King Edward VI Southampton

A Level Coursework

Physics Revision Program

Tom Eaton

supervised by
Mr. Allen and Mr. Mapstone

14th February 2017

Dedicated to Alex Jones, the prophet of reality.

Contents

List of Figures

List of Algorithms

Chapter 1

Analysis

1.1 Abstract

As I am an A Level Maths (with mechanics) and Physics student, I have studied the topics of projectile motion and radioactive decay. I myself found these topics to be difficult as did many of my peers. The current method of practising mathematics and physics questions is to attempt problems from a course textbook. As there are many questions for the student to try, they can seem repetitive or boring causing the student to be more likely to stop. This can be solved by providing a range of questions to provide variety. The majority of problems are provided with no diagram, so the student has to interpret the text and potentially draw their own diagram. While this type of question is likely to come up in the exam, when the student is first learning the topic, it is important that they understand the topic fully before attempting harder, exam style questions. This can be solved by providing diagrams for the student. This allows the question text to be represented visually which make the problem easier to understand, and it will allow the student to relate the question to the underlying concepts that they are learning.

This software will be an easy to use platform. It will generate questions from the projectile motion and radioactive decay topics, and will draw graphs to go with these. The main aim of the software is to be a learning tool, to help students understand the basics of these topics.

1.2 Features of the proposed solution that can be solved using computational methods

1.2.1 Random nature of questions

The questions will be randomly generated. The base question will be stored so that there are elements within it that can be changed. This allows a loop to be created, which will produce n amount of questions, with the elements being

changed to a random number from a specified range. The generated elements can then be used to create the diagram or graph, by putting these elements into the diagram generation function.

1.2.2 Answer Verification

The number of significant figures in the student's answer will not matter. If the student's answer is the same as the calculated answer to any number of significant figures it will be marked as correct. Computationally, this can be implemented by using built in **round** functions, meaning that the mathematics behind this does not have to be coded from scratch. This will be faster than if this was implemented non-computationally.

1.2.3 Generation of diagrams

For both projectile motion and radioactive decay, graphs can be generated using an iterative method. This works by supplying an equation with a variable that can be incremented. This is usually time t. The equation then outputs a value that plotted as the y coordinate on a graph, with the x coordinate being the time value inputted into the equation. This solution will enable graphs of projectile motion and radioactive decay to be quickly generated for display.

1.3 Stakeholders

Students studying A Level physics are the main target users of this program. It will offer an unlimited supply of questions to practice outside of the textbook. The questions will be provided with a graph or diagram which could make the question seem easier to some. This is intended, as the program is meant to aid understanding of the fundamentals of these topics. A Level physics teachers are another target group. They will be able to use this tool in initial lessons to demonstrate how to answer these type of questions. The generated graph will enable easy explanation of the question, and will save time for the teacher. Once the basics have been covered, teachers could task students to complete questions on the application.

1.4 Existing Solutions

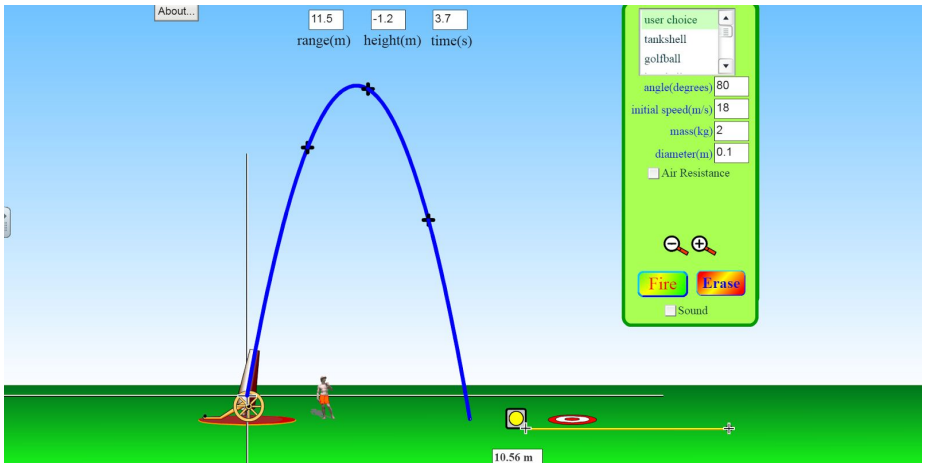

Figure 1.1: PHET Interactive projectile motion simulator. [1]

This piece of software display the path of an object fired out of a cannon. You can modify the mass and diameter of the object being fired. This will only change the motion of the object if air resistance is being simulated. This simulation does have an option to enable air resistance, which will provide a more realistic simulation. However air resistance is not taken into account in the A Level Physics exam, so this feature will be left out. The launch speed and angle can also be changed. The graph generation should have changeable parameters, so that the values in the question can be used to produce a graph of itself. The user won't be allowed to directly change these values, as this is a question generation program, not a sandbox.

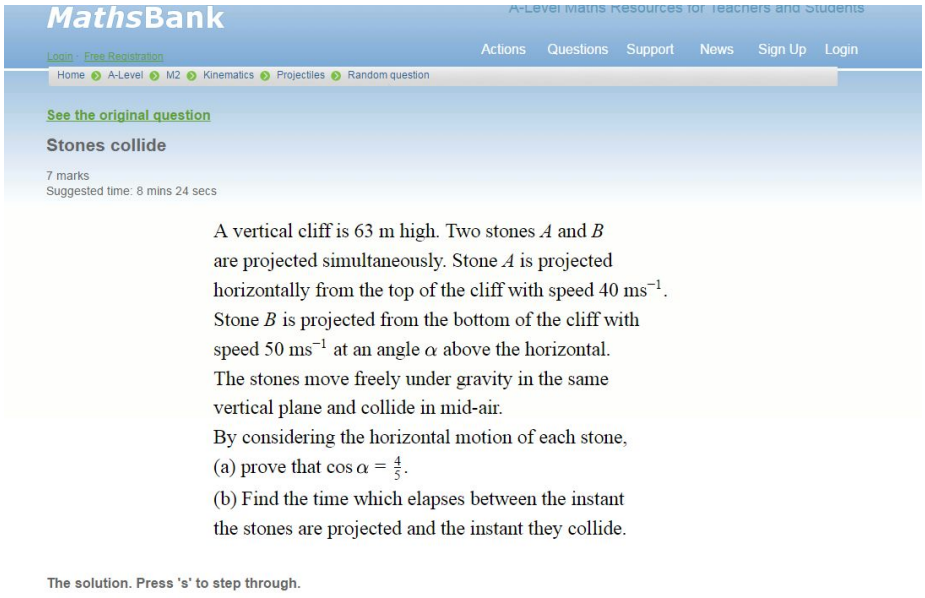

Figure 1.2: MathsBank M2 Projectile motion question bank. [2]

This is an example of an exam style question. The question provides good context, making it easy to understand. A diagram is not provided, as it is expected that the student draws one for themselves. This type of question is not aimed at people just starting the topic, so the style of the question would have to be changed so that all of the information required is displayed obviously, rather than it being hidden in the text.

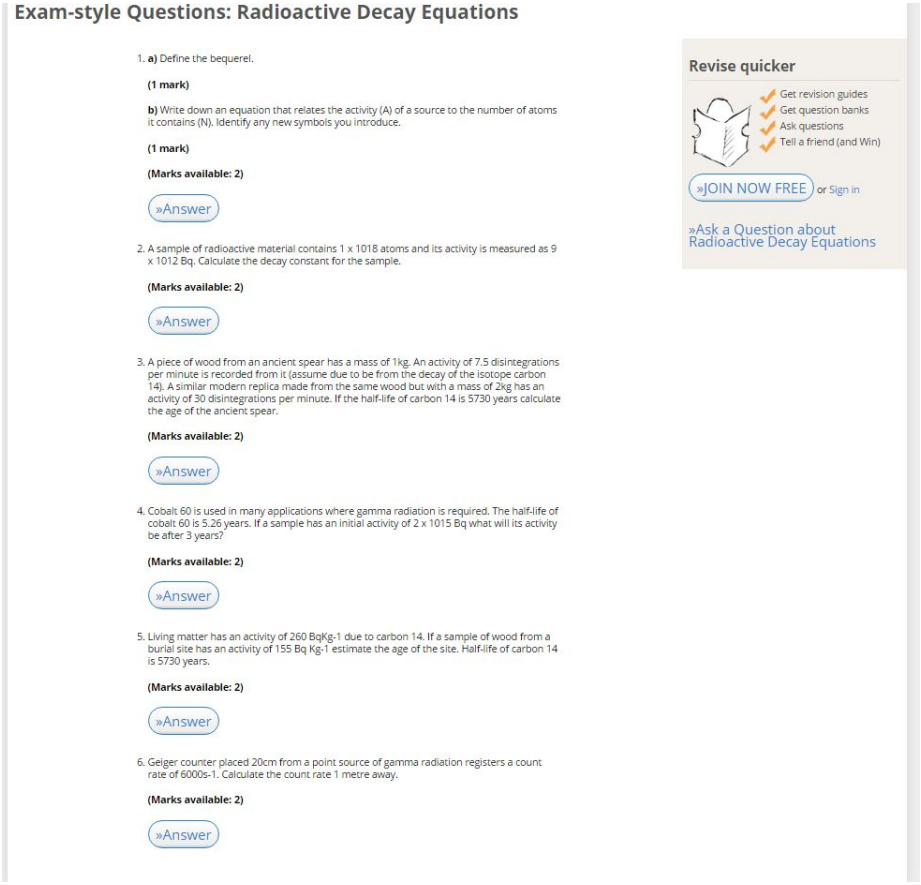

Figure 1.3: Exam-style Questions: Radioactive Decay Equations. [3]

These questions are extensive, and with a realistic context, making it more approachable to the student. Some of the questions are not suitable for random variation like Question 1, which are purely fact recall questions. The style of the other questions are ideal, as they are not too complex, and allow a graph or diagram to be drawn based on the information in the question.

1.5 Solution

1.5.1 Question generation

Projectile motion

The structure of the questions generated will be based on the MathsBank questions seen in Figure 1.2. The length of the question will be much shorter, only containing one part. This will avoid boredom from similar questions, and will

avoid problems with the student having to remember answers to previous parts in order to correctly solve the next part. Figure 1.4 shows an example question,

A ball is projected with speed x ms^{-1}. At the starting point the ball is S m off the ground. The highest point of the ball is y m.

Figure 1.4: Example projectile motion question.

which is the style that will be generated for the user. The parameters which will be changed x, S and y are obvious to the user, and no extra calculations are required. All of the information required to answer the question is easily shown in the question.

Radioactive Decay

The questions used in the program will be very similar to the questions found on S-cool[3]. The only changes will be that the fact recall questions will be removed, due to that fact that these can't be truly randomised.

A sample of radioactive material contains n atoms and its activity is measured as x Bq. Calculate the decay constant for the sample.

Figure 1.5: Example radioactive decay question.

Again, this shows how easy it is to randomise the question, with only the letter variables in Figure 1.5 having to be changed.

1.5.2 GUI

The GUI[1] for the two different topics will be very similar, as they both have three distinct elements:

- Question box
- Graph or diagram
- Answer box

The graph or diagram generated will be based on 1.1, but it will be a much simpler version of this. Only two colours will be used, which will reduce confusion, and avoid distraction. There will also be no representation of the object being fired for clarity.

[1] Graphical User Interface.

1.6 Justification

Random question generation was chosen after looking at the questions from MathsBank [2] and SCool [3] it is obvious that there is only a limited amount of questions available, and that after about an hours use there would be no questions left, meaning that it is useless. The choice to randomise the questions therefore seems very logical, as it will extend the time that the program is useful for.

Generating a graph to go along with the question was chosen to make the question seem less intimidating. From looking at Figure 1.2 the question looks very intimidating. This is because diagrams must be drawn by the student, and extra calculations must be made to get information required to answer the question, or it must be inferred from the question. This is shown clearly in Figure 1.2, where the information is not shown directly, for example instead of giving θ they give cos θ which is confusing.

1.7 Essential Features

- Functional GUI Menu.
- Clear distinction between topics to avoid confusion.
- Questions with random variables.
- Simple random variables that are realistic for the question type.
- Verification of student's answer. Answer is accepted to any amount of significant figures.
- Graph or diagram to supplement question.

Functional GUI Menu

This is to make the software easy to use. The user should be able to use the program with no instruction, so it must be intuitive. By designing like this, the user is more focussed on answering the questions, than trying to use the program.

Clear distinction between topics to avoid confusion

This follows on from the functional GUI. A clear distinction will simplify the experience for the user, and will allow the student to target their question practice to a specific topic.

Simple random variables that are realistic for the question type

By making the variables simple, i.e a number with no decimals places, it makes the question easier to read and more like it would be in an exam. If they are realistic, it gives the question more context so is more interesting to the student.

Verification of student's answer

It is important that the student knows whether they got the question right, as it worthless otherwise. The decision to make the student's answer be accepted to any amount of significant figures, is so that they don't get confused if there answer is marked wrong because of an incorrect amount of significant figures.

Graph or Diagram

The graph or diagram is important as it makes the question easier to understand. This is important, as the idea of the program is to provide a platform to practice and understand the basics of the topics.

1.8 Limits of proposed solution

The main issue with this solution is that the questions won't be truly random. This is because only parts of the question are randomised. For example looking at Figure 1.4 and 1.5 only the letter variables are randomised. There will be a small number of base questions, which determine the style of question and this cannot be randomised as it is out of the scope of this project. For the target user of this program, this won't be a problem. They won't be using the program for too long, as they will learn the basics and then move on to harder questions. In the time that it takes to do that, they won't have time to just learn how to answer the specific type of question, instead of learning the theory behind the question.

1.9 Project Requirements

1.9.1 Programming language

Python 3.5 will be the main project requirement as it is the programming language that the software will be written in. This language is being used because of the great number of libraries available to make the development process easier, and because it is portable, so can run on any modern computer.

1.9.2 GUI

To create the GUI, PyQt will be used, which is a python wrapper for QT. Qt is a cross-platform application framework that is used for developing application software that can be run on various software and hardware platforms with little or no change in the underlying codebase[4].This library will allow portability, and the code won't have to be rewritten due to it being a Python wrapper. Qt comes with a WYSIWYG[1] editor called QtCreator, which is similar to the Visual Basic form designer. A GUI can be easily created using this, as it can be made

[1]What you see is what you get.

visually, instead of having to make it in code, which is time consuming and can be hard to get right. Once the GUI is made in QtCreator, PyUIC5 can be run to convert the Qt code into Python code which can be used to display the GUI.

1.9.3 Graph generation

Matplotlib[1] will be used to generate the graph and diagram for display on the GUI. It is easy to use, and executes quickly, which is important to reduce waiting time for the user. The matplotlib window will not be shown directly to the user, matplotlib will be called to generate an image of the graph which will then be displayed. This is to make sure that there is no issues with graph size.

1.9.4 Hardware requirements

The only requirement is a relatively modern PC, that is running a contemporary operating system. Greater than 2 gigabytes of RAM, and a CPU clock speed greater than 2Ghz is recommended so the questions are displayed promptly.

1.9.5 Final Build

The final build of the software will have a .exe file that will be run by the user. The Python libraries still need to be available to the program, so they need to be installed on the computer. This can be done in two ways, either by installing the Anaconda Python distribution, or providing a virtualenv[2] with the libraries installed. A virtualenv would be easier for the user, as they don't have to install any additional software.

1.10 Success Criteria

- It should have a functional, easy to use GUI.
- It will have randomly generated questions
- It should have realistic values in the question
- Random question should be shown to the user within a second of page display
- Graph or diagram will be generated in relation to question
- Graph should be displayed within a second
- Student's answers should be verified no matter the amount of significant figures.

[1] Python Library used for plotting graphs.
[2] An isolated Python environment

Chapter 2

Design

2.1 Design Overview

The whole project can be broken down into three parts

- Generation of question.
- Generation of graph / diagram.
- Display of GUI.

These subsections can then be broken down even further.

2.1.1 Generation of question

This can be broken down to:

- Loading question store from text file
- Finding what the question type is
- Generating random variables based on parameters in question
- Adding random variables to question string

To be able to generate the questions randomly on the fly, a bare question structure must be implemented. To create this question structure, questions found in Figure 1.2 for example, are analysed to find the variables that are important to the question. These are highlighted in Figure 1.4 as the lettered variables. Once these variables are found, the question is written in a text file in a way that the variables can be formatted to what is required.

2.1.2 Generation of graph / diagram

This can be broken down to:

- Getting random variables from question generator
- Getting question type from question generator
- Generating data needed to draw graph
- Drawing the graph
- Converting the graph to an image and storing it

Matplotlib will be used to generate the graph as discussed earlier. It will require an equation to be able to do this. The equation could be stored in the question in the text file, but it would be easier to just store the question type in the text file. You could then code specific equations depending on the question type. This will prevent the text file from getting too long and hard to read. It will also improve runtime performance, as it will be a shorter and less complex string to parse. The graph will be generated as an image.

2.1.3 Display of GUI

This can be broken down to:

- Displaying main menu
- Allowing the user to choose topics
- Getting question from question generator
- Getting graph from image store
- Displaying question
- Displaying graph
- Taking and verifying user input for answer
- Informing user if they were correct

The display of the GUI will take the generated question, and the image of the graph and collate them to show the GUI. Apart from this, the GUI needs to be able to take a user answer, and check if it is correct. It will need to provide buttons to select a topic, submit an answer and skip a question.

2.2 Dataflow diagrams

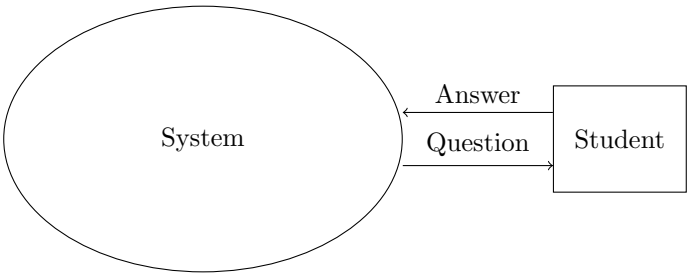

Figure 2.1: Level 0 Dataflow Diagram

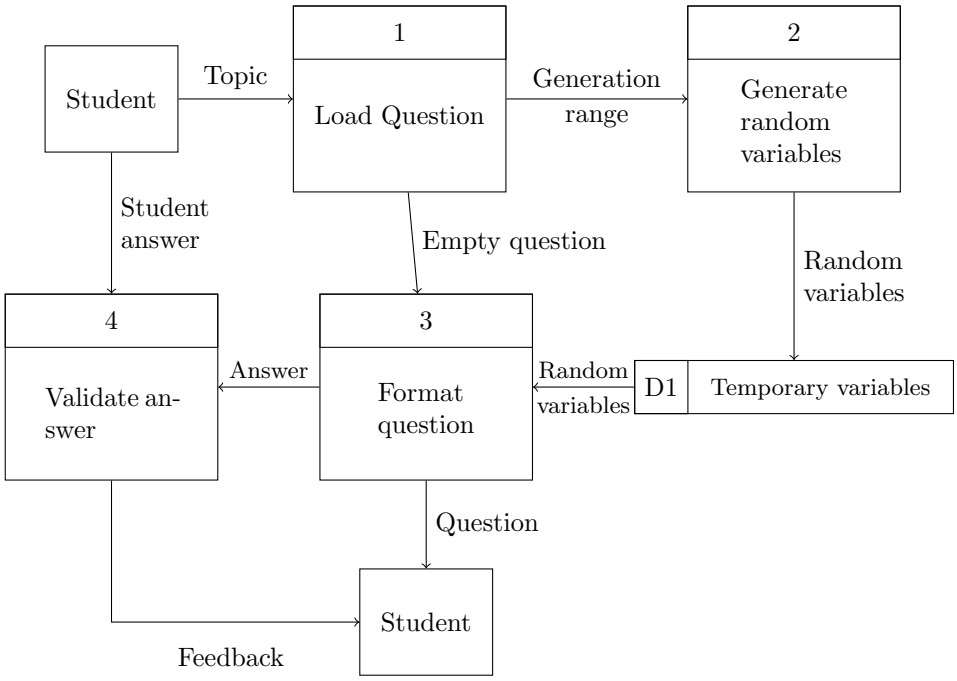

Figure 2.2: Level 1 Dataflow Diagram

Figure 2.2 shows the main flow of the program. As you can see, most of the code will be written to perform the generation of the question, as this is the most complex part.

2.3 Algorithms

N.B In all of these algorithms the classes defined in Section 2.5 will be referenced.

Algorithm 1 Main Algorithm

```
1: Display MainMenu
2: if Projectile topic button pressed then
3:     new ProjectileQuestion
4:     loadQuestion from projectile.txt
5:     ProjectileQuestion(loadQuestion)
6:     if RadioactiveQuestion.answer = student answer then
7:         print "You got it right"
8:         ProjectileQuestion(loadQuestion)
9:     else
10:        print "You got it wrong"
11:        ask "Would you like to see the answer"
12:        if yes then
13:            print ProjectileQuestion.answer
14:            RadioactiveQuestion(loadQuestion)
15:        else
16:            return to function
17:        end if
18:    end if
19: end if
20: if Radioactive decay pressed then
21:    Display RadioactiveQuestion
22:    loadQuestion from radioactive.txt
23:    RadioactiveQuestion(loadQuestion)
24: end if
25: if Close button pressed then
26:    Display PromptBox("Are you sure about that"?)
27:    if Answer = True then
28:        Close application
29:    else
30:        Go to MainMenu
31:    end if
32: end if
```

2.3.1 Justification

This is a simple overview of how the program will run. The only point of interest in this is the fact that when a user gets the question wrong, they are asked whether they would like to see the answer or try again. This was used to prevent

infuriation if the student cannot get the answer, but to also allow students to try again if they think they are close to the answer.

The main algorithm forms a complete solution as it covers all of the input that the user could give, for example wanting to retry the question or to see the answer.

Algorithm run order

The order that this algorithm will run will depend on the run time of each algorithm. For example, the diagram generation may take a long time to execute, so it may be beneficial to generate questions in advance, so that the user does not have to wait a long time when switching question.

Ignoring this, the run order will be this:

1. Let the user choose the topic.
2. Generate the question and graph for that topic.
3. Get the student's answer and check against real answer.
4. If correct go back to step 2
5. If incorrect ask user if they want to skip question or try again.

2.4 Usability Features

Usability features are an important part of this programs success. As the tasks that this program does could be done elsewhere, albeit at the cost of more effort, it is important that the program is as easy to use a possible. This can be resolved by adding key usability features.

The main usability feature is the easy to use GUI. It only has the essential functions, which may mean a small reduction in productivity. However, the GUI will be much more intuitive, and will lead to a more simple experience overall. This is crucial, as ease of use is the main reason that this is a problem.

The feedback on student's answers is another usability feature. If the student gets the answer wrong, they have the choice to either try again, or to see the answer and move onto the next question. The program could just not let you move on if you don't get the right answer, but this could lead to annoyance if they are stuck on a question.

2.5 Datastructures

2.5.1 Classes

Algorithm 2 Randomised

```
1:  Class RANDOMISED
2:      public
3:          Function format
4:          Function get_class
5:          Function get_item
6:          Function get_question_class
7:      endpublic
8:      private
9:          Question_class : Class
10:     endprivate
11: end Class
```

Algorithm 3 Randomised Pseudocode

```
1: Class RANDOMISED
2:     function INIT(self)
3:         self.args ← []
4:         self.question ← None
5:     end function
6:     function FORMAT(string)
7:         formatter ← new RandomisedFormatter
8:         formatter.format(this object, string)
9:     end function
10:    function GETITEM(self, name)
11:        return RandomizedFormatter(name, self.args)
12:    end function
13:    function GETCLASS(self)
14:        if self.args['equation'] = 'findtheta' then
15:            self.question ← ProjectileQuestion(self.args['a'], self.args['b'],
    self.args['c']))
16:        end if
17:        if self.args['equation'] = 'findmaxheight' then
18:            self.question ← ProjectileQuestion(self.args['a'], self.args['b'],
    self.args['c']))
19:        end if
20:        if self.args['equation'] = 'findxdistance' then
21:            self.question ← ProjectileQuestion(self.args['a'], self.args['b'],
    self.args['c']))
22:        end if
23:        if self.args['equation'] = 'findradioactive' then
24:            self.question ← RadioactiveQuestion(self.args['a'], self.args['b'],
    self.args['c']))
25:        end if
26:        if self.args['equation'] = 'findDecayConstant' then
27:            self.question ← RadioactiveQuestion(self.args['a'], self.args['b'],
    self.args['c']))
28:        end if
29:        if self.args['equation'] = 'findParticles' then
30:            self.question ← RadioactiveQuestion(self.args['a'], self.args['b'],
    self.args['c']))
31:        end if
32:    end function
33: end Class
```

Algorithm 4 RandomisedFormatter

```
Class RandomisedFormatter
    public
        Function format
        Function get_name
        Function get_args
    endpublic
    private
        name : String
        args : String Array
    endprivate
end Class
```

Algorithm 5 RandomisedFormatter Pseudocode

```
Class RandomisedFormatter
    function init(self, name, args)
        self.name ← name
        self.args ← args
    end function
    function format(self, fmt)
        op, rest ← fmt.split(':')  ▷ text before ':' into op, text after into rest
        if op == 'type' then
            self.args[self.name] = rest
            return None
        end if
        if op == 'random' then
            low, high = rest.split(':')
            value ← randomNumber(low, high)
            self.args[self.name] ← value
            return string(value)
        end if
    end function
end Class
```

Algorithm 6 ProjectileQuestion

```
1: Class ProjectileQuestion
2:     public
3:         Function calculateProjectile
4:         Function findTheta
5:         Function findXdistance
6:         Function findMaxHeight
7:         Function calculatePoint
8:         Function getTheta
9:         Function getXdistance
10:        Function getMaxHeight
11:    endpublic
12:    private
13:        theta : Integer
14:        xdistance : Float
15:        maxHeight : Float
16:    endprivate
17: end Class
```

Algorithm 7 ProjectileQuestion Pseudocode

```
1:  Class ProjectileQuestion
2:      function init(self, yOffset, initialSpeed, theta)
3:          self.yOffset ← yOffset
4:          self.initalSpeed ← initialSpeed
5:          self.theta ← theta
6:      end function
7:      function calculateProjectile
8:          increment ← 0.01
9:          theta ← radians(theta)
10:         xSpeed ← self.initialSpeed * cos(theta)
11:         ySpeed ← self.initialSpeed * sin(theta)
12:         time ← 0
13:         time ← time + increment
14:         self.xPosArray ← []
15:         self.yPosArray ← []
16:         yPosTemp ← 5                ▷ To satisfy `while` loop during first run
17:         while yPosTemp > 0 do
18:             xPosTemp ← xSpeed × time
19:             yPosTemp ← (ySpeed × time) + (0.5 * -9.8 * time^2 + self.yOffset)
20:             xPosArray.append(xPosTemp)
21:             yPosArray.append(yPosTemp)
22:             time ← time + increment
23:         end while
24:     end function
25:     function findTheta
26:         self.calculateProjectile()
27:         plotAsImg(thetaDiagram, self.xPosArray, self.yPosArray, graph.png)
28:         resize(graph.png, 0.5)
29:         save(graph.png)
30:     end function
31:     function findXDistance
32:         self.calculateProjectile()
33:         plotAsImg(xDistanceDiagram, graph.png)
34:         resize(graph.png, 0.5)
35:         save(graph.png)
36:     end function
37:     function findMaxHeight
38:         self.calculateProjectile()
39:         plotAsImg(maxHeightDiagram, graph.png)
40:         resize(graph.png, 0.5)
41:         save(graph.png)
42:     end function
```

Algorithm 8 ProjectileQuestion Pseudocode continued

```
    function calculatePoint(startX, startY, angle, length)
        endpoint ← [startX + (length × cos(angle)), startY + (length × sin(angle))]
        return endpoint
    end function
    function answerTheta
        return self.theta
    end function
    function answerXDistance
        return max(self.xPosArray)
    end function
    function answerMaxHeight
        return max(self.yPosArray)
    end function
end Class
```

Algorithm 9 RadioactiveQuestion

```
Class RadioactiveQuestion
    public
        Function calculateDecay
        Function findDecayConstant
        Function findHalfLife
        Function findActivity
        Function findParticles
        Function getDecayConstant
        Function getHalfLife
        Function getActivity
        Function getParticles
    endpublic
    private
        decayConstant : Float
        halfLife : Float
        activity : Integer
        particles : Integer
    endprivate
end Class
```

2.5.2 Variables

Variables		
Variable name	**Variable type**	**Comments**
MainApp	Class	Instance of the main menu class
temporaryObject	Class	Question generation class
answer	Integer / float	Answer to the question
studentAnswer	Integer / float	Students answer

2.5.3 Explanation

Algorithm 1 is the class definition for the `Randomised` class. It is used to call the `RandomisedFormatter` class in order to format the question string, and also to return the class.

Algorithm 2 is the pseudocode for the `Randomised`Class. The main bulk of the code is taken up by the getClass Function. This is used to verify

2.5.4 Justification

In Algorithm 2.5.1 and 2.5.1 you can see that the Function names are similar, with there being a find function and a get function for each variable. This is because the find function is used to generate the question and answer, and the get function is used to return the variable. The get function is needed to allow the question to be formatted with data required to answer it.

Classes `Randomised` and `RandomisedFormatter` found in Algorithms 2.5.1 and 2.5.1 respectively, were created using classes as this was required to extend the inbuilt Python string formatter. By default, `str.format()` replaces fields delimited by braces[5].

```
>>> "The sum of 1 + 2 is {0}".format(1+2)
        'The sum of 1 + 2 is 3'
```

Figure 2.3: Default behaviour of Python's `str.format()`[5]

The `RandomisedFormatter` Class extends this functionality, by allowing custom arguments to be specified in the brace delimiter, instead of just a keyword argument, or an index argument.

```
A ball is projected with speed {a:random:20:40}.  At the starting
  point the ball is {b:random:5:30}m of the ground.  The highest
        point of the ball is {equation:type:findtheta}
```

Figure 2.4: Example string for extended string formatter

The syntax in Figure 2.5.4 allows an operand to be specified, and an operator. If necessary, further arguments provided for example a range of values. In the string argument `{a:random:20:40}` `a` is the operand, and this is the name of

the variable in which the random number will be stored. `random` specifies the operation to be performed, in this case generate a random number. Finally `20:40` specifies the range for the random number generation.

The {`equation:type:findtheta`} is less complex, and just allows the equation type to be associated with the string, so the program knows what calculations to perform.

Variables

The variables shown in this section are the main variables that will be used. There are many other variables, but these are mostly covered in the class definitions so are not not worthy. The temporary object is probably the most important variable. This is because as the question generation is random, you need to keep track of the random variables so that you can get an answer from it. This solves the problem by allowing constant access to the class which was instantiated with the random variables, so that you can call methods on that class to get the answer and other things. The `studentAnswer` and `answer` are also important, so that the students answer can be verified.

2.5.5 Validation

The only variable that will require validation is `studentAnswer`. This is because it is user input. The variable will need to be checked that it only contains numbers only. Spaces won't have to be stripped from the variable, as this is automatically done by Python.

2.6 Test data

Test data will be important in determining the order in which the algorithms run, as it is critical that the questions don't take too long to display. The areas to be tested will be shown below.

- Time to generate question
- Time to display graph/diagram
- Generation of variables within question are in specified range
- Graph has a reasonable scale so that the information is easy to read
- Graph image is high enough resolution to be seen clearly

The time to generate question and display graph are being tested so that changes to the algorithm order can be made if necessary. If it takes too long to generate the question, then this will no longer be done at runtime, so that the user doesnt have to wait for the question to be generated. Checking that the variables are within a reasonable range is done so that the questions are realistic. For example it would not be realistic for a sample of material to have 12 atoms in it. Checking

that the graph has reasonable axis is done so that the graph is easy to read, and fits well into the space. It would not be helpful if only a small part of the graph is being shown, and neither would it be helpful if too much of the graph was shown. The right amount of graph to show will be refined during user testing. Testing that the graph image is high enough resolution is done partly to make sure that the image can be seen clearly on the GUI, but it also links back to the time to display graph. If the image resolution is greater, then it will take longer to create, but will be easier to see. During testing I will have to find a balance point between resolution and time to create.

Chapter 3

Development

3.1 Initial testing

3.1.1 Storing questions - Regex

One of the features that the program requires is an ability to generate random questions. An initial idea was to just store the questions in a text file.

```
A ball is projected with speed 20 ms^-1.  At the starting point
the ball is 12m of the ground.  The highest point of the ball is
                        20m.  Find theta.
```

Figure 3.1: Initial question store

The problem with this method is that it is hard to randomise the question. This is because the parts to be randomised in the string are not easily identifiable.

```
A ball is projected with speed [speed].  At the starting point
the ball is [height] of the ground.  The highest point of the
               ball is [highpoint].  Find theta.
```

Figure 3.2: Question store with identifiable variables

Now the variables can be differentiated from other text, this allows a regular expression to be constructed. The regular expression will return text enclosed by brackets.

```
import re
question = "A ball is projected with speed [speed]."
variables = re.findall(r'\[\w+\]', question)
print(variable)
>>> ['[speed]']
```

Figure 3.3: Extracting variables from a string using Python regex

This shows how the variables could be extracted using python code. A random number could then be inserted using `random.randint(range)` function.

There are downsides to using regular expressions. While it is easy to generalise areas of the string that you would like to capture, the expression can get very complex and hard to maintain. I don't think that I have sufficient experience in using regular expressions to be able to confidently use them in this project, so this method was discarded.

3.1.2 Storing questions - String formatting

Python has built in methods to format strings. This is shown in Figure 2.3 found in Section 2.5.4. This functionality is quite basic, and does not allow you to extract the name of the identifiable variable stored in the string (Figure 3.2). This means that it is hard to extract the name of the variable.

Fortunately there is a way to extend the functionality of the inbuilt Python string formatter, so that it can be customised to your needs. More details about that can be found in PEP 3101[6]. This shows that you can create a class, that takes a string as an input, and formats the text however you would like it to.

```
1  class object:
2    def __format__(self, format_spec):
3      return format(str(self), format_spec)
```

Figure 3.4: Example of extended string formatting[6]

This shows how much you can custom the string formatter. An example string for use in this is shown in Figure 2.4.

This method was chosen for use in the project.

3.1.3 Graph generation

There are many Python libraries that allow you to plot graphs and draw diagrams. The most popular library is Matplotlib, which was initially created in 2002 by John Hunter. It allows data to be plotted in a highly customisable way, and interacts very well with Python arrays, which is useful when the data being plotted is generated at runtime.

```
1  import matplotlib.pyplot as plt
2  plt.plot([1,2,3,4])
3  plt.ylabel('some numbers')
4  plt.show()
5
```

Figure 3.5: Code to create Figure 3.6 [7]

This is code used to generate the simple graph seen below.

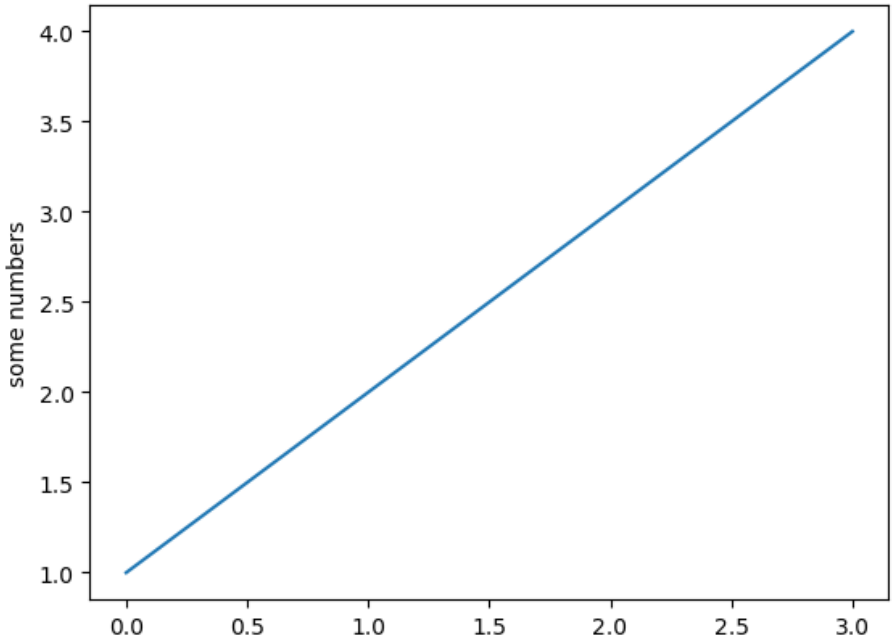

Figure 3.6: Graph generated by code in Figure 3.5

The code in Figure 3.5 shows how easy it is to draw a clean looking graph, which would be suitable for use in the program.

Other Python plotting libraries were not looked at, as Matplotlib is perfect for the needs of the project. It is also part of the Anaconda platform[8] which is a Python installation containing many scientific Python libraries. This library was chosen for use in the project.

3.1.4 GUI

There are hundreds of libraries to display GUI's in Python, but there are three that are most commonly used which are:

- Tkinter
- PyQT
- WxPython

These are all Python wrappers for C/C++ code, which can make development hard because in some cases error codes may not be returned to the Python console, so you don't know what you have done wrong.

WxPython was immediately discounted, as it does not have stable support for Python version 3.x .

Next tkinter was considered. tkinter is a Python wrapper of Tcl/tk. It is class based, so to create a GUI you must create a class that extends the base frame class, and to customise this frame, you must override methods from the parent class. While the code is not that complex itself, the positioning of elements can get very complex, as it must be done entirely in code, as there is no WYSIWYG[1] editor. As this project has a time constraint, tkinter was discarded.

Lastly PyQt was looked at. PyQt is a Python wrapper for Qt, which is a highly popular GUI framework, which is even used for the main touchscreen in the Tesla Model S. Qt comes with an excellent WYSIWYG editor, allowing you to easily create a GUI without having to worry about the code behind it. The code generated however is in C, which would be hard to integrate with the other Python code.

Luckily, there is a program called PyUIC5, which converts C code used to display a Qt GUI, into Python code that performs exactly the same function. This is the perfect tool for this project, as it allows the GUI to be created easily, and functionality to be easily added to the buttons, by extending some classes in the Python code.

Here is how the GUI design workflow will work with PyQt:

1. Design GUI in QtCreator
2. Convert generated C code into Python
3. Add functionality to buttons in Python code

PyQt was chosen as the GUI tool.

3.1.5 Summary

- Extending the Python string formatter for storing questions
- Matplotlib for drawing the graphs
- PyQt for displaying the GUI

[1] What you see is what you get.

3.2 Storing questions

I decided to code each element of the program separately, and then once they are working on their own, to integrate them into the final project. I started writing the question store code first, as it just used vanilla Python, so I didn't have to worry about trying to install libraries.

The code for overriding the Python string formatter was implemented first.

```
#RandomizedFormatter class. Child of Randomized class. Overrides
    String formatter to allow
#question formatting.
class RandomizedFormatter(object):
  def __init__(self, name, args):
    self.name = name
    self.args = args

  #This function overrides the Python string formatter.
  def __format__(self, fmt):
    op, rest = fmt.split(':', 1)

    if op == 'type':
      self.args[self.name] = rest
      return ""
    elif op == 'random':
      low, high = rest.split(':')
      value = random.randint(int(low), int(high))
      self.args[self.name] = value
      return str(value)
```

Listing 3.1: Formatter override

This piece of code overrides the Python string formatter. It then formats the question with this algorithm:

- Find text enclosed by curly braces and place into a list.
- Split the text at the colons.
- Determine what the operator is.
- Format the text according to the operator.
- Store the randomized variable.

The operators available are `type`, and `random`. If the operator is type, then in the questionstore file the variable will look like this {`equation:type:equationtype`}. This allows the program to know which equation to use with the data it is given.

For the `random` operator, the variable will look like this `a:random:5:30` where `a` is the name of the variable that the random value will be stored in, `random` is the operator, and `5:30` is the range for the random number generation.

Now that the code for formatting the string is complete, it was decided to create a parent class which would have other methods. This class would have to be able to return the answer to the question, the values of the randomly generated variables, and also the formatted string. This is the first version of the code for this.

```
class Randomized(object):
  def __init__(self):
    self.args = {}
    self.question = None

  #Returns the value of the randomized variable in the string.
  def __getitem__(self, name):
    return RandomizedFormatter(name, self.args)

  #Formats the string, using the overridden method found in the
    RandomizedFormatter class.
  def format(self, s):
    return string.Formatter().vformat(s, args=(), kwargs=self)

  #Returns the question class depending on the equation type.
  def get_class(self):
    if self.args['equation'] == "findtheta":
      self.question = questionplotclass.ProjectileQuestion(self.
    args['b'], self.args['a'], random.randint(40, 60))
      answer = self.question.answer_theta()
      return answer
    if self.args['equation'] == "findmaxheight":
      self.question = questionplotclass.ProjectileQuestion(self.
    args['c'], self.args['a'], self.args['b'])
      answer = self.question.answer_max_height()
      return answer
    if self.args['equation'] == 'findxdistance':
      self.question = questionplotclass.ProjectileQuestion(self.
    args['c'], self.args['a'], self.args['b'])
      answer = self.question.answer_xdistance
      return answer
```

Listing 3.2: questionStore Parent Class v1

This code couldn't be properly tested, as the `questionPlotClass` hadn't been written yet, so random numbers were given as answers for testing. While this code returns the answer, there is a problem because when the class is initialised, a random number is used, and this is not stored. This means that if another method wants to be called on the class, it will most likely have a different result, as the random number generated will be different.

This problem was solved by instead of just returning the answer, the instance of the class was returned. This meant that any method could be called on this instance of the class, without having to store the random number.

A function was also added to retrieve a random line from a text file of questions. This is the `load` The second version of the `get_class` function:

```
#Returns the question class depending on the equation type.
def get_class(self):
  if self.args['equation'] == "findtheta":
    self.question = questionplotclass.ProjectileQuestion(self.args[
    'b'], self.args['a'], random.randint(40, 60))
    self.question.find_theta()
    return self.question
  if self.args['equation'] == "findmaxheight":
    self.question = questionplotclass.ProjectileQuestion(self.args[
    'c'], self.args['a'], self.args['b'])
```

```
    self.question.find_max_height()
    return self.question
  if self.args['equation'] == 'findxdistance':
    self.question = questionplotclass.ProjectileQuestion(self.args['c'], self.args['a'], self.args['b'])
    self.question.find_xdistance()
    return self.question
```

Listing 3.3: Second iteration of the get_class function

You can see that this code now returns the instance of the class instead of the answer.

Now that both the parent and child class were complete, they were put together to make the final Python file for the question formatting.

```
import random
import string
import questionplotclass

#Randomized class, that is used to format strings, and return question objects.
#This is the parent of the RandomizeddFormatter class.
class Randomized(object):
    def __init__(self):
        self.args = {}
        self.question = None

    #Returns the value of the randomized variable in the string.
    def __getitem__(self, name):
        return RandomizedFormatter(name, self.args)

    #Formats the string, using the overridden method found in the RandomizedFormatter class.
    def format(self, s):
        return string.Formatter().vformat(s, args=(), kwargs=self)

    #Returns the question class depending on the equation type.
    def get_class(self):
        if self.args['equation'] == "findtheta":
            self.question = questionplotclass.ProjectileQuestion(self.args['b'], self.args['a'], random.randint(40, 60))
            self.question.find_theta()
            return self.question
        if self.args['equation'] == "findmaxheight":
            self.question = questionplotclass.ProjectileQuestion(self.args['c'], self.args['a'], self.args['b'])
            self.question.find_max_height()
            return self.question
        if self.args['equation'] == 'findxdistance':
            self.question = questionplotclass.ProjectileQuestion(self.args['c'], self.args['a'], self.args['b'])
            self.question.find_xdistance()
            return self.question

#RandomizedFormatter class. Child of Randomized class. Overrides String formatter to allow
#question formatting.
class RandomizedFormatter(object):
```

```
38      def __init__(self, name, args):
39          self.name = name
40          self.args = args
41
42      #This function overrides the Python string formatter.
43      def __format__(self, fmt):
44          op, rest = fmt.split(':', 1)
45
46          if op == 'type':
47              self.args[self.name] = rest
48              return ""
49          elif op == 'random':
50              low, high = rest.split(':')
51              value = random.randint(int(low), int(high))
52              self.args[self.name] = value
53              return str(value)
54
55
56  def load(questionType, object):
57      lines = [line.rstrip('\n') for line in open(questionType + ".
        txt")]
58      return object.format(random.choice(lines))
59
60  #This code only runs if the file is run specifically, not when it
        is imported.
61  if __name__ == '__main__':
62      rr = Randomized()
63      load("projectilemotionquestions", rr)
64      print(rr.get_class())
```

Listing 3.4: Final questionStore code

Here is a list of features that this piece of code performs:

- Formats a question string by replacing variables with random numbers and storing value.
- Allows the instance of the class used to display the question to be returned so other methods can be called on it.
- Loads a random question string from a text file.

3.3 Graph Generation

After the code for string formatting was completed, next the code for the graph generation was made. This code is linked to the question generation, as it requires the random variables, and the equations used to generate the graph can also be used to create an answer.

3.3.1 Equations

To generate the graphs, a series of coordinates that the particle will pass through need to be calculated. matplotlib will then draw a smooth curve between these

points. To keep things simple, and similar to the A2 Maths and Physics syllabus air resistance is ignored. So the horizontal speed won't change, and can be written as

$$v_x = u\cos\theta$$

where u is the initial speed, and θ is the angle the particle is launched to the horizontal. The position can then be calculated by using distance = speed x time, so the equation for horizontal displacement is

$$S_x = tu\cos\theta$$

The vertical position is a bit harder to calculate as gravity is being simulated. Luckily there is an equation of motion which is perfect for this scenario:

$$S = ut + \frac{1}{2}at^2$$

where t is the time in seconds, and a is the acceleration due to gravity. This equation needs to be tweaked slightly, as we need resolve the initial speed to find the vertical component.

$$v_y = u\sin\theta$$

We can then put this into the equation of motion

$$S_y = ut\sin\theta + \frac{1}{2}at^2$$

Now that we have equations for both the x and y displacement, we can generate of list of coordinates using an algorithm like this: Now that we have

Algorithm 10 Coordinate list generation

1: **for** i in range 0, upperBound **do**
2: tempPos = []
3: tempPos.append($iu\cos\theta$)
4: tempPos.append($S_y = ui\sin\theta + \frac{1}{2}ai^2$)
5: coordinateArray.append(tempPos)
6: **end for**

the pseudocode, this loop needs to be implemented in Python.

```
while (yPosTemp > 0): #While the particle is above the ground
  xPosTemp = xSpeed * time
  yPosTemp = (ySpeed * time) + (0.5 * -9.8 * time ** 2)
  xPos.append(xSpeed * time)
  yPos.append((ySpeed * time) + (0.5 * -9.8 * time ** 2))
  time += increment
```

Listing 3.5: Coordinate generation loop

For this code I used a separate array for the x position and the y position, but that should not matter at this stage. A while loop is used in this case, as I want the graph to only show up to when it hits the ground again after being fired, so the while loop keeps generating coordinates until it is below the ground.

3.3.2 Graph drawing

Using the list of coordinates generated using the code above, we can use matplotlib to display the graph. I started from the bottom, so a simple graph was made first.

```
import matplotlib.pyplot as plt
import numpy as np

increment = 0.0001  # Time increment used when calculating values
initialSpeed = 10
theta = np.radians(70)  # Angle that the direction of launch makes to the horizontal
xSpeed = initialSpeed * np.cos(theta)  # Using trig to calculate xspeed. Assumes there is no air resistance
ySpeed = initialSpeed * np.sin(theta)
xOffset = 0  # Initial x position. Can be changed to add an offset which can be in a question
yOffset = 0  # Initial y position. Can be changed to add an offset which can be in a question
xPos = []  # X position array
yPos = []  # Y position array
time = 0  # Initial time

xPos.append(xOffset)  # Adding the offset in the first slot of the position array
yPos.append(yOffset)  # Adding the offset in the first slot of the position array

time += increment

xPosTemp = xSpeed * time  # Calculating first x value so while loop doesn't fail instantly
yPosTemp = (ySpeed * time) + (
0.5 * -9.8 * time ** 2)  # Using SUVAT to calculate first y value. This is so that the while loop doesn't fail instantly
xPos.append(xPosTemp)
yPos.append(yPosTemp)

while (yPosTemp > 0):
  xPosTemp = xSpeed * time
  yPosTemp = (ySpeed * time) + (0.5 * -9.8 * time ** 2)
  xPos.append(xSpeed * time)
  yPos.append((ySpeed * time) + (0.5 * -9.8 * time ** 2))
  time += increment

plt.plot(xPos, yPos)
plt.axis([0, xPos[len(xPos) - 1], 0, max(yPos)])
plt.xlabel("X displacement")
plt.ylabel("Y displacement")
plt.show()
```

Listing 3.6: First graph generation code

This generates

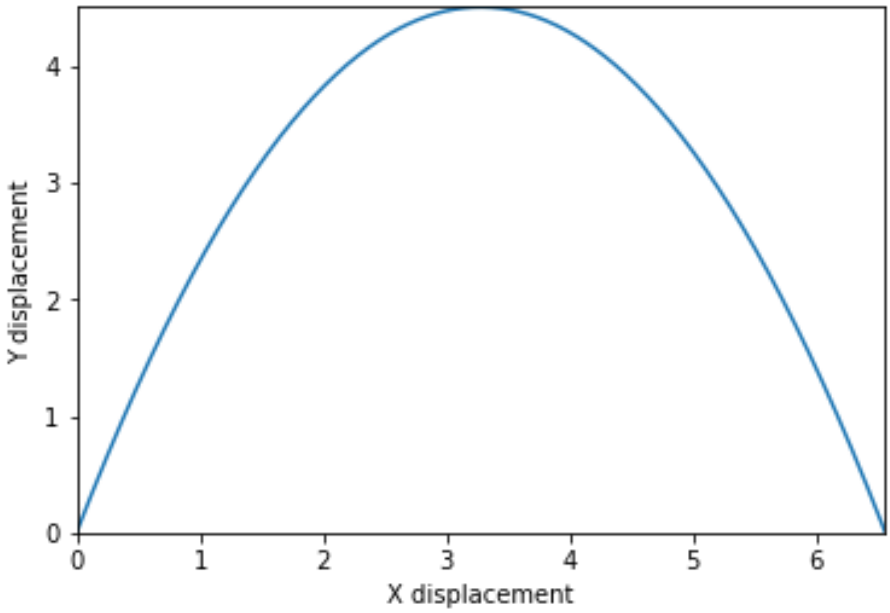

Figure 3.7: First graph

This is perfectly acceptable graph, although it is a bit boring, and the y axis is a bit small, as the top of the graph is cut off slightly. This can be easily fixed by changing line 36 in Listing 3.6 to `plt.axis([0, xPos[len(xPos) - 1], 0, max(yPos) + max(yPos) * .2])`. This will increase the height of the axis by the same amount, no matter the launch parameters of the particle, as it is being multiplied by a constant.

At this point it was decided that questions generated using this graph would be too boring, and simple. So another type of question was chosen similar to this

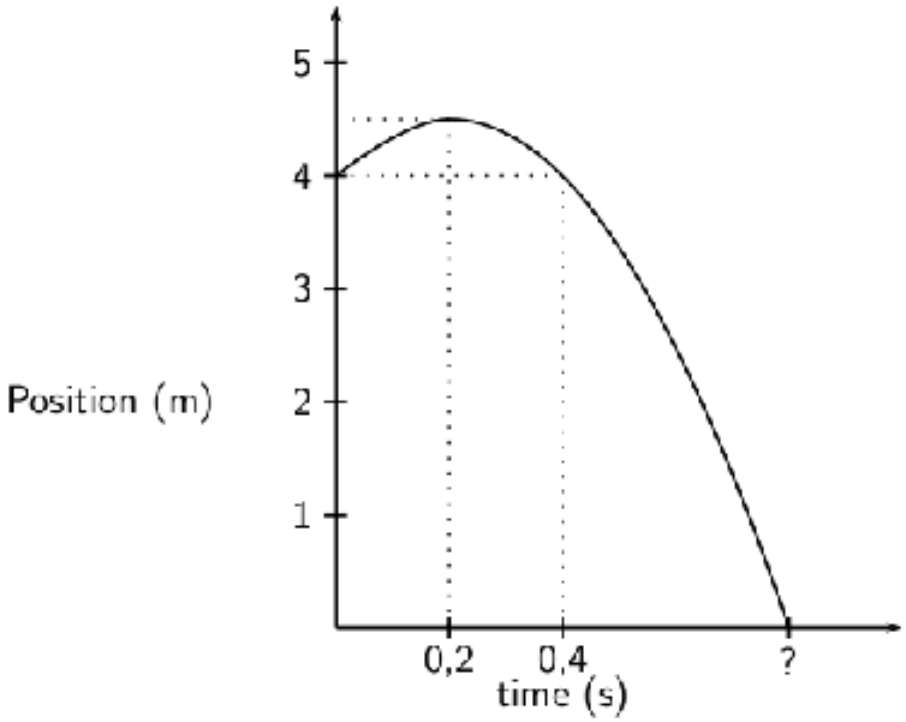

Figure 3.8: Example of a graph [9]

This type of graph allows the initial height offset to be randomised, which can make for a more interesting question, and requires some more complex mathematics to solve.

The next generation of the graph was then coded, which meant that particle started at an offset above the ground. Labels were added for the maximum height, and the distance travelled before it hits the ground.

```
import matplotlib.pyplot as plt
import numpy as np


def y_displacement(y_offset, initial_speed, theta):
  increment = 0.001
  theta = np.radians(theta)
  x_speed = initial_speed * np.cos(theta)
  y_speed = initial_speed * np.sin(theta)
  x_pos = []
  y_pos = []
  time = 0
  x_pos.append(0)

  y_pos.append(y_offset) # Adding the offset in the first slot of
    the position array
```

```
16
17      time += increment
18      x_pos_temp = x_speed * time  # Calculating first x value so while
            loop doesn't fail instantly
19      y_pos_temp = (y_speed * time) + (
20      0.5 * -9.8 * time ** 2) + y_offset
21
22      x_pos.append(x_pos_temp)
23      y_pos.append(y_pos_temp)
24
25      while y_pos_temp > 0:
26          y_pos_temp = (y_speed * time) + (0.5 * -9.8 * time ** 2 + 12)
27          x_pos.append(x_speed * time)
28          y_pos.append((y_speed * time) + (0.5 * -9.8 * time ** 2) + 12)
29          time += increment
30          x_distance = max(x_pos)
31      plt.plot(x_pos, y_pos)
32      plt.annotate(s="", xy=(-1, y_offset), xytext=(-1, 0), arrowprops=
            dict(arrowstyle='<->'))
33      plt.annotate(s="", xy=(0, -1), xytext=(max(x_pos), -1),
            arrowprops=dict(arrowstyle='<->'))
34      plt.plot([0, x_distance], [0, 0], color='k', linestyle='-',
            linewidth=2)
35      plt.plot([0, 0], [0, y_offset], color='k', linestyle='-',
            linewidth=2)
36      plt.plot([-0, -1], [y_offset, y_offset], color='k', linestyle='-'
            , linewidth=2)
37      plt.plot([0, x_pos[y_pos.index(max(y_pos))]], [y_offset, y_offset
            ], color='k', linestyle='--', linewidth=2)
38      plt.annotate(s="", xy=(x_pos[y_pos.index(max(y_pos))], max(y_pos)
            ),
39      xytext=(x_pos[y_pos.index(max(y_pos))], y_offset), arrowprops=
            dict(arrowstyle='<->'))
40      plt.text(-max(x_pos) * 0.08, y_offset / 2, str(y_offset), style='
            normal')
41      plt.text(max(x_pos) / 2, -max(x_pos) * 0.05, str(max(x_pos)),
            style='normal')
42      plt.text(x_pos[y_pos.index(max(y_pos))] + 1, ((max(y_pos) -
            y_offset) / 2) + y_offset, str(max(y_pos) - y_offset),
43      style='normal')
44      plt.axis([-5, x_distance, -5, max(y_pos) + 5])
45      plt.axis('off')
46      plt.show()
47
48  y_displacement(12, 20, 30)
```

Listing 3.7: Second iteration of graph generation

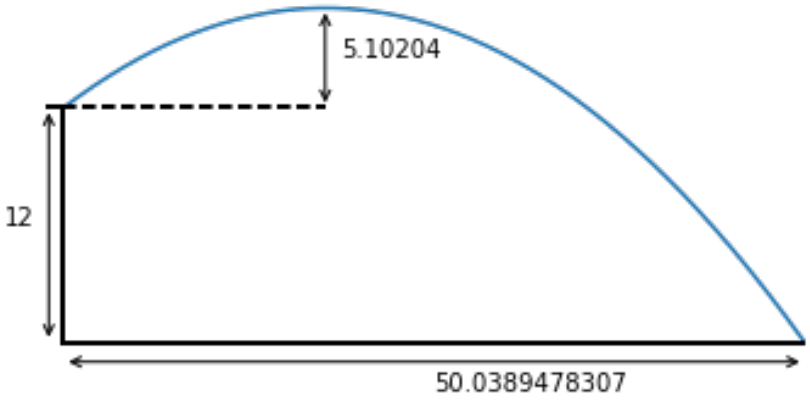

Figure 3.9: Second iteration of graph

As you can see this graph is much more interesting that the previous graph. It shows the data needed to answer questions clearly, and it is easy to randomise the variables in it.

Then questions needed to be designed around this style of graph, so I the questions will be either finding θ, finding the maximum height or finding the distance travelled before it hits the ground. To display these without giving the answer away, I will have to not show some information on the graph depending on the type of question, and also add an indication of which angle θ is on the graph.

First I added an arrow showing the initial launch. To draw an arrow you have to give the function two points. The first point was trivial, it was just (0, y offset) however the second point was more involved as it had to be at the right angle. I used trigonometry to work out this point, and the Python code is shown below.

```
1 def calculate_point(start_x, start_y, angle, length):
2   endpoint = [start_x + (length * np.cos(angle)), start_y + (length
      * np.sin(angle))]
3   return endpoint
```

Listing 3.8: Arrow coordinate generator

When you call the function it will return a list of the two coordinates. This can then be used to generate the arrow. This arrow was added with this line of code `plt.annotate(s="", xy=(0, y_offset), xytext=(calculate_point(0, 12, theta, 10)), arrowprops=dict(arrowstyle='<-'))` The graph now looks like this:

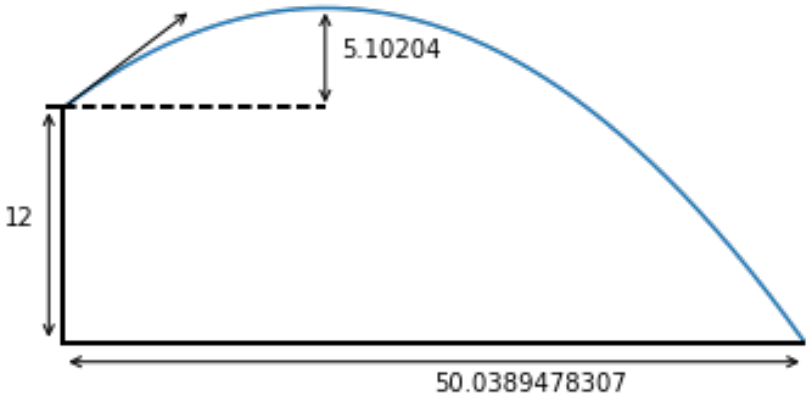

Figure 3.10: Third iteration of graph

Next θ needed to be displayed on the graph. There were three ways to do this, either to draw the symbol manually with a line and an eclipse, use an image of the symbol or use LaTeXto generate the symbol. I decided to use LaTeXas it meant that I could easily use other symbols if necessary.

"LaTeX, which is pronounced Lah-tech or Lay-tech (to rhyme with blech or Bertolt Brecht), is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents but it can be used for almost any form of publishing."[10]. This is also what I am using to create this document.

To use LaTeXwith Python is not trivial. You must first get a working installation of LaTeXon your computer, which is usually done with MikTeX on a Windows machine. Then you must configure matplotlib to use LaTeXwhich is done by adding the lines:

```
1 plt.rc('text', usetex=True)
2 plt.rc('font', family='serif')
```

Listing 3.9: Requirements for LaTeXin matplotlib

Then to use LaTeXto display math characters, you must precede the string with `r`.

```
1 r'\textbf{\ensuremath{\theta}'
```

Listing 3.10: String used to display θ using LaTeX

The symbol was then added to the graph with this line of code `plt.text(0 + initial_speed * 0.09, y_offset + initial_speed * 0.02, r'\textbf{\ensuremath{\theta}')` A label for the initial speed was also added.

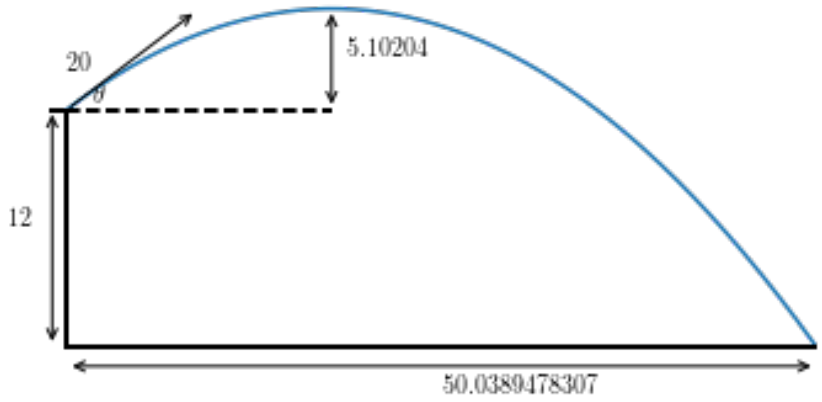

Figure 3.11: Fourth iteration of graph

Finally the angle arc had to be drawn. This was done using Matplotlib patches, which were quite complex. They work essentially like stickers, where you create the item you would like to place, and then you place it at specific coordinates on the graph. The arc was create using this code

```
arc = patches.Arc((0, y_offset), 7, 7,
angle=0, theta1=360, theta2=30, linewidth=1)
```

Listing 3.11: Requirements for LaTeXin matplotlib

This however is hard coded, so the arc will not change when the angle changes. This can be fixed by changing `theta2=30` to `theta2=np.degrees(theta)`.

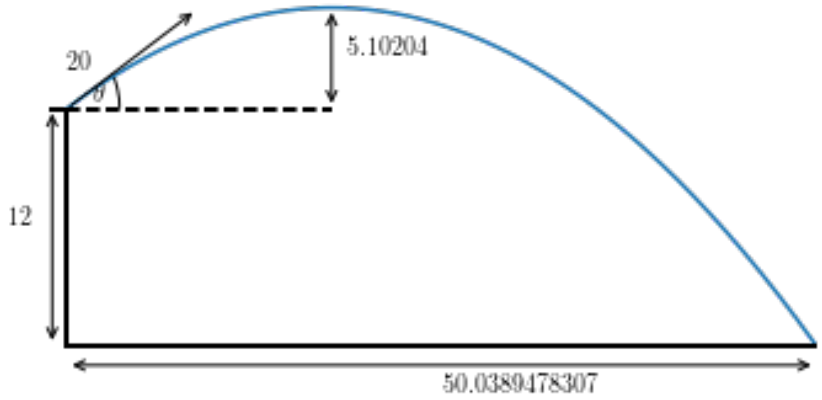

Figure 3.12: Final iteration of graph

At this point I realised that I would not have enough time to complete the radioactive decay part of the questions, so I decided to focus on the projectile motion questions.

3.3.3 Question class

Once the graph generation was coded, this was put into a class to collect the methods together and make it easy to reuse. The Class created can be found in Algorithm 6 and 7 in Section 2.5

3.4 GUI Design

As mentioned before, Qt has a great GUI designer which I will use to create the GUI. Before the GUI was made in QT Creator, the design was prototyped.

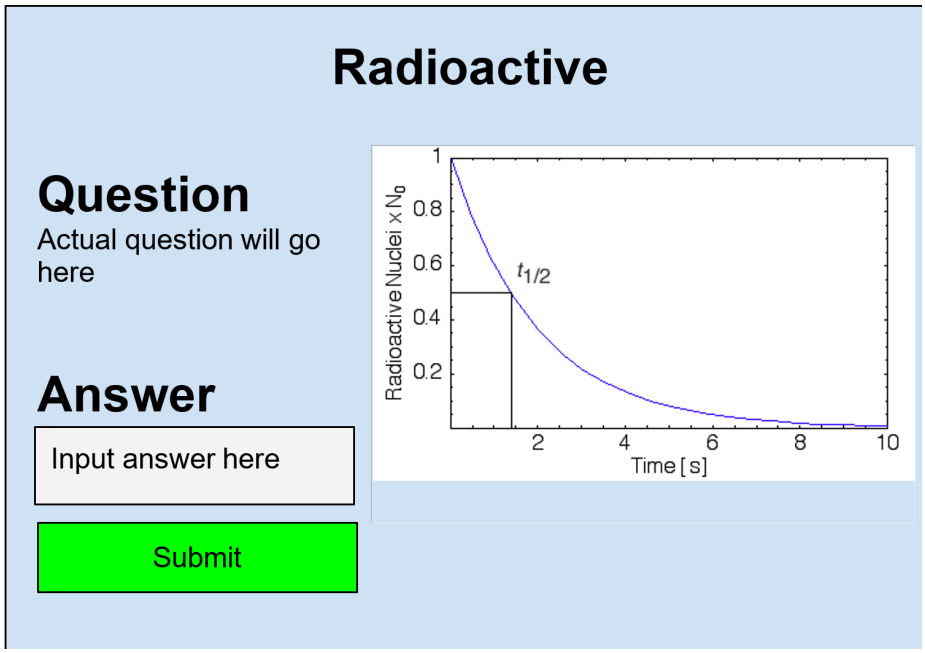

Figure 3.13: GUI Design Prototype

Figure 3.13 is clear and functional. It has large title, so it is clear to the user which topic they are practising.

Now that the prototype was made, the design was then created in Qt Creator. This program was slightly more difficult to use than expected, as the building was not truly drag and drop. A grid had to be created first, and then items put into it with spacers to change their position, otherwise the window just collapses on itself. This was discovered after the first prototype was created and it was then changed after that.

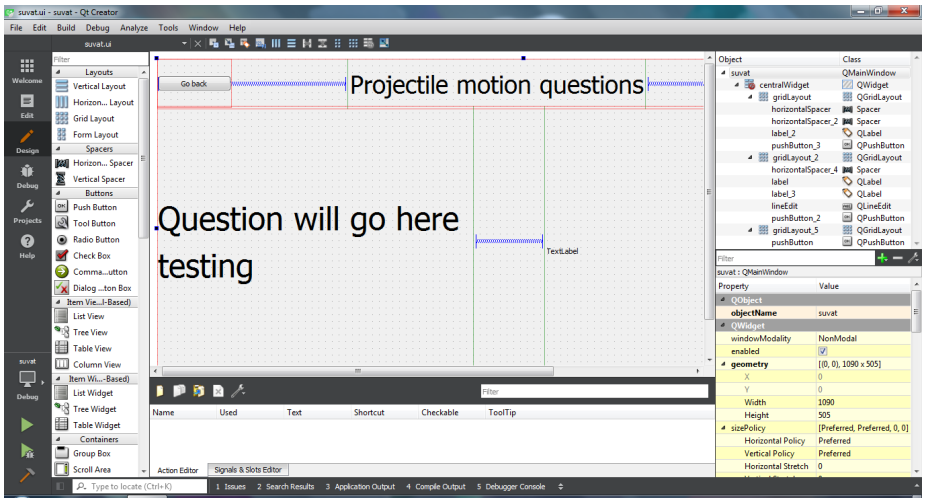

Figure 3.14: Projectile motion screen in Qtcreator

On the right hand side of the image, you can see a list of the components used to make up the screen. The parent component is the grid, which is used so that spacers can be placed, and so that elements will keep their shape.

Looking at the main window, The blue squiggly lines are spacers. These are used to move elements around, as elements can not be dragged and dropped into position, as was found out in the initial test of the prototype.

The green boxes are the areas that contain a label, for example the title "Projectile motion questions". Labels are also used as a placeholder for the image, as there is not an image element. The label is then replaced by an image when the GUI is being prepared for display.

Buttons can also be seen on the screen, in the top left and at the bottom next to the answer text box.

The answer text box at the bottom is a simple textbox to allow the user to enter their answer to the question.

3.4.1 Porting GUI into Python

Now that the GUI was created in Qtcreator, and the Qt code was automatically created by Qtcreator, the Qt code had to be converted into Python code so that the GUI could be interacted with in Python.

Luckily this was not too hard as there is a command line tool called `pyuic`. This allows you to convert the Qt code into Python code with the command `pyuic -o file` where `file` is the filename that the Python code will be exported to.

During the development stage, the GUI design was changed frequently, and it became tiresome having to run the `pyuic` command everytime a change was

made. To solve this problem, a Windows `.bat` file was created so that the command did not have to be written out each time.

```
IF EXIST C:\Users\TomEaton\PycharmProjects\school\projectilegui.py
    del /F C:\Users\TomEaton\PycharmProjects\school\projectilegui.
    py

pyuic5 -o C:\Users\TomEaton\PycharmProjects\school\projectilegui.py
     C:\Users\TomEaton\Documents\suvat\suvat.ui
```

Listing 3.12: Batch script to update Python code

This script checks if `projectilegui.py` file already exists. This file contains the Python code generated by pyuic. If it does exist, it is deleted. This is because when `pyuic` writes to a file, it fails if the file already exists. The next line then runs the `pyuic` command detailed above.

The Python code generated is by no means the most optimal, but the performance gain from writing the code by hand would have minimal benefit compared to the time saved by having the code automatically generated.

```
self.lineEdit.setObjectName("lineEdit")
self.gridLayout_2.addWidget(self.lineEdit, 1, 0, 1, 1)
self.pushButton_2 = QtWidgets.QPushButton(self.gridLayoutWidget_2)
self.pushButton_2.setObjectName("pushButton_2")
self.gridLayout_2.addWidget(self.pushButton_2, 1, 1, 1, 1)
self.label = QtWidgets.QLabel(self.gridLayoutWidget_2)
sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Expanding,
    QtWidgets.QSizePolicy.Preferred)
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(self.label.sizePolicy().
    hasHeightForWidth())
self.label.setSizePolicy(sizePolicy)
self.label.setMinimumSize(QtCore.QSize(500, 0))
self.label.setMaximumSize(QtCore.QSize(10000000, 16777215))
self.label.setObjectName("label")
self.gridLayout_2.addWidget(self.label, 0, 2, 2, 2)
```

Listing 3.13: `PyQt` code generated by `pyuic`

This is an extract of the generated code. As this is only 15 lines out of 100, it is clear that this would take to long to code by hand. Additionally the iterative development cycle for this would be lengthy as each time the code is tested, the Python code would have to been run and it would take a long time.

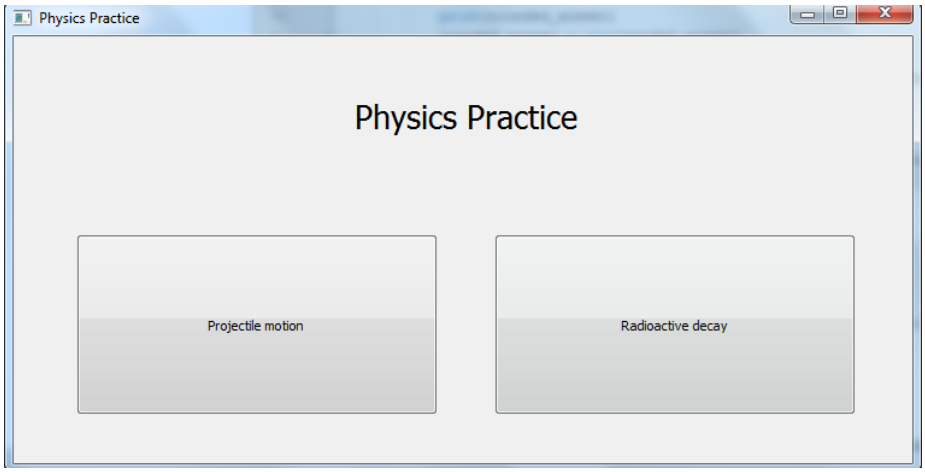

Figure 3.15: Main menu GUI as seen when run in Python

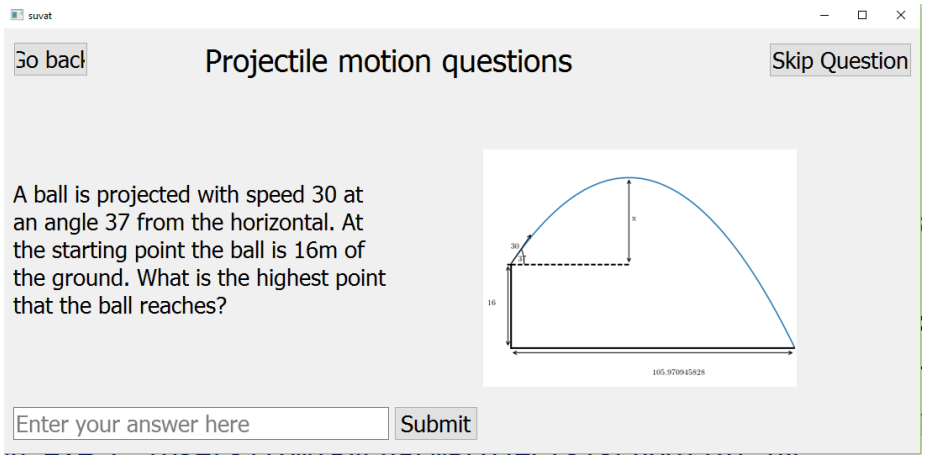

Figure 3.16: Projectile motion question screen as seen when run in Python

3.4.2 Adding functionality to the GUI

Firstly a class for the frame must be made so that the code can extend the main PyQt GUI code. The syntax for this class looks like this

```
class ClassName(QtWidgets.QMainWindow, <UI_FILE>.Ui_main):
    def __init__(self):
    super(self.__class__, self).__init__()
    self.setupUi(self)
```

Listing 3.14: Syntax for Python class declaration

The first line creates the class, passing the PyQt Main window class, and the class containing the window code. `<UI_FILE>` is replaced with the name of the class which contains the code which displays the GUI. In the class initialisation, the first line inherits the methods from the classes passed to the class. The UI is then set up on the second line.

To add functionality to the GUI, PyQt events must be associated with a function inside of the class. This is done like so

```
self.itemName.clicked.connect(self.class_function_name)
```

Listing 3.15: Syntax for linking PyQt event with class function

where `itemName` is the name of the GUI item, and `class_function_name` is the name of the function in the class. `clicked` can be replaced with other events that can happen to an item, for example when the text is changed in a textbox.

Other events that are not item specific can trigger functions by overriding the function in the main PyQt window class. In this project, only the window close event is used.

```
def closeEvent(self):
```

Listing 3.16: Syntax for item exclusive events

The code in this function will then trigger when a window is attempted to be closed.

3.4.3 Main menu

Now that functionality is able to be be added to the GUI, this must now be done. The main menu is the simpler of the two menus, with only two buttons required. These buttons need to take the user to the next screen, which is the question screen.

First the button must be linked to a function. The function will be called `button_clicked`. `self.pushButton.clicked.connect(self.button_clicked)` links the button click to the function `button_clicked`. This function must hide the main menu screen, and then show the question screen. In order to show the question screen, we must provide a link to the Class of this screen so that the `show()` function can be called on it. This is done by `self.suvatApp = SuvatApp(self)`. Then the code for switching screens is trivial.

```
def button_clicked(self):
  self.hide()
  self.suvatApp.show()
```

Listing 3.17: GUI screen switch function

Now that this button is coded, the main menu could be considered complete. However at this stage when the user attempts to close the program, it exits immediately with no confirmation. This could be a problem, as they would lose their progress. This problem can be solved by adding a confirmation popup when they try to close the window. To do this, we must add the function found in Listing 3.16 to our class, and then make a confirmation window appear.

```
def closeEvent(self, event):
  reply = QtWidgets.QMessageBox.question(self, 'Alert', "Are you
    sure about that?",
  QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.No,
  QtWidgets.QMessageBox.No)
  if reply == QtWidgets.QMessageBox.Yes:
    event.accept()
  else:
    event.ignore()
```

Listing 3.18: Window close confirmation function

The `reply` variable is the PyQt message box, containing two buttons and a message asking for confirmation. The conditional logic after this then closes the program if the user replies yes to the popup. Notice that an additional parameter is passed to the `closeEvent` function. This `event` parameter is the close event in this case, and is needed to reject or allow the closing of the application. Now this is complete, the main menu has all of the functionality required.

```
class MainApp(QtWidgets.QMainWindow, mainui.Ui_main):
  def __init__(self):
    super(self.__class__, self).__init__()
    self.setupUi(self)
    self.pushButton.clicked.connect(self.button_clicked)
    self.suvatApp = SuvatApp(self)

  def button_clicked(self):
    self.hide()
    self.suvatApp.show()

  def closeEvent(self, event):
    # noinspection PyCallByClass,PyTypeChecker
    reply = QtWidgets.QMessageBox.question(self, 'Alert', "Are you
    sure about that?",
    QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.No,
    QtWidgets.QMessageBox.No)
    if reply == QtWidgets.QMessageBox.Yes:
      event.accept()
    else:
      event.ignore()
```

Listing 3.19: Final code for the main menu

3.4.4 Question screen

As the main menu is complete, the question screen can now be coded. This is a lot more complex than the main menu screen, as the graph and question generation code has to be linked to the GUI.

Linking question generation

First a question skeleton must be obtained. A text file of random question skeletons was created and called `projectilemotionquestion.txt`. An example question skeleton from this text file

```
A ball is projected with speed {a:random:20:40}.  At the starting
point the ball is {b:random:5:30}m of the ground.  The highest
point of the ball is {equation:type:findtheta}
```

Figure 3.17: Example question skeleton

Each question skeleton takes up one line of the text file, so to choose a random skeleton all that needs to be done is choose a random line from the file.

```
def load(questionType, object):
  lines = [line.rstrip('\n') for line in open(questionType + ".txt"
    )]
  return object.format(random.choice(lines))
```

Listing 3.20: Function to return random formatted question

This function takes the question type, and an object as parameters. `questionType` is used to create the file name of the questions, and `object` is the `Randomized` object. This is so that the instance of the function is the same when the graph is generated, so that the variables created for the question are consistent with the ones used for the graph generation. The first line uses a one line for loop, that goes through every line in the text file, and strips the new line character from the end of them. These stripped lines are then added to a list. The next line then chooses a random item from the list, formats it and then returns it.

So now the GUI has access to a random question, and a Randomized class with the same variables as in the question.

Linking graph generation

In Section 3.3, code was created to make the diagram. To generate the diagram related to the question, we need to give it the variables from the question, and also the question type so that it knows which variables to not show in the diagram, as these are to be found by the user. This was done by creating a class with different methods for the three different variables to be found, x distance, θ and the maximum height. The code for these methods are very similar to each other, with only some labels missed out.

To get the graph to be generated, we must run the `find_x` function, where `x` is the name of the variable to be found. However, we must then find the answer to the question, so the class must be returned as well.

```
def get_class(self):
  if self.args['equation'] == "findtheta":
    self.question = questionplotclass.ProjectileQuestion(self.args[
    'b'], self.args['a'], random.randint(40, 60))
    self.question.find_theta()
    return self.question
  if self.args['equation'] == "findmaxheight":
    self.question = questionplotclass.ProjectileQuestion(self.args[
    'c'], self.args['a'], self.args['b'])
    self.question.find_max_height()
    return self.question
  if self.args['equation'] == 'findxdistance':
```

```
11      self.question = questionplotclass.ProjectileQuestion(self.args[
        'c'], self.args['a'], self.args['b'])
12      self.question.find_xdistance()
13      return self.question
```

Listing 3.21: Additional method in `Randomized class to generate graph and return class used to do this`

This is a small piece of conditional logic that takes the equation type, which is stored in the args when the string is formatted, and generates the graph depending on it. After this is done it then returns the Class used to generate the diagram, so additional functions can be called on it, for example finding the answer to the question for answer verification.

To then interact with this function from the GUI, the `get_class` function must be called.

Using links

Now that we have links to the necessary parts of code, we can now use these in the GUI. First we must display the question, and this was done like so

```
1  def generate_question(self):
2    string = str(questionStore.load("projectilemotionquestions", self
       .randomised))
3    temp = string
4    temporary_object = self.randomised.get_class()
5    if (self.randomised.args['equation'] == 'findtheta'):
6      temp = str(string) + str(temporary_object.answer_max_height())
       + " Find theta in degreees."
7      self.answer = temporary_object.answer_theta()
8    if (self.randomised.args['equation'] == 'findmaxheight'):
9      self.answer = temporary_object.answer_max_height()
10   if (self.randomised.args['equation'] == 'findxdistance'):
11     temp = str(string) + str(
12     temporary_object.answer_max_height()) + ". How far does the
       ball travel before it hits the ground?"
13     self.answer = temporary_object.answer_xdistance()
14   self.label_3.setText(temp)
```

Listing 3.22: Generate and display question

The first part of the code uses the load function described earlier, and then generates the graph using the `get_class` function. It then checks what the type of equation is and then generates the string based on that. The last line actually sets the label text to the question string.

Now that the question is displayed, the diagram must be displayed. Previously when testing the graph generation, the graph was displayed in an interactive window created by Matplotlib. To use this interactive window in the GUI would be very complex, and interactivity is not essential for the diagram. So the best solution is to generate an image instead. This can be done easily by replacing plt.show() with plt.savefig('filename.png'). In the context of this project the code has been changed to this plt.savefig('test.png', bbox_inches='tight', pad_inches=0).

`bbox_inches='tight'` removes unnecessary white space around the image, as does `pad_inches`.

To scale this image the `PIL` library was used, as this is one of the most documented Python image libraries available. First the `test.png` image is opened by PIL. This image is the large diagram generated by Matplotlib. This image is then reduced in size by a scale factor of about 0.5, and then saved as a new image called `smaller.png`.

```
plt.savefig('test.png', bbox_inches='tight', pad_inches=0)
im = Image.open('test.png')
size = np.asarray(im.size)
size = size / 1.4
size.tolist()
im.thumbnail(size)
im.save('smaller.png')
```

Listing 3.23: Code to scale image keeping aspect ratio

In this code first the diagram is saved as an image. This image is then opened by the `PIL` library. The image size is then stored in array, and then converted into a `numpy` array. This is so that division can be done on the array, which is dividing each element individually. This saves time as an iterative loop to do this does not have to be developed. This array is then divided by 1.4, and this smaller array is used as the size of the next image. The `thumbnail` function from `PIL` is used to create an image given the size, so the thumbnail function is then called using the smaller size. The created thumbnail is now the scaled version of the original image, and this is then saved. This code was then added to the graph generation code.

Now the smaller diagram must be displayed on the GUI. This can be done using a `Qt Pixmap`. As said before, there is no easy to use image item in Qt creator, so to show an image, pixmap must be used. The code to do this is shown here

```
pixmap = QtGui.QPixmap("smaller.png")
self.label.setPixmap(pixmap)
os.remove('test.png')
os.remove('smaller.png')
```

Listing 3.24: Setting a label to display an image

This code first creates a `QPixmap` object using the small diagram. It then sets the pixmap of the label to this. The last two lines remove the two images files so that when the program is run again, there is not a problem with permissions when Python tries to overwrite the image.

The code for preparing the question GUI is complete, and now looks like this

```
def generate_question(self):
  string = str(questionStore.load("projectilemotionquestions", self
    .randomised))
  temp = string
  temporary_object = self.randomised.get_class()
  if (self.randomised.args['equation'] == 'findtheta'):
    temp = str(string) + str(temporary_object.answer_max_height())
    + " Find theta in degreees."
```

```
    self.answer = temporary_object.answer_theta()
  if (self.randomised.args['equation'] == 'findmaxheight'):
    self.answer = temporary_object.answer_max_height()
  if (self.randomised.args['equation'] == 'findxdistance'):
    temp = str(string) + str(
    temporary_object.answer_max_height()) + ". How far does the
    ball travel before it hits the ground?"
    self.answer = temporary_object.answer_xdistance()
  self.label_3.setText(temp)

  pixmap = QtGui.QPixmap("smaller.png")
  self.label.setPixmap(pixmap)
  os.remove('test.png')
  os.remove('smaller.png')
```

Listing 3.25: Completed question GUI preparation function

Answer verification

Now that the prepared question screen is available to the user, the user's answer must be checked so that they can be given feedback based on it. In the `generate_question` function, the answer to the question is stored as a class variable. So the first method of checking the user's answer was to compare the class variable with the answer, and the text in the textbox when the user submits their answer. This is done like so

```
if str(self.answer) == str(self.lineEdit.text()):
  QtWidgets.QMessageBox.information(self, "Well done", "Congrats")
```

Listing 3.26: First method of verifying user's answers

When the user's answer is correct, a message box appears to congratulate them.

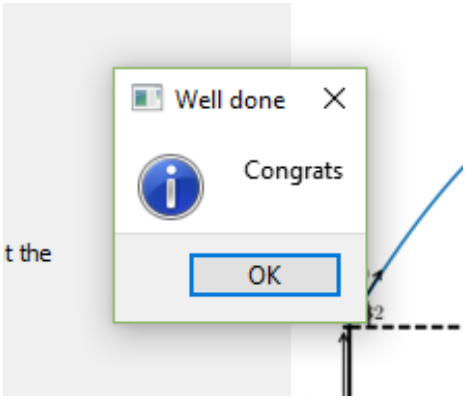

Figure 3.18: Congratulation popup

The problem with this approach is that the answers to the randomly generated questions are unlikely to be a whole number, and as this approach requires the

user's answer to be exactly the same as the computed answer, the user's answer would have to be over 9 decimal places long. This is an unrealistic expectation, and would cause frustration for the user. A better approach is to accept the user's answer no matter the amount of significant figures it is given to, as long as their answer is correctly rounded.

First a method needs to be made that determines the number of significant figures that the user's answer has been given to. From the range of numbers that are used to generate the question, the correct answers will never be less than 10. This means that to calculate the number of significant figures, you can find the number of digits in the answer given by the user, and then round the actual answer to this amount of digits. This is technically not rounding a number of significant figures in every case, as sometimes the number of significant figures can be ambiguous. For example, is 40 given to 1 or 2 significant figures? So pseudocode for this There is a limitation to the Python `round` function, in that

Algorithm 11 Rounded answer to same significant figures as user's answer

```
sigFig ← length(userAnswer)
roundedAnswer ← round(answer, sigFig)
```

it only rounds a number to a number of decimal places. An example use of the function would be round(1.557, 2) would output 1.56. This is a problem as users may not round their answer to include decimal places. First the case when users give an answer with decimal places will be considered. This algorithm will work

Algorithm 12 Rounding answer to same number of decimal places in user's answer

```
if '.' in userAnswer then
    dp ← length(split(userAnswer at '.'))
    roundedAnswer ← round(answer, dp)
end if
```

whenever a user has a decimal place in their answer. Now for when the user does not give decimal places The two algorithms above can now be put together

Algorithm 13 Rounding to same significant figures with no d.p

```
roundedAnswer ← 17.7954
roundedAnswer ← answer / 10                    ▷ roundedAnswer = 1.7954
roundedAnswer ← round(roundedAnswer, 1)        ▷ roundedAnswer = 1.8
roundedAnswer ← roundedAnswer * 10             ▷ roundedAnswer = 18
```

to provide code that will mark the student's answer as correct no matter the significant figures as long as it is rounded correctly.

```
def submit(self):
  if '.' in self.lineEdit.text():
```

```
3       exponent = len(self.lineEdit.text().split('.')[1])
4       rounded_answer = round(self.answer, exponent)
5       print(rounded_answer)
6     else:
7       rounded_answer = self.answer / 10
8       print(rounded_answer)
9       rounded_answer = round(rounded_answer, 1)
10      rounded_answer *= 10
11      print(rounded_answer)
12      rounded_answer = int(rounded_answer)
13    if str(rounded_answer) == str(self.lineEdit.text()):
14      QtWidgets.QMessageBox.information(self, "Well done", "Congrats"
        )
15      self.lineEdit.setText("")
16      self.generate_question()
17    else:
18      QtWidgets.QMessageBox.critical(self, "Incorrect", "Unlucky m8")
19      self.lineEdit.setText("")
```

Listing 3.27: Answer verification

This code uses the algorithm above, and gives feedback depending on the answer, either a congratulation box, or a rejection box. If the user gets the question correct, the textbox is cleared and a new question is generated.

With the completion of this code, the initial development of the software is complete, and now testing must be done to add further developments to the software.

Chapter 4

Testing to inform development

4.1 Testing areas

As stated in Section 2.6 of the design, the areas to be tested are

- Time to generate question
- Time to display graph/diagram
- Generation of variables are in specified range
- Graph has reasonable scale so information is easy to read
- Graph image is high enough resolution to be seen clearly

4.2 Testing tools

4.2.1 Timing

The majority of this chapter will be on the optimisation of code so that its run time is shorter. So how does one measure the run time of a piece of code? The first obvious method is to use a stopwatch, which works for code where there is a visual representation of when the code stops and starts. However this is inaccurate, and for small optimisations the difference in run time will not be noticeable. Luckily the built in Python library `timeit` can measure the time taken to execute a piece of code.

```
import timeit
print(timeit.timeit("for i in range(1,1000):  print(i)",
number=1))
>>> 0.006363171571877327
```

Figure 4.1: Timing code execution using `timeit`

4.2.2 User review

While not a big feature in this section, the reception of the user interface is important, as the users of the program will have useful input on what features to add, and how to change the program to make it better to use.

4.3 Time taken to generate question string

To test this, we must add in some statements from the `time` function. First we create a variable before the code to be executed called `start`. This variable will be assigned the exact time when the code reaches that point. Then another variable is created after the code that is being timed. This variable is assigned the exact time at that point as well. Now that we have two variables, one with the time before execution and one with the time after execution, we can subtract the initial time from the final time to get the time taken to execute the piece of code. This is done like so

```
1 start = timeit.default_timer()
2 #Code to be tested
3 stop = timeit.default_timer()
4 print(stop - start)
```

Listing 4.1: Timing code runtime

When the code which generates the question string is tested, its runtime is much less that 0.1 of a second.

```
Time taken: 0.00035555575004758976
Answer: 29.3919094966
Time taken: 0.0004284902628786 1136
Answer: 125.489997425
Time taken: 0.00040405720108083 19
Answer: 52.8563205999
Time taken: 0.00051090626237737 52
Answer: 43.5939815548
```

Figure 4.2: Recorded runtime of code that generates random question string.

The time taken for this is clearly acceptable, and the code cannot be optimized more than this. The area of question preparation that is expected to take the longest amount of time to execute is the graph generation.

4.4 Time taken to generate diagram

First, the time taken to generate the graph using the first iteration of the code was tested.

```
Time taken: 1.73458798077890843
Answer: 20.354893242
Time taken: 1.453458934590034509
Answer: 78.237849023
Time taken: 1.923457647732843847
Answer: 54
Time taken: 1.843276478628934789
Answer: 42.58239054774
```

Figure 4.3: Recorded runtime of initial graph generation code

Looking at the results above, this is not an acceptable runtime. In the success

criteria, it was stated that the questions must be prepared within 1 second, otherwise the user would get bored, and it would make the user interface much more tiresome to use. So some modifications to the code must be made.

A hypothesis was created as to why the runtime was so long, and this was because, as the range of graph got bigger, it meant that more points had to be generated, and therefore it took too long. This was tested by increasing the x distance, so that more points had to be created. The results of this test are shown below.

```
Time taken: 0.48238492348932048 3
Answer: 1.23432432
Time taken: 0.882348238498234898
Answer: 50.2348293483
Time taken: 1.389228394823948234
Answer: 100.28349032874
Time taken: 2.182384923489290824
Answer: 200.2394280934890324
```

Figure 4.4: Recorded runtime as x distance is increased

From these results it is obvious that as the number of calculations increases, so does the runtime. In order to optimise the code, we must know what the complexity of this algorithm is. To help with this, a graph of run time against number of calculations has been drawn using `matplotlib`.

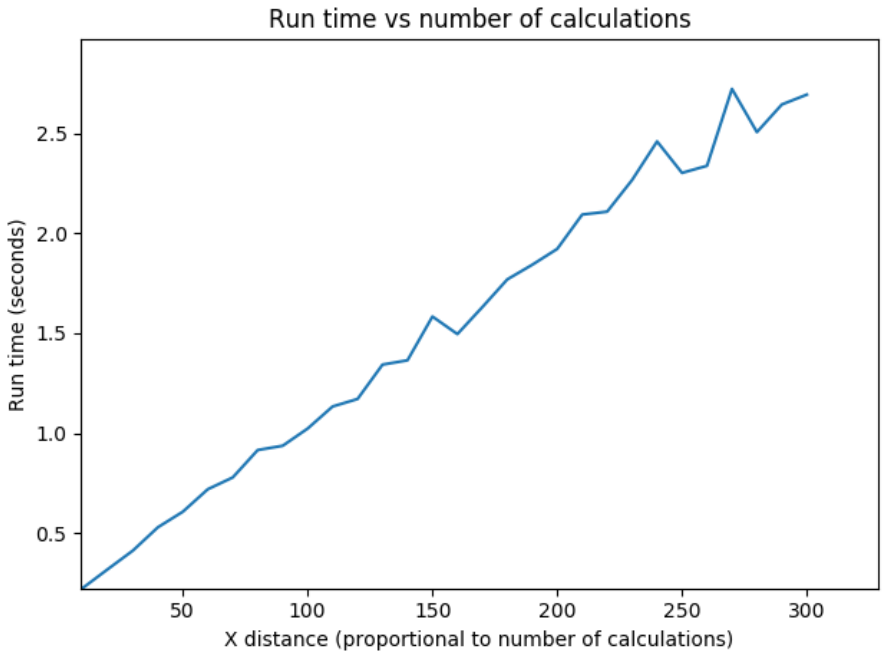

Figure 4.5: Runtime as a function of number of calculations

From looking at the graph, it shows that this algorithm has a linear relationship between runtime and number of calculations, so in Big-O notation, this could be written as `O(n)`. Now that we know this, we must find a way to reduce the number of calculations as the x distance gets bigger. The best way of doing this would be to increase the increment between points as the x distance gets bigger. This is because as the x distance gets bigger, the graph will get bigger so the resolution of the graph will become less important. So to save time we can increase the distance between points. The downside of this is that the graph may appear slightly jagged, but `matplotlib` can smoothly interpolate points to generate a smooth curve that goes through the points.

So the way to fix this is to multiply the increment by a small constant and the x distance. So as the x distance gets bigger, the increment gets bigger, but not by a large amount. The code is changed like so

```
1 increment = 0.001 * 0.1 * self.x_distance
```

Listing 4.2: Optimisation change to graph generation

Now the increment will be proportional to the number of calculations, unlike before when it did not change. In theory this should mean that it takes the same amount of time to run no matter the x distance, as it should always be doing the same amount of calculations, but in practice that will not be the case because the scale factor will not be accurate enough for that.

Another collection of runtime tests were done, and a graph was produced to show the difference in runtime between the optimised and non optimised code.

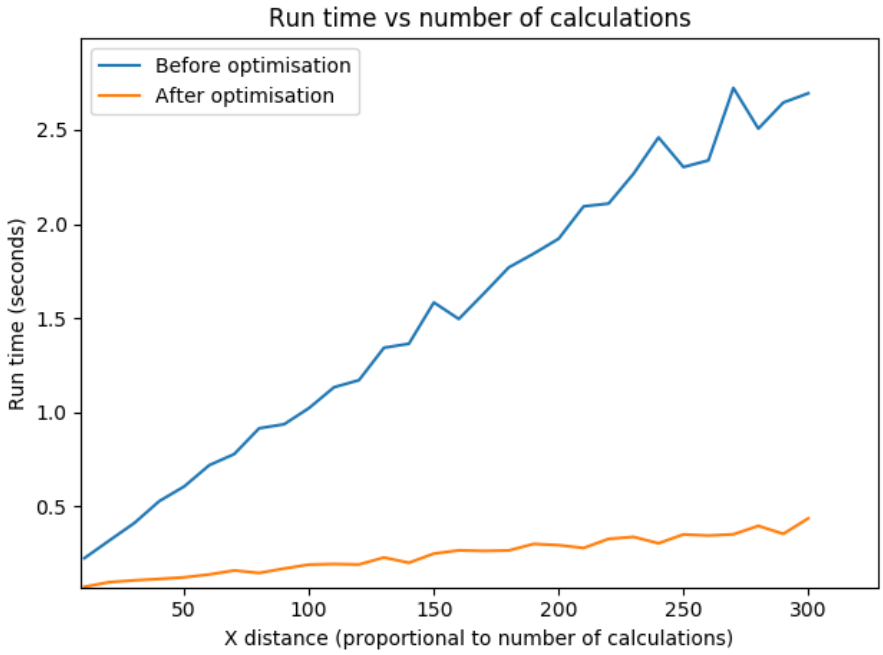

Figure 4.6: Comparison between optimised and non optimised code

It is now obvious from this figure that the optimised code has done its job. Now that this code has been optimised, the overall runtime will be tested.

4.5 Overall runtime

Now that the two main parts of the question preparation have been optimised, the overall runtime can be tested to see if it is under 1 second, as specified in the success criteria.

Runtime (s)	Average runtime (s)
1.282348	
0.972347	
1.588344	1.218165
1.194223	
1.053565	

Figure 4.7: Runtimes of `generate_question function`

So the average runtime is 1.218165 seconds. This is above the 1 second stated by the success criteria, but it is not by a big amount, and there is little to be done to the code to optimise it further.

4.6 Diagram clarity

During the design of the diagram, the ideal resolution was tested, to find a balance between time taken to create the image and clarity of the image. As the resolution of the image is increased the time to create also increases. So during the development this was tested to find the perfect resolution which balances out the two sides.

Also the size of the image was important, and it was found in testing that the image was too big to fit on the GUI. The code was changed to accommodate this, by using the `PIL` Python library to reduce the image size by just under a half.

Chapter 5

Testing to inform evaluation

5.1 Generation of variables

As outlined in the design section, checking that the variables are within the specified range is to be tested. This is not too hard to test, as it is just making sure that the Python `random` function is working correctly. To do this a small piece of code was made

```
for i in range(1, 10000):
  temp = random.randint(20,40)
  if( temp < 20 or temp > 40 ):
    print("Failed")
```

Listing 5.1: Variable range test

This code passed, so there is no problem with the range of variable generation.

5.2 Realistic values

It is stated in the success criteria that the program should have "realistic values in the question.". To test this I ran the program a few times, and looked at the answers for the questions. While they were in a realistic range, i.e < 150 meters, almost none of them were whole numbers, except for the "find θ" type questions. The "find θ" questions always have whole number answers, as the question is generated around the answer, not the other way round. However with the other two question types, "find max height" and "find x distance", the answer is based on the randomly generated question. This means that the answer can be messy. As the answers are not exact, a system for accepting any answers no matter the significant figures had to be developed.

5.3 Graph generation time

This was tested mostly in the previous section, found in Section 4.4. This is an extract of the important data

Runtime (s)	Average runtime (s)
1.282348	
0.972347	
1.588344	1.218165
1.194223	
1.053565	

Figure 5.1: Runtimes of `generate_question function`

showing that the average runtime of the question generation time, was slightly greater than one second.

5.4 Answer verification

To test this, answers with many significant figures were tested.

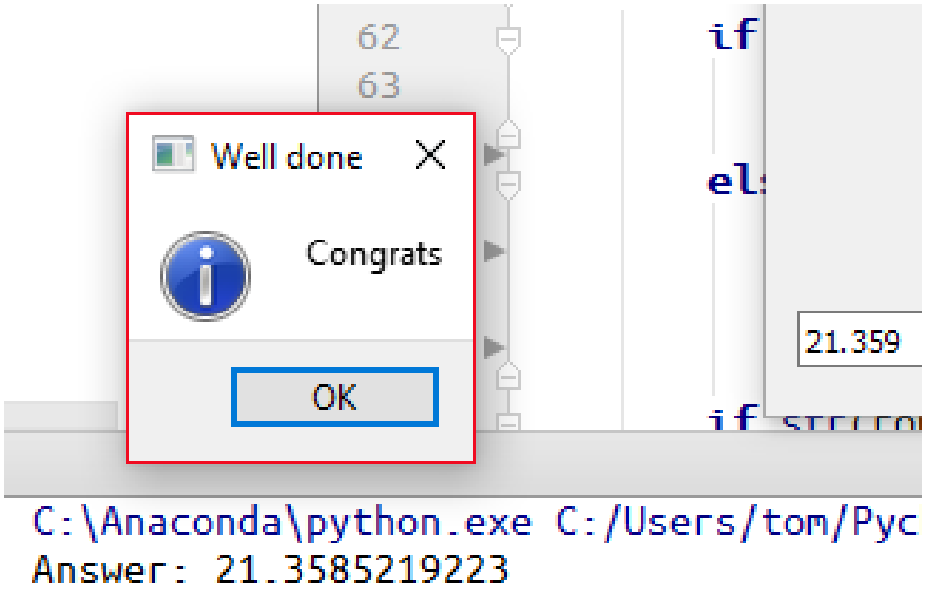

Figure 5.2: Significant figures test scenario

In the middle right of the image is the submitted answer, to the bottom is the actual answer, and you can see the confirmation box showing that the answer was accepted.

However there was a scenario where the answer was rejected when it was correct. This happened when the user gave an answer that had a greater number of significant figures than the computer had calculated.

Chapter 6

Evaluation

6.1 Comparing test data with success criteria

The success criteria for this project were outlined in Section 1.10, but will be rewritten here.

- It should have a functional, easy to use GUI.
- It will have randomly generated questions
- It should have realistic values in the question
- Random question should be shown to the user within a second of page display
- Graph or diagram will be generated in relation to question
- Graph should be displayed within a second
- Student's answers should be verified no matter the amount of significant figures.

The test data collected in the previous section will now be checked against the success criteria, to see how well the program has met these. Only the non subjective success criteria will be checked in this section.

6.1.1 Randomly generated questions

This was tested in Section 5.1. As seen in the test data, the questions were random. This was confirmed by generating 10000 questions, and checking that they were in the correct range as specified in the question skeleton, and also checking that they were suitably random. The generated variables were confirmed to be in the correct range, so this part of the test was passed.

It is hard to say whether the values produced are absolutely random. This is because of our perception of randomness. A human determines whether a number

is random or not based upon the numbers before and after it in a sequence. So if we had the number 5 on its own for example, it would not be considered random. However if we had the sequence "1, 4, 19, 234324, 5432, 5, 394" and we looked at the number 5, this would be perceived as being random, as it has no obvious relation to the other numbers in the sequence. So from looking at the test data, each individual number has no obvious relation to another in the sequence, so the number generation can be classified as random. This means that the variables in the question can be called random.

The last part of the question generation is the skeleton itself. This part could not be called random, as there are only three types of base questions. To this extent the questions are not fully randomly generated.

To summarise, this point of the success criteria has been almost fully met. This is because the numbers generated are random, but the actual question itself is not fully random. This can cause a problem, as if the program is used extensively, the user may start to learn tricks on how to answer that specific type of question, rather than focusing on the general method of solving mechanics problems. I am not fully happy with the question generation, but with the time available, I think it was the best solution that could have been created.

6.1.2 Realistic values

The values can be realistic in two different senses, that they are values that would appear in a real life scenario, or they are values that would appear in an exam question. The values produced in this program conform more to the latter, as this tool is intended to be used mainly for revision purposes. The data collected is shown in Section 5.2, and the main point shown is that the answers are not exact. This is a good feature, as it is how it would be in an exam, and making all of the answers exact means that the user can tell if their answer is more likely to be correct. This means that the questions where the user is asked to "find θ" could be improved to have non exact answers, as all of the answers to this question type are an integer between 40 and 60 degrees, so this can help the user to check their answer, which is an unwanted feature. This is an unwanted feature because there would not be a confidence range like this in exam questions.

This point in the success criteria is met well, as the answers are similar to what would be found in an exam. The only problem in this section is the answers to the "find θ" questions, as they are not like what would be seen in an exam.

6.1.3 Display time

The data for this is analysed in detail from Section 4.3 to 4.5. The important data to take from this is Figure 4.7, which is shown below.

Runtime (s)	Average runtime (s)
1.282348	
0.972347	
1.588344	1.218165
1.194223	
1.053565	

Figure 6.1: Runtimes of `generate_question function`

This shows that the average run time is just over 1 second, at 1.218165 seconds. This does not fully meet the point in the success criteria, that states that the "Random question should be displayed to the user within a second". However, 1.2 seconds is only slightly above the require time, so the users will not be affected by this too much. Over a long period of use, this additional time would add up, and it could become an annoyance to the user.

Overall this point in the success criteria has not been met very well. It is over the required time, and this time was set, to avoid user frustration. Additionally, one of the points of developing this software was to save time for the user, so that they don't have to spend time finding questions on a topic that they would like to practice. So having to wait a long time for a question to be displayed to them might make the user stop using using the program.

6.1.4 Answer verification

The test data for this can be found in Section 5.4. It has shown that the student's answer is marked as correct no matter the number of digits, not the number of significant figures. There is one case where the student's answer was marked as incorrect when it was correct, which is when the user gave their answer to more digits than the computer had calculated in their answer.

The point in the success criteria, "Student's answers should be verified no matter the amount of significant figures", has almost been met, but instead of depending on the number of significant figures, this solution depends on the number of digits in the student's answer. In practice their is no difference in using significant figures or number of digits, so this is not a problem. However, the case where user's answers were marked incorrect when they gave their answer to more digits than the programs answer is a problem, as it will confuse students when they have a technically correct answer that is marked wrong. This is an extreme case however, as it is unlikely that a student will give their answer to this amount of digits. Also if the student gave their answer to this many digits in an exam, they would be marked wrong.

6.1.5 GUI

There is no test data for this part, as it is mostly subjective. So from my experience of the program, and from the feedback from stakeholders who have tested the product, the GUI is easy to use. The question is clearly visible, and

so is the diagram. The diagram could be zoomed in more, as the values can be hard to read when the particle is launched with a high speed and angle. It is clear where to submit your answer, due to the placeholder text in the submit box.

Improvements to be made are to make the layout cleaner, as at the moment there are large gaps between the elements, and this is wasted space on the screen. Also the text in the buttons needs to be redesigned so that it is smaller, or the button is bigger. This is because not all of the button text is visible, which could cause confusion to the user.

6.2 Usability features

6.2.1 GUI

The usability feature required was "An easy to use, intuitive GUI". This has been accomplished as discussed in the previous section, Section 6.1.5. The user knows what to do without being told, and they can do this easily. To improve usability, tooltips could be added on certain buttons, to explain to the user further what specific buttons do.

6.2.2 Answer feedback

This has also been achieved. If the user gets the answer wrong they are taken back to the question screen, and they have the option to skip the question using the easy to see button in the top right hand side of the screen. The user does not have an option to skip the question in the popup menu telling them that they got the question wrong however, so this could be added in later development to save time for the users.

6.3 Further development

6.3.1 Question generation

To further develop this there are two options, either create many more question skeletons, or create a system that generates question skeletons. The latter solution would require complex machine learning, and is much too complex for a program like this. It would be better to create more question skeletons, which would not take much time. All that has to be done is find exam questions, or textbook questions and then isolate the variable that should be randomised in the question. The question can then be added to the textfile using syntax required. A new equation type method would have to be developed for each new question skeleton so this would take a long time. Also a new diagram would have to be created for them, which would also take time.

6.3.2 Graph generation

The scale for the diagram can be changed so that the values are easier to read when the image is zoomed out. The graph becomes too zoomed out when the maximum height of the projectile is big, so this could be fixed by either reducing the maximum angle and maximum speed in the question skeleton, or changing the size of the labels depending on the zoom factor of the diagram.

6.3.3 GUI

This can be developed to improve the layout, so there is less blank space between each important element on screen. Also it could be made so that it is scalable with higher resolution screens. This was found to be a problem when the program was tested on a 3840 x 2160 screen. Also when the program is resized, the elements don't move. This can cause the elements to be squashed off the visible space, or leave a big space of grey, which is ugly.

6.3.4 Answer feedback

A way of restricting user's answers to the same number of digits as the answer calculated by the program should be developed, to stop situations where a technically correct answer is marked as incorrect by the program.

6.4 Maintenance and limitations

The only maintenance required for this software would be to change the questions, or to add new ones when the exam board changes the question style. As there will be a new maths syllabus starting next academic year, this could render this program obsolete.

References

[1] PHET Interactive Simulations. *Projectile motion*. URL: https://phet.colorado.edu/en/simulation/projectile-motion (visited on 13/01/2017).

[2] MathsBank. *M2 Projectile motion question bank*. URL: http://mathsbank.co.uk/home/a-level/m2/kinematics/projectiles/stones-collide (visited on 16/01/2017).

[3] S-cool. *Exam-style Questions: Radioactive Decay Equations*. URL: http://www.s-cool.co.uk/a-level/physics/radioactive-decay-equations/test-it/exam-style-questions (visited on 16/01/2017).

[4] Wikipedia. *Qt (software) — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=Qt_(software)&oldid=759833448 (visited on 17/01/2017).

[5] Python Software Foundation. *Built in types - Sequence types - String formatting*. URL: https://docs.python.org/2/library/stdtypes.html#str.format (visited on 18/01/2017).

[6] Talin. *PEP 3101 - Advanced string formatting*. URL: https://www.python.org/dev/peps/pep-3101/.

[7] Matplotlib development team. *Matplotlib 2.0 Documentation*. URL: http://matplotlib.org/contents.html.

[8] Continuum Analytics. *Anaconda package list*. URL: https://docs.continuum.io/anaconda/pkg-docs.

[9] Openstax CNX. *Motion in two dimensions - Vertical projectile motion*. URL: http://archive.cnx.org/contents/86e923ab-903f-40bb-b207-08bb57768704@1/motion-in-two-dimensions-vertical-projectile-motion (visited on 24/02/2017).

[10] The Latex Project. *An introduction to LaTeX*. URL: http://www.latex-project.org/about/ (visited on 24/02/2017).