing the centres and radii of all detected circles in Mat object circles. Once this was finished the circles were marked with a pink ring around them to indicate detection by the system.

4.6. The GUI

The GUI was the last component to be coded. This was done to make sure I knew exactly what went where in the GUI. Having developed the other components before, it was only a matter of creating an easy-to-use GUI and repositioning the components to work through the GUI.

The GUI used Swing fundamentals like JFrame, JMenu, JPanel and JLabel to build a basic but easy-to-use interface.

On clicking on any of the menu items in Figure 16, their respective action listeners would sense the event occurrence and respond. The actionPerformed methods were overwritten to use a JFileChooser object to help users navigate to their desired files in their directories. The selected file was then further used for either generating a new pdf quiz or semi-validation (whichever was selected).

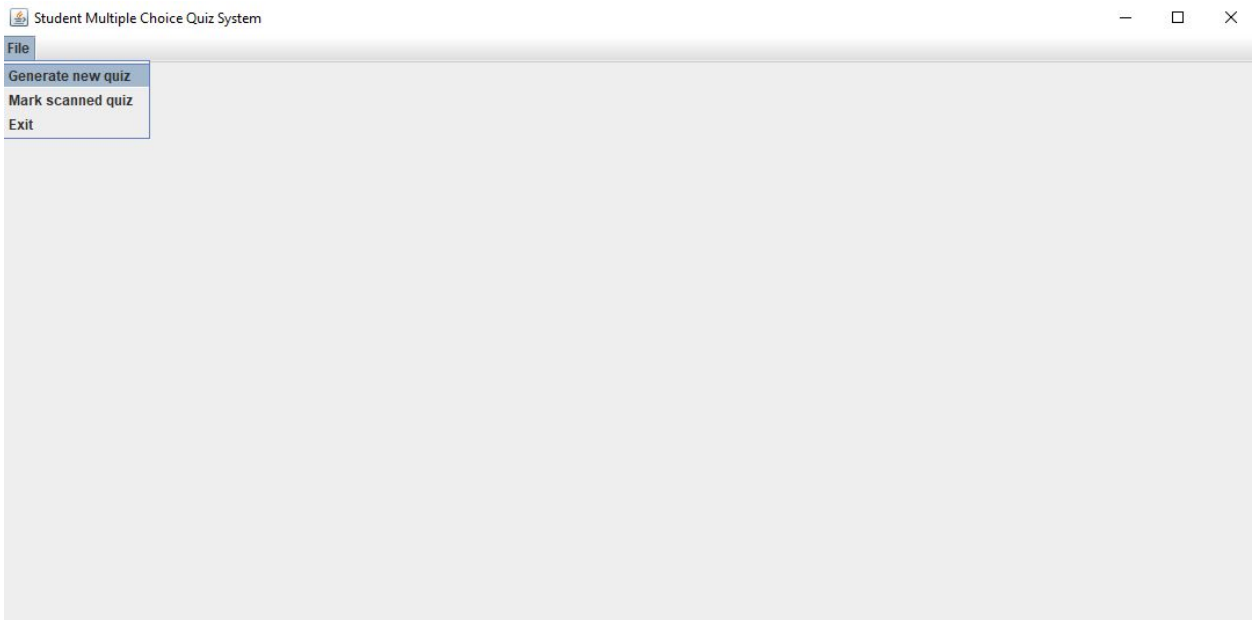


Figure 16: The File menu options

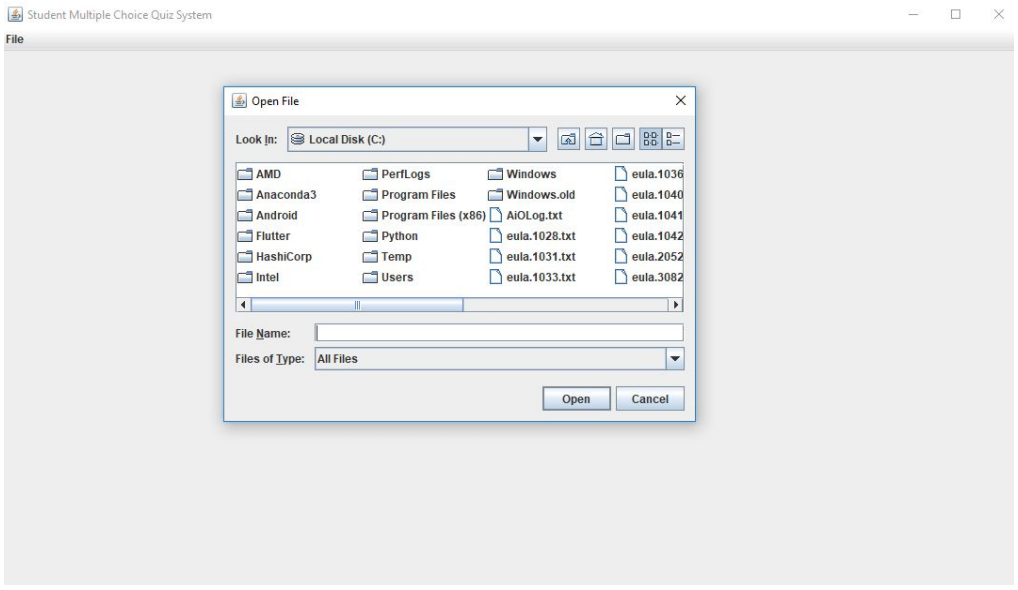


Figure 17: Navigating to find a scan or yaml file

All in all, in the entire run of implementation, the biggest chunk of time was taken in the process of determining which approach and tools to use, and the best way to use them. Starting with drawing borders on the page, there was the option of doing it with discrete lines (possibly giving more stylistic freedom) and then the option of using addRect() from PDFbox.

Similarly, when drawing circles there were more than one ways of going about it. The more suitable one could only become clear after a fair amount of trial and error. Even amongst all these steps, the greatest amount of effort and time was taken up by deskewing and cropping.

The two steps do not contribute directly to the output of the system, but make a significant difference. Because their implementation was started in the first half of stage two, there were several ways to go about doing them, and the Open CV library had not been used with this project until then. With deskewing, some form of edge detection algorithm (possibly coupled with Hough Line transform) would have done it easier. Some form of corner detection algorithm using OpenCV would have similarly simplified the cropping process.

With deskewing, there were more than one ways a page could be skewed in. The first scans that were used to test the system were brought in using the CamScanner app. This app enables scanning documents using a phone camera, which meant that aside from being rotated at an angle, the page could also have a slight (upwards or downwards) curvature to it (a bulge or a cave). With those scans, there were still issues after correcting the first kind of skew. More time was spent in finding if something could be done to refine the results. Later, the implementation process moved over to the next stage. Considering the fact that this software was aimed at academic institutes and formal assessments, it was clear that the input would come from traditional lid-equipped scanners. They would inevitably flatten the sheet onto the scanning surface, thus eliminating the need to spend any more effort in the straightening process on the system's end.

5. Results

The following is how the three components of the system performed over all:

1) The GUI was intuitive, easy to use, and worked as intended. Although opening new files required running the program again, and navigating through the directories.

2) The PDF generator successfully generated a PDF from the selected yaml file.

3) The answer detector had mixed results. I used a set of six uniquely attempted quiz scans for testing. The first two of these scans had perfectly attempted answers, i.e. perfectly filled circles. The second two had flawed attempts mixed with good attempts, crossed circles, partly filled circles and so on. The last two scans had poorly attempted answers, deviating far from expected attempts.

The answer detector was coded to draw pink rings around detected circles and display the result in the GUI window. The full results can be seen in from Appendix H to M. With the exception of the 2nd and 3rd questions in scan-1, the detector worked well when looking for well filled circles. It detected all circles that were consistently filled and circular in shape. Why it did not detect questions 2 and 3 in scan-1 is still unclear. In cases where there was a slight deviation in either shape or consistency, the detector did not recognise the attempt as a circle.

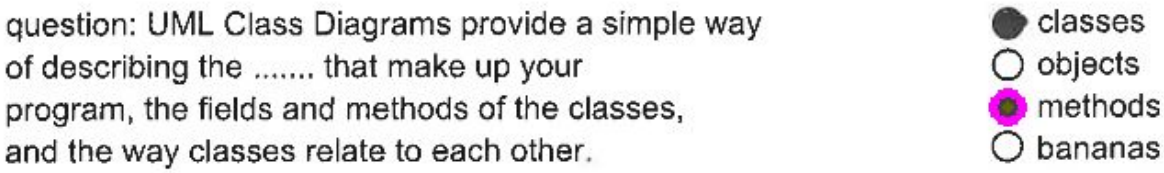


Figure 18: Scan 6, question 1

In the above example (question 1 from scan-6), the detector recognizes option 3 (methods) as a valid attempt, though not option 1 (classes). Although this is in one way advantageous, in that a filled circle that was slashed/crossed out later will not be detected, it's also risky, because human attempted quiz sheets will not always contain perfectly filled circles. On the contrary, a timed exam is likely to pressure students into hastily attempting answers, causing some ink spillage outside of the circles. The above is an example of how a minor inconsistency in shape introduces a slippage.

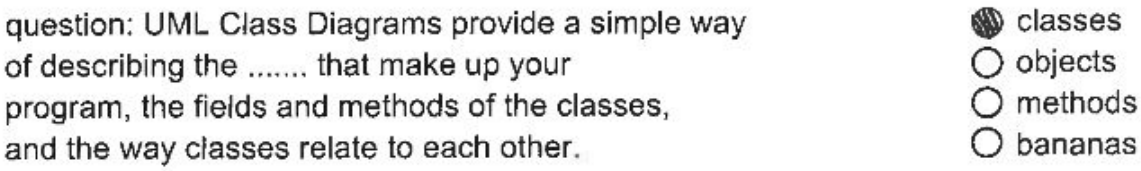


Figure 19: Scan 5, question 1

There is also this other situation where if a circle is filled in the correct shape but not consistently enough, it may not be detected as a filled circle. As in the previous case, the advantage of this being it accounts for a situation where a student starts attempting an option but changes mind in the midst of it, leaving the circle inconsistently filled. The attempt going undetected in that case serves the system. However, in a situation where the student filled the circle hurriedly, or their pen did not work well, this is something to be cautious about.

6. Conclusion and Future Work

Out of the three big components of the system, all three are functional and extensible. However only the first two (the gui, and the pdf generator) fulfill their roles completely. The third works with partial results, and needs further optimisation. Let's go over them one by one. The GUI could be further improved by:

1) Repositioning the draw code to the repaint() method for the panel that displays the image. This would fix the need to close the program for opening a new file/performing a different task.

2) Placing the current/specified folder structure on the sidebar of the main panel. This would bypass the need to select the file from by navigating through to the directory everytime. It would also allow quickly switching between different file in the same directory and instantly seeing the results reflected.

The processing of scans (deskewing and cropping) although functional, would be more tricky if there was more noise in the scans (for e.g.: ink spots scattered on the paper). These mechanisms could be made more efficient and robust by employing computer vision techniques, and Java's OpenCV Library should be helpful in that.

Although the detector detects perfectly filled circles, it needs further work to expand its reach. Perhaps the parameters of the Hough Transform could be loosened to detect less perfect circles. Previous attempts at this have resulted in non-circular shapes on the page being detected as circles. For example, the letters 'e' and 'o' in the questions have been detected as circles. This could be eliminated by limiting the area of search to a vertical strip confined to a few pixels wider than the column of all circles.

The problem of not detecting inconsistently filled or slightly off-shape circles could be remedied in future work, by a combination of the manual circle-of-pixels scan (mentioned before) and Hough Transform. The Hough Transform could be used to detect some of the circles, and the position of all circles on the page could be extrapolated from there (by arithmetic and comparison). Finding the average value of all those circles of pixels and comparing it to a suitable threshold would let us know the attempt and what confidence it's been detected with, and comparing it with the saved answer would tell us whether it's correct or not.

7. References

[1] Origins and Purposes of Multiple Choice Tests, The Center for Teaching and Faculty Development, San Francisco State University, Retrieved from *http://ctfd.sfsu.edu/teaching-practices/origins-and-purposes-of-multiple-choice-tests*, 2018-10-25.

[2] Multiple Choice, Wikipedia, Retrieved from *https://en.wikipedia.org/wiki/Multiple_choice*, 2018-10-25.

[3] Top 10 quiz makers for teachers and educators, DigitalChalk, Retrieved from *https://www.digitalchalk.com/blog/top-10-quiz-makers-teachers-educators*, 2018-10-25.

[4] Tools for teachers, My e-Learning World, Retrieved from *https://myelearningworld.com/top-10-free-online-quiz-makers-for-teachers-and-educators*, 2018-10-25.

[5] Optical Mark Recognition, Wikipedia, Retrieved from *https://en.wikipedia.org/wiki/Optical_mark_recognition,* 2018-10-25.

[6] YAML, Wikipedia, Retrieved from *https://en.wikipedia.org/wiki/YAML*, 2018-10-25.

[7] Computer Vision, Wikipedia, Retrieved from *https://en.wikipedia.org/wiki/Computer_vision*, 2018-10-25.

[8] The Hough Transform, AIShack, Retrieved from *http://aishack.in/tutorials/hough-transform-basics*, 2018-10-25.

[9] Understanding Hough Transform with Python, Alyssa Quek, Retrieved from *https://alyssaq.github.io/2014/understanding-hough-transform*, 2018-10-25.

[10] The Hough Transform, Matthew Thurley, Luleå Tekniska Universitet, Retrieved from *https://www.ltu.se/cms_fs/1.36192!/e0005e_lecture05_hough_transform.dvi.pdf*, 2018-10-25.

[11] Circle Hough Transform, Wikipedia, Retrieved from *https://en.wikipedia.org/wiki/Circle_Hough_Transform*, 2018-10-25.

[12] Hough Transform, Wikipedia, Retrieved from *https://en.wikipedia.org/wiki/Hough_transform,* 2018-10-25.

[13] Edge Detection, Patrice Delmas, University of Auckland, Retrieved from *https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Edge%20detection-Sobel_2up.pdf*, 2018-10-25.

[14] The Sobel and Laplacian Edge Detectors, AIShack, Retrieved from *http://www.aishack.in/tutorials/sobel-laplacian-edge-detectors*, 2018-10-25.

[15] Convolution Matrix, Gimp Docs, Retrieved from *https://docs.gimp.org/2.8/en/plug-in-convmatrix.html*, 2018-10-25.

[16] Jonathan Knudsen, 1999, Java 2D Graphics (Chapter 11), O' Reilly and Associates Inc, California USA.

Appendix

A. Project Description

This is a development project that aims to develop a unified multi-choice paper based quiz system. A system currently exists that automates the generation and marking of multi—choice quizzes, however, its development is fragmented using a number of different languages and stores information about the quizzes in number of different files using different formats. Also the layout of the quizzes are basically fixed with the positioning of answer circles hard coded. This project would completely re-implement the quiz application all within a single framework, this would unify the storage of questions with the generation of pdfs for printing along with the marking and recording of marks all into one single system. This would enable a more flexible layout and generation of questions. It is also hoped that it would be a project that could be used in different contexts with more ease.

The project would involve:

- Researching and writing a review of different approaches currently used for a multiple choice questions.

- Provide background to: the computer vision approaches used, the yml format, and pdf library used.

- Design and build a Java based system that, uses yml files for storing question data, and can generate pdfs based on these questions. The questions and answer order would be randomized.

- Once the answers are scanned the system could semi-automate the marking, entering of marks, and emailing students results. This will

involve tracking the location of answer circles between different quiz sheets.

- The GUI should be easy to use.

- The accuracy of the automatic marking should be evaluated.

The report should include:

- Introduction – gives an overview of the project, provides a summary and the current system and limitations, an overview of other similar currently available systems, and a statement of contributions made in the project.

- Background - the computer vision approaches used, the yml format, and pdf library used.

- Design - this should include requirements, class structure, the GUI design, and data file formats spec.

- Implementation – Structure of implementation, summary of tools and libraries used, configuration description.

- Evaluation – Would describe how the system verification was done. A testing summary, An evaluation of the GUI. Along with some testing done on the accuracy of the auto marking.

- Conclusion

- Appendix – User manual

B. Study Contract

Australian National University

INDEPENDENT STUDY CONTRACT PROJECTS

Note: Enrolment is subject to approval by the course convenor

SECTION A (Students and Supervisors)

UniID: _____u6010425_____

SURNAME: ____Sinha_____ FIRST NAMES: _____Aditya_____

PROJECT SUPERVISOR (*may be external*): _____Eric McCreath_____

FORMAL SUPERVISOR (*if different, must be an RSSCS academic*): ____As above_____

COURSE CODE, TITLE AND UNITS: ___COMP4560 -Advanced Computing Project_____

SEMESTER ☒ S1 ☐ S2 YEAR: __Summer Session 2017/2018 and S1 2018 _____

PROJECT TITLE:

Mutliple Choice Student Quiz System

LEARNING OBJECTIVES:

The student will strengthen their Java programming abilities with a focus on: GUI application development, computer vision, and xml/yml processing. More generally the project would strengthen general programming and problem solving abilities along with research skills associated with exploring approaches and ideas and then implementing, testing and evaluating these approaches and ideas. Also it is expected that the student would gain general skills relating to: writing a report, and giving a seminar.

PROJECT DESCRIPTION:

This is a development project that aims to develop a unified multi-choice paper based quiz system. A system currently exists that automates the generation and marking of multi—choice quizes, however, its development is fragmented using a number of different languages and stores information about the quizes in number of different files using different formats. Also the layout of the quizes are basically fixed with the positioning of answer circles hard coded. This project would completely re-implement the quiz application all within a single framework, this would unify the storage of questions with the

generation of pdfs for printing along with the marking and recording of marks all into one single system. This would enable a more flexible layout and generation of questions. It is also hoped that it would be a project that could be used in different contexts with more ease.

The project would involve:

- Researching and writing a review of different approaches currently used for a multiple choice questions.
- Provide background to: the computer vision approaches used, the yml format, and pdf library used.
- Design and build a Java based system that, uses yml files for storing question data, and can generate pdfs based on these questions. The questions and answer order would be randomized.
- Once the answers are scanned the system could semi-automate the marking, entering of marks, and emailing students results. This will involve tracking the location of answer circles between different quiz sheets.
- The GUI should be easy to use.
- The accuracy of the automatic marking should be evaluated.

The report should include:

- Introduction – gives an overview of the project,
  provides a summary and the current system and limitations,
  an overview of other similar currently available systems, and
  a statement of contributions made in the project.
- Background - the computer vision approaches used, the yml format, and pdf library used.
- Design - this should include requirements, class structure, the GUI design, and data file formats spec.
- Implementation – Structure of implementation, summary of tools and libraries used, configuration description.
- Evaluation – Would describe how the system verification was done. A testing summary, An evaluation of the GUI. Along with some testing done on the accuracy of the auto marking.
- Conclusion
- Appendix – User manual

ASSESSMENT (as per the project course's rules web page, with the differences noted below.

Assessed project components:	**% of mark**	**Due date**	**Evaluated by:**
Report: name style: _____ (e.g. research report, software description..., no less than 45% weight assigned)	45%		(examiner)

Australian National University

Artefact: name kind: _____ (e.g. software, user interface, robot..., no more than 45% weight assigned)	45%		(supervisor)
Presentation:	10%		(course convenor)

MEETING DATES (IF KNOWN):
Weekly.

STUDENT DECLARATION: I agree to fulfil the above defined contract:

…….. ……..
Signature Date

SECTION B (Supervisor):

I am willing to supervise and support this project. I have checked the student's academic record and believe this student can complete the project. I nominate the following examiner, and have obtained their consent to review the report (via signature or attached email)

……..……..…….. ……..
Signature Date

Examiner:

Name: …… Signature ……
(Nominated examiners may be subject to change on request by the supervisor or course convenor)

REQUIRED DEPARTMENT RESOURCES:

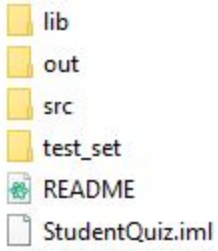
none

SECTION C (Course convenor approval)

Research School of Computer Science *Form updated July 2017*

.....	.....
Signature	Date

C. Artefacts produced



The code is stored in file Main inside folder src.

D. User Guide

Except in places where explicitly mentioned the code, and the artefact is my own work. The development was done on a Windows 10 - 64 bit system, on IntelliJ IDE and so was the testing. The following is a brief guide on using the system.

1) Check the dependencies. Some may be absent owing to the size requirements for the submission. The list of all required dependencies, alongside the ones omitted are listed in the README file.

2) Once the dependencies are all gathered. Execute the java file. This should make the program open up.

3) Click on File Menu.

4) Select an option that you wish to use.

5) Example: Select Generate New Quiz to generate a new pdf quiz.

6) Navigate to the directory that contains your yaml questions.

7) Select the yaml file.

8) Enter a desired name for your resultant pdf file.

9) If the file was created successfully, you will be notified by a dialog box.

10) This newly generated file can be found in the same directory as the yaml.

Alternatively select Mark Scanned Quiz. Select your scan, and the processed, validated image will be displayed in the GUI window with pink rings around the detected answers.

E. Code snippet for detecting filled circles

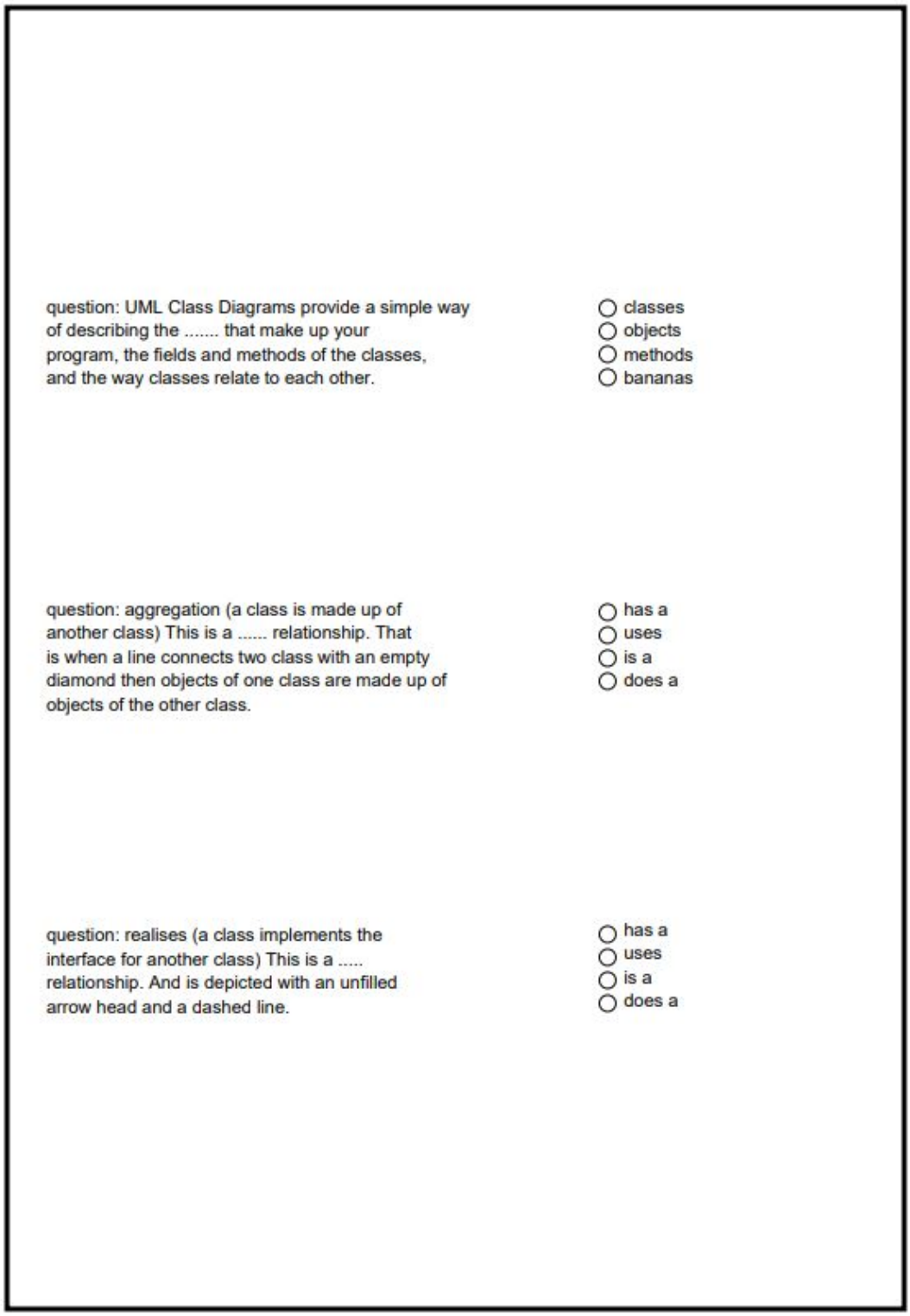
Initially proposed code that scans vertically downwards finding average value of a circle of pixels at each step

```
        ArrayList<Integer> pixels = new ArrayList<>();
        Integer sum = 0;
        int r = 10;
        for(int x = ap_2; )
        int x = 40; // x-coordinate of the centre of the circle
        int y = 500; // y-coordinate of the centre of the circle
        int x = (int) (distance_ratio * cropped.getWidth());
        int y;
        for (y = 10; y < cropped.getHeight() - 10; y += 20) {
            for (i = x - r; i <= x + r; i++)
                for (j = y - r; j <= y + r; j++)
                    if ((i - x) * (i - x) + (j - y) * (j - y) <= r *
r)
                        pixels.add(cropped.getRGB(i, j));
            for (Integer pixel : pixels)
                sum += pixel;
            // System.out.println("Average value for circle at
centre: (" + x + "," + y + "): " + (int) (sum.doubleValue() /
pixels.size()));
            pixels.clear();
            sum = 0;
        }
```

F. Input yaml file

```yaml
questions:
  - question:  UML Class Diagrams provide a simple way of describing the ....
    answers:
      - classes
      - objects
      - methods
      - bananas
    answer: 1
  - question: aggregation (a class is made up of another class) This is a ...
    answers:
      - 'has a'
      - 'uses'
      - 'is a'
      - 'does a'
    answer: 1
  - question: realises (a class implements the interface for another class) Tl
    answers:
      - 'has a'
      - 'uses'
      - 'is a'
      - 'does a'
    answer: 4
```

G. Newly generated pdf output



question: UML Class Diagrams provide a simple way of describing the ..... that make up your program, the fields and methods of the classes, and the way classes relate to each other.

- ○ classes
- ○ objects
- ○ methods
- ○ bananas

question: aggregation (a class is made up of another class) This is a ..... relationship. That is when a line connects two class with an empty diamond then objects of one class are made up of objects of the other class.

- ○ has a
- ○ uses
- ○ is a
- ○ does a

question: realises (a class implements the interface for another class) This is a ..... relationship. And is depicted with an unfilled arrow head and a dashed line.

- ○ has a
- ○ uses
- ○ is a
- ○ does a

H. Results from Test_Scan - 1

question: UML Class Diagrams provide a simple way of describing the ..... that make up your program, the fields and methods of the classes, and the way classes relate to each other.

- (●) classes
- ( ) objects
- ( ) methods
- ( ) bananas

question: aggregation (a class is made up of another class) This is a ..... relationship. That is when a line connects two class with an empty diamond then objects of one class are made up of objects of the other class.

- ( ) has a
- ( ) uses
- (●) is a
- ( ) does a

question: realises (a class implements the interface for another class) This is a ..... relationship. And is depicted with an unfilled arrow head and a dashed line.

- ( ) has a
- (●) uses
- ( ) is a
- ( ) does a

I. Results from Test_Scan - 2

question: UML Class Diagrams provide a simple way of describing the ..... that make up your program, the fields and methods of the classes, and the way classes relate to each other.

- ○ classes
- ○ objects
- ● methods
- ○ bananas

question: aggregation (a class is made up of another class) This is a ..... relationship. That is when a line connects two class with an empty diamond then objects of one class are made up of objects of the other class.

- ● has a
- ○ uses
- ○ is a
- ○ does a

question: realises (a class implements the interface for another class) This is a ..... relationship. And is depicted with an unfilled arrow head and a dashed line.

- ○ has a
- ○ uses
- ○ is a
- ● does a

J. Results from Test_Scan - 3

question: UML Class Diagrams provide a simple way of describing the ..... that make up your program, the fields and methods of the classes, and the way classes relate to each other.

- (●) classes
- ( ) objects
- ( ) methods
- ( ) bananas

question: aggregation (a class is made up of another class) This is a ..... relationship. That is when a line connects two class with an empty diamond then objects of one class are made up of objects of the other class.

- (●) has a
- ( ) uses
- ( ) is a
- ( ) does a

question: realises (a class implements the interface for another class) This is a ..... relationship. And is depicted with an unfilled arrow head and a dashed line.

- ( ) has a
- ( ) uses
- ( ) is a
- (●) does a

K. Results from Test_Scan - 4

question: UML Class Diagrams provide a simple way of describing the ..... that make up your program, the fields and methods of the classes, and the way classes relate to each other.

- classes
- objects
- methods
- bananas

question: aggregation (a class is made up of another class) This is a ..... relationship. That is when a line connects two class with an empty diamond then objects of one class are made up of objects of the other class.

- has a
- uses
- is a
- does a

question: realises (a class implements the interface for another class) This is a ..... relationship. And is depicted with an unfilled arrow head and a dashed line.

- has a
- uses
- is a
- does a

L. Results from Test_Scan - 5

question: UML Class Diagrams provide a simple way of describing the ..... that make up your program, the fields and methods of the classes, and the way classes relate to each other.

- (●) classes
- ( ) objects
- ( ) methods
- ( ) bananas

question: aggregation (a class is made up of another class) This is a ..... relationship. That is when a line connects two class with an empty diamond then objects of one class are made up of objects of the other class.

- (●) has a
- ( ) uses
- ( ) is a
- ( ) does a

question: realises (a class implements the interface for another class) This is a ..... relationship. And is depicted with an unfilled arrow head and a dashed line.

- ( ) has a
- ( ) uses
- ( ) is a
- (●) does a

M. Results from Test_Scan - 6

question: UML Class Diagrams provide a simple way of describing the ..... that make up your program, the fields and methods of the classes, and the way classes relate to each other.

- classes
- objects
- methods
- bananas

question: aggregation (a class is made up of another class) This is a ..... relationship. That is when a line connects two class with an empty diamond then objects of one class are made up of objects of the other class.

- has a
- uses
- is a
- does a

question: realises (a class implements the interface for another class) This is a ..... relationship. And is depicted with an unfilled arrow head and a dashed line.

- has a
- uses
- is a
- does a