



KING EDWARD VI SOUTHAMPTON

A LEVEL COURSEWORK

Physics Revision Program

Tom Eaton

supervised by
Mr. ALLEN and Mr. MAPSTONE

3rd March 2017

Contents

1	Analysis	5
1.1	Abstract	5
1.2	Features of the proposed solution that can be solved using computational methods	5
1.2.1	Random nature of questions	5
1.2.2	Answer Verification	6
1.2.3	Generation of diagrams	6
1.3	Stakeholders	6
1.4	Existing Solutions	7
1.5	Solution	9
1.5.1	Question generation	9
1.5.2	GUI	10
1.6	Justification	11
1.7	Essential Features	11
1.8	Limits of proposed solution	12
1.9	Project Requirements	12
1.9.1	Programming language	12
1.9.2	GUI	12
1.9.3	Graph generation	13
1.9.4	Hardware requirements	13
1.9.5	Final Build	13
1.10	Success Criteria	13
2	Design	14
2.1	Design Overview	14
2.1.1	Generation of question	14
2.1.2	Generation of graph / diagram	15
2.1.3	Display of GUI	15
2.2	Dataflow diagrams	16
2.3	Algorithms	17
2.3.1	Justification	17
2.4	Usability Features	18
2.5	Datastructures	19
2.5.1	Classes	19

2.5.2	Variables	25
2.5.3	Explanation	25
2.5.4	Justification	25
2.5.5	Validation	26
2.6	Test data	26
3	Development	28
3.1	Initial testing	28
3.1.1	Storing questions - Regex	28
3.1.2	Storing questions - String formatting	29
3.1.3	Graph generation	29
3.1.4	GUI	30
3.1.5	Summary	31
3.2	Storing questions	32
3.3	Graph Generation	35
3.3.1	Equations	35
3.3.2	Graph drawing	37
3.3.3	Question class	44
3.3.4	GUI Design	44
	References	46

List of Figures

1.1	PHET Interactive projectile motion simulator.	7
1.2	MathsBank M2 Projectile motion question bank.	8
1.3	Exam-style Questions: Radioactive Decay Equations.	9
1.4	Example projectile motion question.	10
1.5	Example radioactive decay question.	10
2.1	Level 0 Dataflow Diagram	16
2.2	Level 1 Dataflow Diagram	16
2.3	Default behaviour of Python's <code>str.format()[5]</code>	25
2.4	Example string for extended string formatter	25
3.1	Initial question store	28
3.2	Question store with identifiable variables	28
3.3	Extracting variables from a string using Python regex	29
3.4	Example of extended string formatting[6]	29

3.5	Code to create Figure 3.6 [7]	30
3.6	Graph generated by code in Figure 3.5	30
3.7	First graph	38
3.8	Example of a graph [9]	39
3.9	Second iteration of graph	41
3.10	Third iteration of graph	42
3.11	Fourth iteration of graph	43
3.12	Final iteration of graph	44
3.13	GUI Design Prototype	45

List of Algorithms

1	Main Algorithm	17
2	Randomised	19
3	Randomised Pseudocode	20
4	RandomisedFormatter	21
5	RandomisedFormatter Pseudocode	21
6	ProjectileQuestion	22
7	ProjectileQuestion Pseudocode	23
8	ProjectileQuestion Pseudocode continued	24
9	RadioactiveQuestion	24
10	Coordinate list generation	36

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

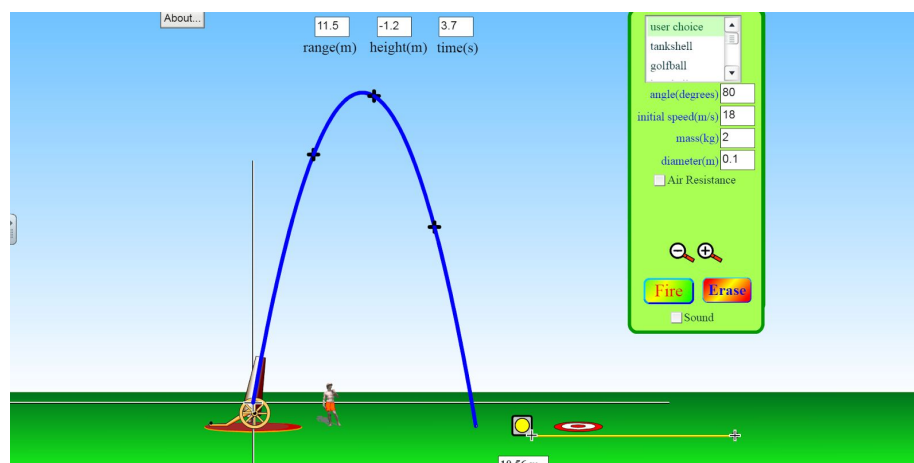


Figure 1.1: PHET Interactive projectile motion simulator. [1]

This piece of software displays 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.

MathsBank A-Level Maths Resources for Teachers and Students

[Home](#)
[A-Level](#)
[M2](#)
[Kinematics](#)
[Projectiles](#)
[Random question](#)

[See the original question](#)

Stones collide

7 marks
Suggested time: 8 mins 24 secs

A vertical cliff is 63 m high. Two stones A and B are projected simultaneously. Stone A is projected horizontally from the top of the cliff with speed 40 ms^{-1} . Stone B is projected from the bottom of the cliff with speed 50 ms^{-1} at an angle α above the horizontal. The stones move freely under gravity in the same vertical plane and collide in mid-air. By considering the horizontal motion of each stone,

(a) prove that $\cos \alpha = \frac{4}{5}$.

(b) Find the time which elapses between the instant the stones are projected and the instant they collide.

The solution. Press 's' to step through.

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 becquerel.

(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 1×10^{18} atoms and its activity is measured as 9×10^{12} 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 1 kg. 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 2 kg 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 2×10^{15} Bq what will its activity be after 3 years?

(Marks available: 2)

»Answer

5. Living matter has an activity of 260 Bq kg⁻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 20 cm from a point source of gamma radiation registers a count rate of 6000 s⁻¹. Calculate the count rate 1 metre away.

(Marks available: 2)

»Answer

Revise quicker



- ✓ Get revision guides
- ✓ Get question banks
- ✓ Ask questions
- ✓ Tell a friend (and Win)

»JOIN NOW FREE or Sign in

»Ask a Question about
Radioactive Decay Equations

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 $x \text{ ms}^{-1}$. At the starting point the ball is $S \text{ m}$ off the ground. The highest point of the ball is $y \text{ m}$.

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 $x \text{ Bq}$. 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 GUI¹ 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.

¹Graphical 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 $\cos \theta$ 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 is 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 WYSIWYG¹ editor called QtCreator, which is similar to the Visual Basic form designer. A GUI can be easily created using this, as it can

¹What 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

Matplotlib¹ 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 virtualenv² 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.

¹Python Library used for plotting graphs.

²An 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

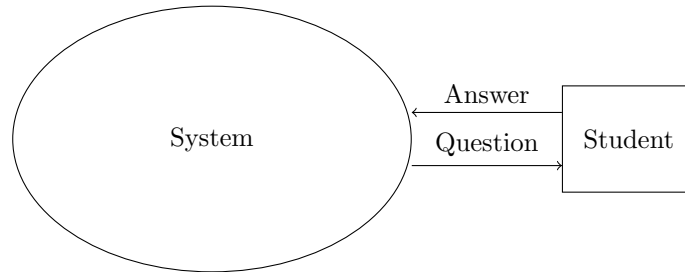


Figure 2.1: Level 0 Dataflow Diagram

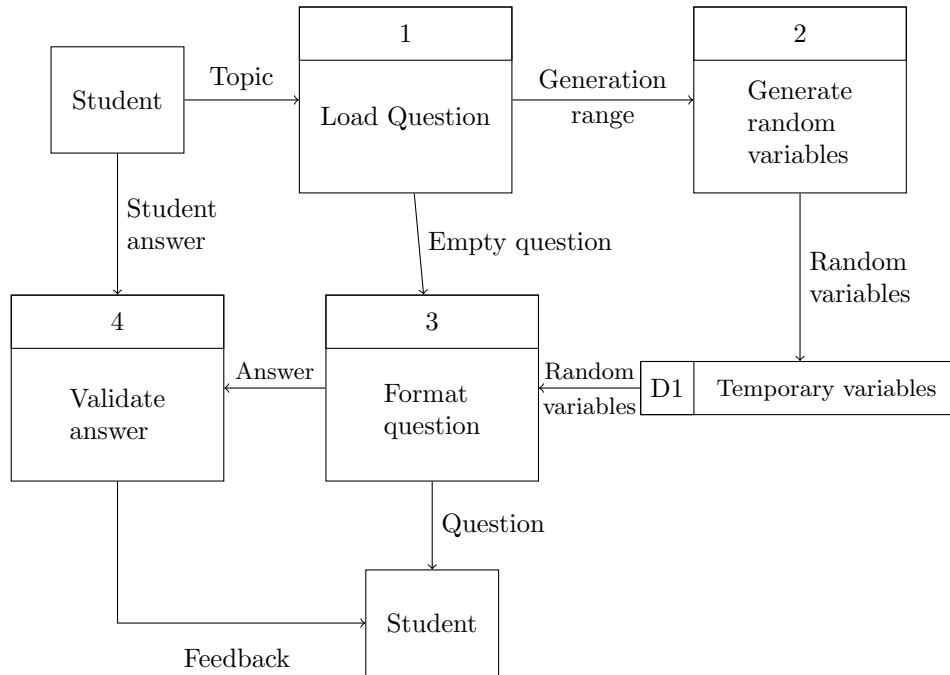


Figure 2.2: Level 1 Dataflow Diagram

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  $\leftarrow$  []
4:     self.question  $\leftarrow$  None
5:   end function
6:   function FORMAT(string)
7:     formatter  $\leftarrow$  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  $\leftarrow$  ProjectileQuestion(self.args['a'], self.args['b'],
self.args['c'])
16:    end if
17:    if self.args['equation'] = 'findmaxheight' then
18:      self.question  $\leftarrow$  ProjectileQuestion(self.args['a'], self.args['b'],
self.args['c'])
19:    end if
20:    if self.args['equation'] = 'findxdistance' then
21:      self.question  $\leftarrow$  ProjectileQuestion(self.args['a'], self.args['b'],
self.args['c'])
22:    end if
23:    if self.args['equation'] = 'findradioactive' then
24:      self.question  $\leftarrow$  RadioactiveQuestion(self.args['a'], self.args['b'],
self.args['c'])
25:    end if
26:    if self.args['equation'] = 'findDecayConstant' then
27:      self.question  $\leftarrow$  RadioactiveQuestion(self.args['a'], self.args['b'],
self.args['c'])
28:    end if
29:    if self.args['equation'] = 'findParticles' then
30:      self.question  $\leftarrow$  RadioactiveQuestion(self.args['a'], self.args['b'],
self.args['c'])
31:    end if
32:  end function
33: end Class
```

Algorithm 4 RandomisedFormatter

```
1: Class RANDOMISEDFORMATTER
2:   public
3:     Function format
4:     Function get_name
5:     Function get_args
6:   endpublic
7:   private
8:     name : String
9:     args : String Array
10:  endprivate
11: end Class
```

Algorithm 5 RandomisedFormatter Pseudocode

```
1: Class RANDOMISEDFORMATTER
2:   function INIT(self, name, args)
3:     self.name  $\leftarrow$  name
4:     self.args  $\leftarrow$  args
5:   end function
6:   function FORMAT(self, fmt)
7:     op, rest  $\leftarrow$  fmt.split(':')  $\triangleright$  text before ':' into op, text after into rest
8:     if op == 'type' then
9:       self.args[self.name] = rest
10:      return None
11:    end if
12:    if op == 'random' then
13:      low, high = rest.split(':')
14:      value  $\leftarrow$  randomNumber(low, high)
15:      self.args[self.name]  $\leftarrow$  value
16:      return string(value)
17:    end if
18:  end function
19: 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  $\leftarrow$  yOffset
4:     self.initialSpeed  $\leftarrow$  initialSpeed
5:     self.theta  $\leftarrow$  theta
6:   end function
7:   function CALCULATEPROJECTILE
8:     increment  $\leftarrow$  0.01
9:     theta  $\leftarrow$  radians(theta)
10:    xSpeed  $\leftarrow$  self.initialSpeed * cos(theta)
11:    ySpeed  $\leftarrow$  self.initialSpeed * sin(theta)
12:    time  $\leftarrow$  0
13:    time  $\leftarrow$  time + increment
14:    self.xPosArray  $\leftarrow$  []
15:    self.yPosArray  $\leftarrow$  []
16:    yPosTemp  $\leftarrow$  5 ▷ To satisfy while loop during first run
17:    while yPosTemp > 0 do
18:      xPosTemp  $\leftarrow$  xSpeed  $\times$  time
19:      yPosTemp  $\leftarrow$  (ySpeed  $\times$  time) + (0.5 * -9.8 * time2 + self.yOffset)
20:      xPosArray.append(xPosTemp)
21:      yPosArray.append(yPosTemp)
22:      time  $\leftarrow$  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

```
43:   function CALCULATEPOINT(startX, startY, angle, length)
44:       endpoint  $\leftarrow$  [startX + (length  $\times$  cos(angle)), startY + (length  $\times$ 
        sin(angle))]
45:       return endpoint
46:   end function
47:   function ANSWERTHETA
48:       return self.theta
49:   end function
50:   function ANSWERXDISTANCE
51:       return max(self.xPosArray)
52:   end function
53:   function ANSWERMAXHEIGHT
54:       return max(self.yPosArray)
55:   end function
56: end Class
```

Algorithm 9 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
```

2.5.2 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 **RandomisedClass**. 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 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 doesn't 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.

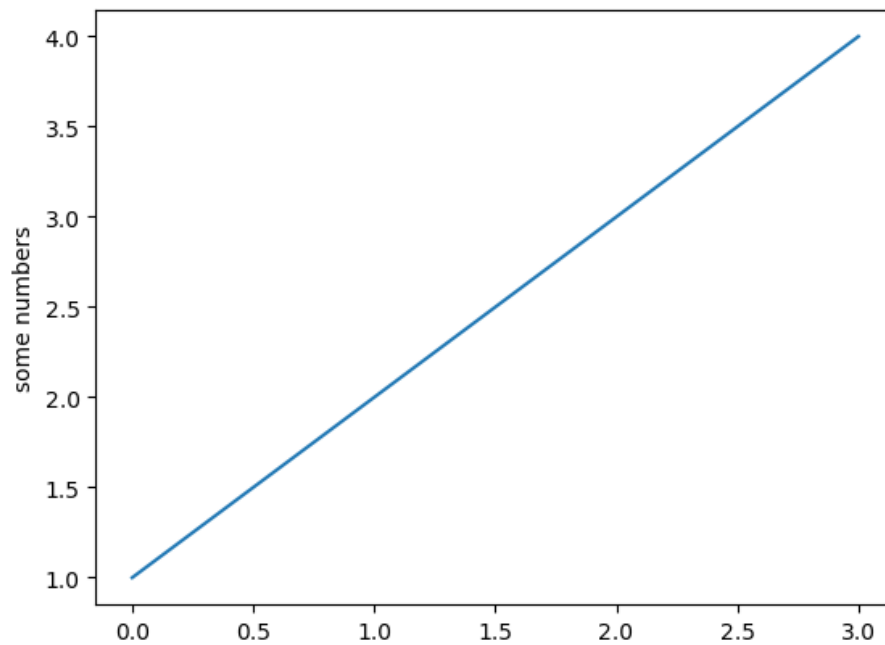


Figure 3.6: Graph generated by code in Figure 3.5

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 WYSIWYG¹ 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

¹What 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.

```
1 #RandomizedFormatter class. Child of Randomized class. Overrides
   String formatter to allow
2 #question formatting.
3 class RandomizedFormatter(object):
4     def __init__(self, name, args):
5         self.name = name
6         self.args = args
7
8     #This function overrides the Python string formatter.
9     def __format__(self, fmt):
10         op, rest = fmt.split(':', 1)
11
12         if op == 'type':
13             self.args[self.name] = rest
14             return ""
15         elif op == 'random':
16             low, high = rest.split(':')
17             value = random.randint(int(low), int(high))
18             self.args[self.name] = value
19             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.


```

1 class Randomized(object):
2     def __init__(self):
3         self.args = {}
4         self.question = None
5
6     #Returns the value of the randomized variable in the string.
7     def __getitem__(self, name):
8         return RandomizedFormatter(name, self.args)
9
10    #Formats the string, using the overridden method found in the
11    #RandomizedFormatter class.
12    def format(self, s):
13        return string.Formatter().vformat(s, args=(), kwargs=self)
14
15    #Returns the question class depending on the equation type.
16    def get_class(self):
17        if self.args['equation'] == "findtheta":
18            self.question = questionplotclass.ProjectileQuestion(self.
19            args['b'], self.args['a'], random.randint(40, 60))
20            answer = self.question.answer_theta()
21            return answer
22        if self.args['equation'] == "findmaxheight":
23            self.question = questionplotclass.ProjectileQuestion(self.
24            args['c'], self.args['a'], self.args['b'])
25            answer = self.question.answer_max_height()
26            return answer
27        if self.args['equation'] == 'findxdistance':
28            self.question = questionplotclass.ProjectileQuestion(self.
29            args['c'], self.args['a'], self.args['b'])
30            answer = self.question.answer_xdistance
31            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:

```

1 #Returns the question class depending on the equation type.
2 def get_class(self):
3     if self.args['equation'] == "findtheta":
4         self.question = questionplotclass.ProjectileQuestion(self.args[
5         'b'], self.args['a'], random.randint(40, 60))
6         self.question.find_theta()
7         return self.question
8     if self.args['equation'] == "findmaxheight":
9         self.question = questionplotclass.ProjectileQuestion(self.args[
10        'c'], self.args['a'], self.args['b'])

```

```

9     self.question.find_max_height()
10    return self.question
11    if self.args['equation'] == 'findxdistance':
12        self.question = questionplotclass.ProjectileQuestion(self.args[
13            'c'], self.args['a'], self.args['b'])
14        self.question.find_xdistance()
15    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.

```

1  import random
2  import string
3  import questionplotclass
4
5  #Randomized class, that is used to format strings, and return
   question objects.
6  #This is the parent of the RandomizedddFormatter class.
7  class Randomized(object):
8      def __init__(self):
9          self.args = {}
10         self.question = None
11
12         #Returns the value of the randomized variable in the string.
13         def __getitem__(self, name):
14             return RandomizedFormatter(name, self.args)
15
16         #Formats the string, using the overridden method found in the
           RandomizedFormatter class.
17         def format(self, s):
18             return string.Formatter().vformat(s, args=(), kwargs=self)
19
20         #Returns the question class depending on the equation type.
21         def get_class(self):
22             if self.args['equation'] == "findtheta":
23                 self.question = questionplotclass.ProjectileQuestion(
24                     self.args['b'], self.args['a'], random.randint(40, 60))
25                 self.question.find_theta()
26                 return self.question
27             if self.args['equation'] == "findmaxheight":
28                 self.question = questionplotclass.ProjectileQuestion(
29                     self.args['c'], self.args['a'], self.args['b'])
30                 self.question.find_max_height()
31                 return self.question
32             if self.args['equation'] == 'findxdistance':
33                 self.question = questionplotclass.ProjectileQuestion(
34                     self.args['c'], self.args['a'], self.args['b'])
35                 self.question.find_xdistance()
36                 return self.question
37
38         #RandomizedFormatter class. Child of Randomized class. Overrides
           String formatter to allow
39         #question formatting.
40         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 + ".
58     txt")]
59     return object.format(random.choice(lines))
60
61 #This code only runs if the file is run specifically, not when it
62 #is imported.
63 if __name__ == '__main__':
64     rr = Randomized()
65     load("projectilemotionquestions", rr)
66     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.

```

1 while (yPosTemp > 0): #While the particle is above the ground
2     xPosTemp = xSpeed * time
3     yPosTemp = (ySpeed * time) + (0.5 * -9.8 * time ** 2)
4     xPos.append(xSpeed * time)
5     yPos.append((ySpeed * time) + (0.5 * -9.8 * time ** 2))
6     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.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 increment = 0.0001 # Time increment used when calculating values
5 initialSpeed = 10
6 theta = np.radians(70) # Angle that the direction of launch makes
   to the horizontal
7 xSpeed = initialSpeed * np.cos(theta) # Using trig to calculate
   xspeed. Assumes there is no air resistance
8 ySpeed = initialSpeed * np.sin(theta)
9 xOffset = 0 # Initial x position. Can be changed to add an offset
   which can be in a question
10 yOffset = 0 # Initial y position. Can be changed to add an offset
   which can be in a question
11 xPos = [] # X position array
12 yPos = [] # Y position array
13 time = 0 # Initial time
14
15 xPos.append(xOffset) # Adding the offset in the first slot of the
   position array
16 yPos.append(yOffset) # Adding the offset in the first slot of the
   position array
17
18 time += increment
19
20 xPosTemp = xSpeed * time # Calculating first x value so while loop
   doesn't fail instantly
21 yPosTemp = (ySpeed * time) + (
22 0.5 * -9.8 * time ** 2) # Using SUVAT to calculate first y value.
   This is so that the while loop doesn't fail instantly
23 xPos.append(xPosTemp)
24 yPos.append(yPosTemp)
25
26 while (yPosTemp > 0):
27     xPosTemp = xSpeed * time
28     yPosTemp = (ySpeed * time) + (0.5 * -9.8 * time ** 2)
29     xPos.append(xSpeed * time)
30     yPos.append((ySpeed * time) + (0.5 * -9.8 * time ** 2))
31     time += increment
32
33 plt.plot(xPos, yPos)
34 plt.axis([0, xPos[len(xPos) - 1], 0, max(yPos)])
35 plt.xlabel("X displacement")
36 plt.ylabel("Y displacement")
37 plt.show()
```

Listing 3.6: First graph generation code

This generates

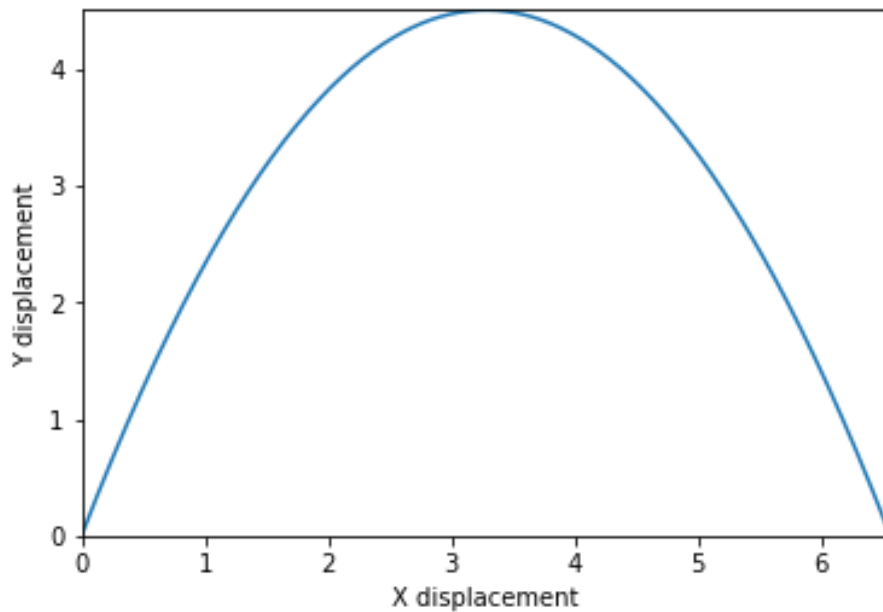


Figure 3.7: First graph

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

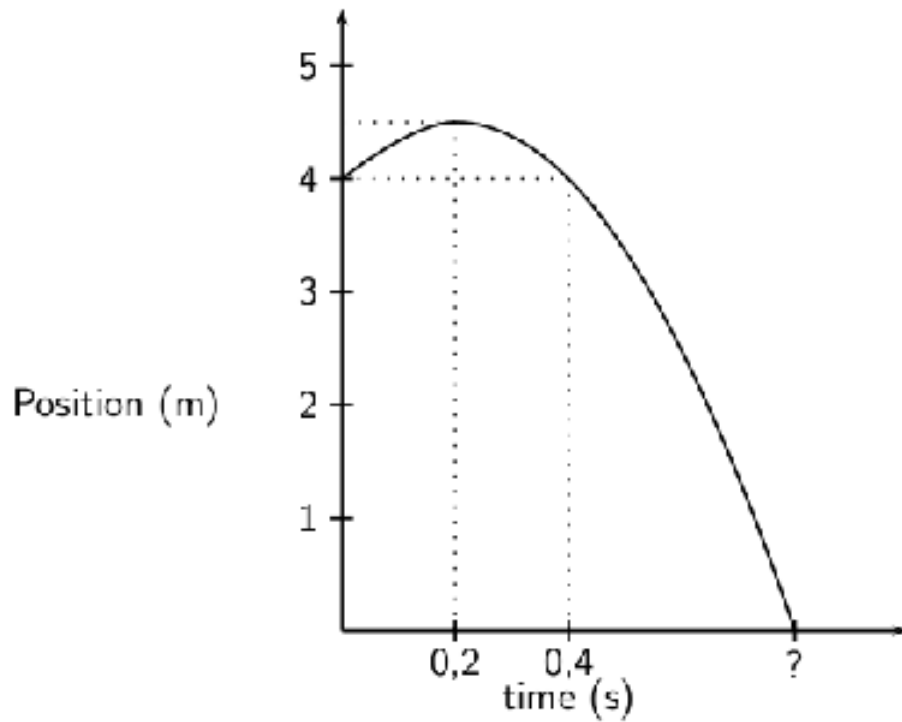


Figure 3.8: Example of a graph [9]

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.

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4
5 def y_displacement(y_offset, initial_speed, theta):
6     increment = 0.001
7     theta = np.radians(theta)
8     x_speed = initial_speed * np.cos(theta)
9     y_speed = initial_speed * np.sin(theta)
10    x_pos = []
11    y_pos = []
12    time = 0
13    x_pos.append(0)
14
15    y_pos.append(y_offset) # Adding the offset in the first slot of
                           # the position array

```

```

16 time += increment
17 x_pos_temp = x_speed * time # Calculating first x value so while
18   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=
33     dict(arrowstyle='<->'))
34 plt.annotate(s="", xy=(0, -1), xytext=(max(x_pos), -1),
35     arrowprops=dict(arrowstyle='<->'))
36 plt.plot([0, x_distance], [0, 0], color='k', linestyle='-',
37     linewidth=2)
38 plt.plot([0, 0], [0, y_offset], color='k', linestyle='-',
39     linewidth=2)
40 plt.plot([-0.5, -1], [y_offset, y_offset], color='k', linestyle='-',
41     linewidth=2)
42 plt.plot([0, x_pos[y_pos.index(max(y_pos))]], [y_offset, y_offset],
43     color='k', linestyle='-', linewidth=2)
44 plt.annotate(s="", xy=(x_pos[y_pos.index(max(y_pos))], max(y_pos)
45     ),
46     xytext=(x_pos[y_pos.index(max(y_pos))], y_offset), arrowprops=
47     dict(arrowstyle='<->'))
48 plt.text(-max(x_pos) * 0.08, y_offset / 2, str(y_offset), style='
49     normal')
50 plt.text(max(x_pos) / 2, -max(x_pos) * 0.05, str(max(x_pos)),
51     style='normal')
52 plt.text(x_pos[y_pos.index(max(y_pos))] + 1, ((max(y_pos) -
53     y_offset) / 2) + y_offset, str(max(y_pos) - y_offset),
54     style='normal')
55 plt.axis([-5, x_distance, -5, max(y_pos) + 5])
56 plt.axis('off')
57 plt.show()
58
59 y_displacement(12, 20, 30)

```

Listing 3.7: Second iteration of graph generation

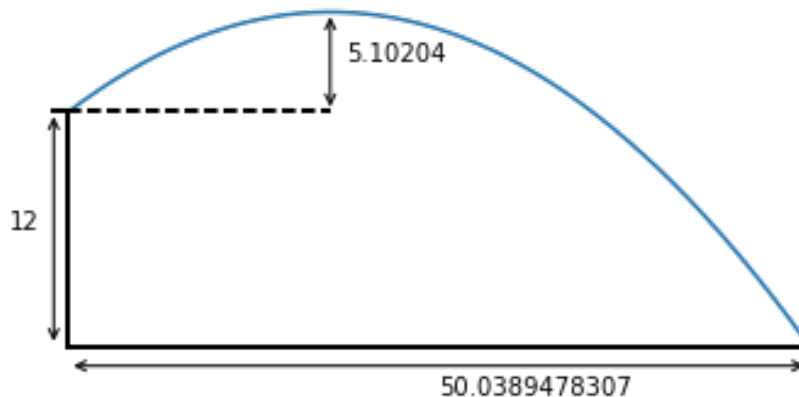


Figure 3.9: Second iteration of graph

As you can see this graph is much more interesting than 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 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 $(0, y_{\text{offset}})$ 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
3                 * np.sin(angle))]
4     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:

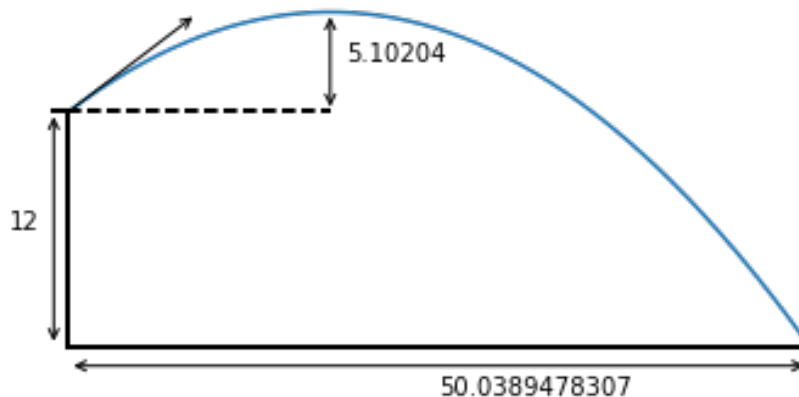


Figure 3.10: Third iteration of graph

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 \LaTeX to generate the symbol. I decided to use \LaTeX as 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 \LaTeX with Python is not trivial. You must first get a working installation of \LaTeX on your computer, which is usually done with MikTeX on a Windows machine. Then you must configure matplotlib to use \LaTeX which is done by adding the lines:

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

Listing 3.9: Requirements for \LaTeX in matplotlib

Then to use \LaTeX to 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.

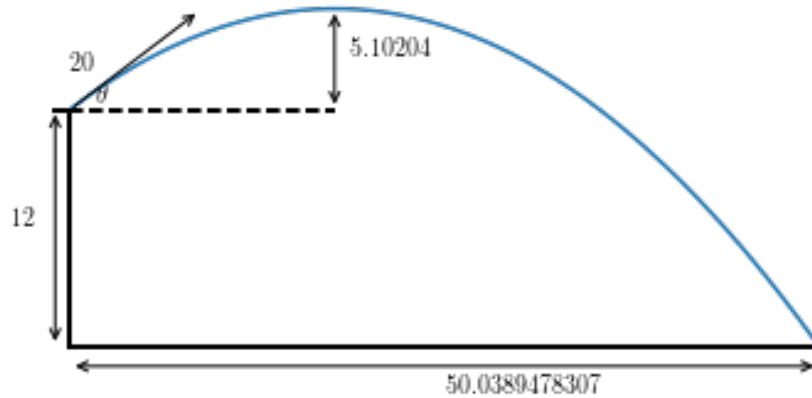


Figure 3.11: Fourth iteration of graph

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

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

Listing 3.11: Requirements for \LaTeX in 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)`.

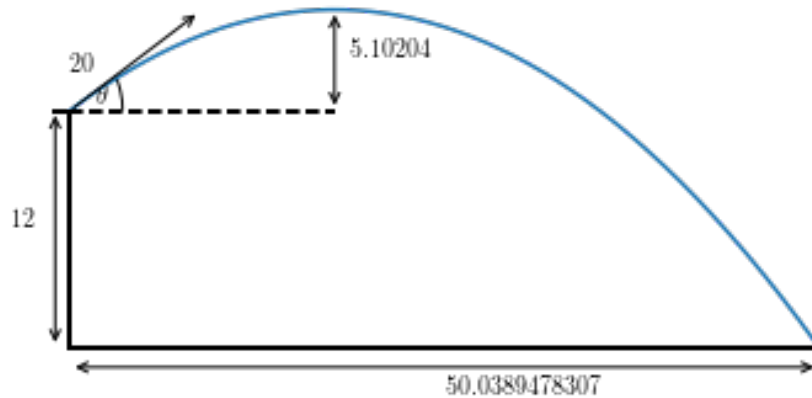


Figure 3.12: Final iteration of graph

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

3.3.4 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.

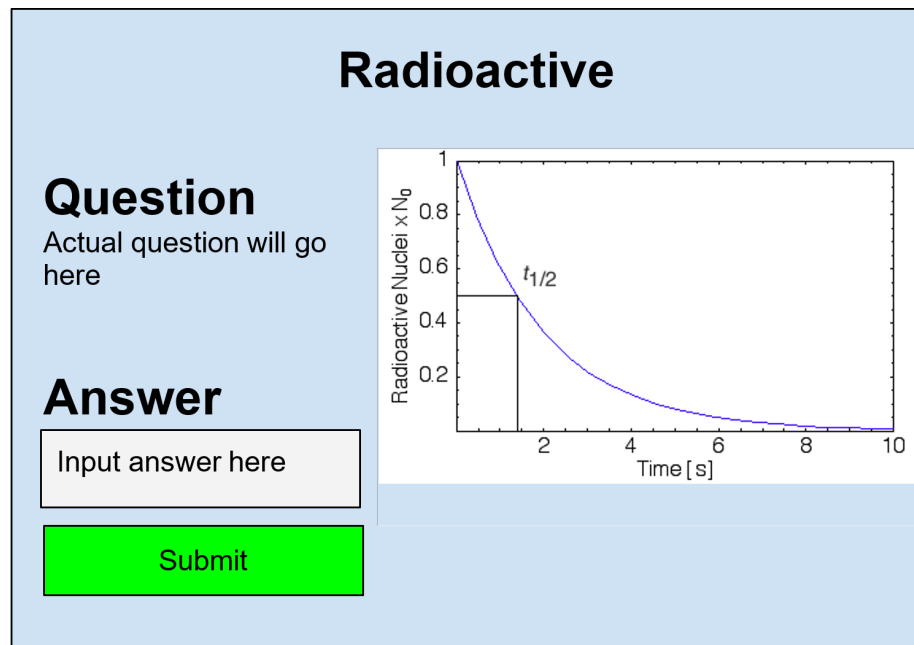


Figure 3.13: GUI Design Prototype

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](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 L^AT_EX*. URL: <http://www.latex-project.org/about/> (visited on 24/02/2017).