King Edward VI Southampton

A Level Coursework

Physics Revision Program

Tom Eaton

supervised by
Mr. Allen and Mr. Mapstone

31st January 2017

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

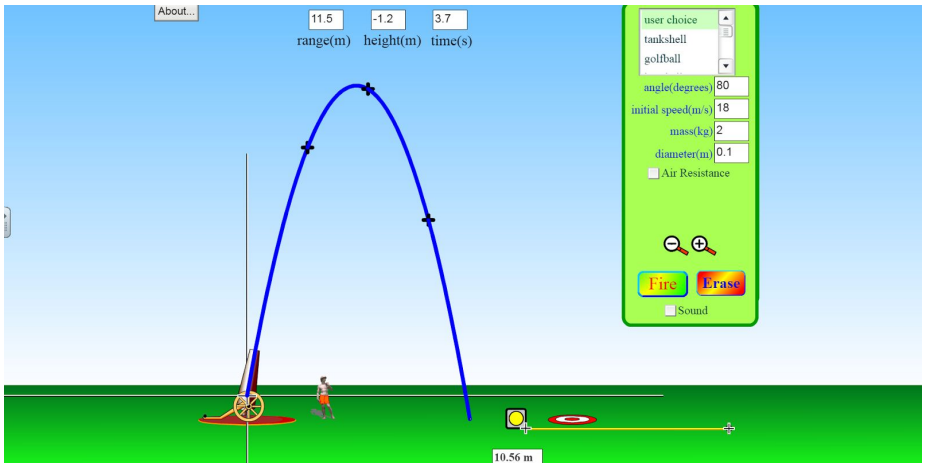


Figure 1.1: PHET Interactive projectile motion simulator. [1]

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.

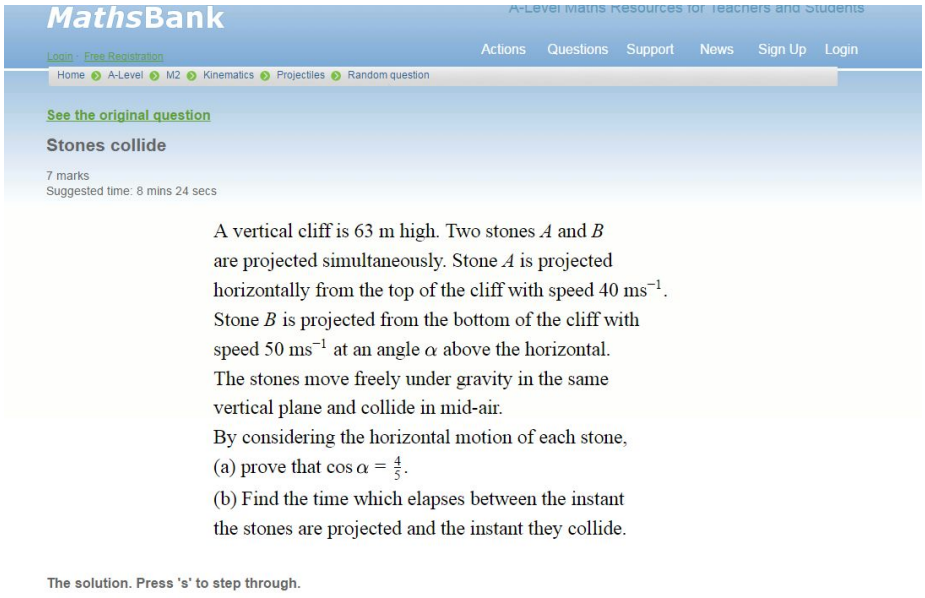
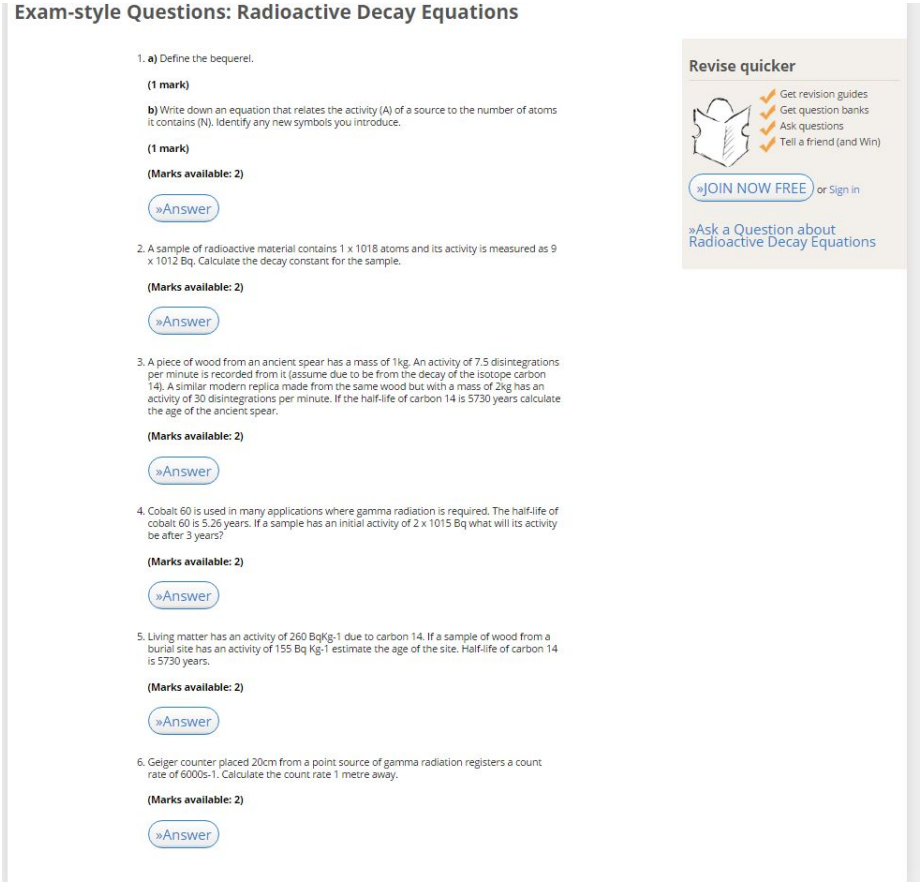


Figure 1.2: MathsBank M2 Projectile motion question bank. [2]

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.



Exam-style Questions: Radioactive Decay Equations

1. **a)** Define the bequerel.

   (1 mark)

   b) Write down an equation that relates the activity (A) of a source to the number of atoms it contains (N). Identify any new symbols you introduce.

   (1 mark)

   (Marks available: 2)

   »Answer

2. A sample of radioactive material contains 1 x 1018 atoms and its activity is measured as 9 x 1012 Bq. Calculate the decay constant for the sample.

   (Marks available: 2)

   »Answer

3. A piece of wood from an ancient spear has a mass of 1kg. An activity of 7.5 disintegrations per minute is recorded from it (assume due to be from the decay of the isotope carbon 14). A similar modern replica made from the same wood but with a mass of 2kg has an activity of 30 disintegrations per minute. If the half-life of carbon 14 is 5730 years calculate the age of the ancient spear.

   (Marks available: 2)

   »Answer

4. Cobalt 60 is used in many applications where gamma radiation is required. The half-life of cobalt 60 is 5.26 years. If a sample has an initial activity of 2 x 1015 Bq what will its activity be after 3 years?

   (Marks available: 2)

   »Answer

5. Living matter has an activity of 260 BqKg-1 due to carbon 14. If a sample of wood from a burial site has an activity of 155 Bq Kg-1 estimate the age of the site. Half-life of carbon 14 is 5730 years.

   (Marks available: 2)

   »Answer

6. Geiger counter placed 20cm from a point source of gamma radiation registers a count rate of 6000s-1. Calculate the count rate 1 metre away.

   (Marks available: 2)

   »Answer

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

[1]What you see is what you get.

be made 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.

2.1.1 Generation of question

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

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

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 Diagrams

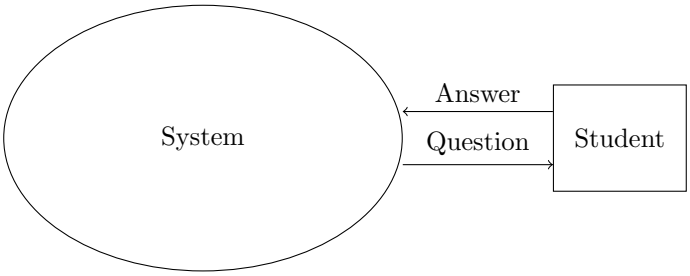


Figure 2.1: Context Diagram

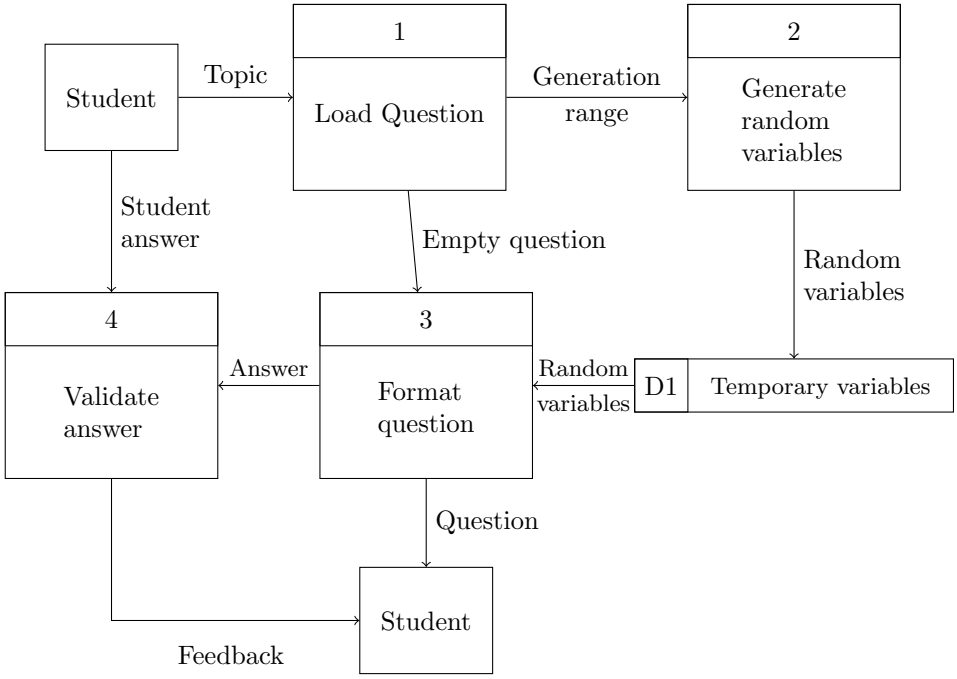


Figure 2.2: Level 1 Dataflow Diagram

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 Datastructures

Algorithm 1 Randomised

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

Algorithm 2 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 GETANSWER(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 ← ProjectileQuestion(self.args['a'], self.args['b'],
    self.args['c']))
25:        end if
26:        if self.args['equation'] = 'findtheta' then
27:            self.question ← ProjectileQuestion(self.args['a'], self.args['b'],
    self.args['c']))
28:        end if
29:        if self.args['equation'] = 'findtheta' then
30:            self.question ← ProjectileQuestion(self.args['a'], self.args['b'],
    self.args['c']))
31:        end if
32:    end function
33: end Class
```

Algorithm 3 RandomisedFormatter

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

Algorithm 4 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 5 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 6 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, self.xPosArray, self.yPosArray,
   graph.png)
34:        resize(graph.png, 0.5)
35:        save(graph.png)
36:    end function
37:    function findMaxHeight
38:        self.calculateProjectile()
39:        plotAsImg(maxHeightDiagram, self.xPosArray, self.yPosArray,
   graph.png)
40:        resize(graph.png, 0.5)
41:        save(graph.png)
42:    end function
```

Algorithm 7 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 8 RadioactiveQuestion

```
1: Class RadioactiveQuestion
2:     public
3:         Function calculateDecay
4:         Function findDecayConstant
5:         Function findHalfLife
6:         Function findActivity
7:         Function findParticles
8:         Function getDecayConstant
9:         Function getHalfLife
10:        Function getActivity
11:        Function getParticles
12:    endpublic
13:    private
14:        decayConstant : Float
15:        halfLife : Float
16:        activity : Integer
17:        particles : Integer
18:    endprivate
19: end Class
```

2.3.1 Justification

In Algorithm 2.3 and 2.3 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 2.3 and 2.3 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 2.3.1 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.

2.4 Algorithms

N.B In all of these algorithms the classes defined in Section 2.3 will be referenced.

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).