Year 8 Mathematics Investigation - The Cartesian Plane and Pythagorean Triples

March 20, 2024

1 INTRODUCTION

It can be argued that the Cartesian plane is one of the greatest mathematical inventions, since in one bold stroke, it unified the world of geometry and the world of algebra by allowing us to easily visualise relationships between quantities. In this investigation, the Cartesian plane will be the stage on which entities move about as you explore their relationships and interactions using well thought-out computational algorithms.

As you go through the algorithm design process, make sure to document your progress in presentation slides. The designs in your slides should be in plain English using brief dot-points (with indentation to indicate blocks of instructions) or you can use diagrams, pseudocode or flowcharts if you prefer. It is important that you demonstrate a deep understanding all formulae that you use – document these in your slides with an explanation of how they work.

You will likely need to make assumptions as you engage in design work. Make sure to document them in your slides (these slides must be saved in the same folder that contains your code).

1.1 Create and explain a reusable means of achieving translations (10 Marks)

This investigation takes place on a Cartesian plane of a specified size. There will initially be three entities on it; `player one`, `player two` and `destination`. When your program first runs, the three entities will be placed randomly on the Cartesian plane. The two players will take turns moving a short distance towards the `destination`. The `destination` will stay put.

The main restriction is that the players can only move along the hypotenuse of a right-angled triangle, and the lengths of the sides of the triangle must form a primitive Pythagorean triple. The non-hypotenuse sides (called `a` and `b`) must be parallel to the x and y axes of the Cartesian plane. The triangle can be oriented such that the players can move roughly towards the top right, top left, bottom left and bottom right of the Cartesian plane.

For example, let us say `player one` wants to move 5 units *roughly* towards the top left corner of the plane. Using the Pythagorean triple (3, 4, 5), they can move 3 units left and 4 units up (a translation of `[-3,4]`) or they can move 4 units left and 3 units up (a translation of `[-4,3]`). Notice the subtle change in direction. If `player two` wants to move 13 units *roughly* towards the bottom left corner of the plane, they can apply a translation of `[-5,-12]` or `[-12,-5]` using the Pythagorean triple (5, 12, 13). *Notice the much more pronounced change in direction between the two translations.*

You need to find a way to translate a player a specified distance in a particular direction. Since

trigonometry is outside the scope of this investigation (do NOT use trigonometry), we are going to specify eight approximate directions numbered 1 to 8. The eight directions are obtained by dividing each quadrant of the cartesian plane diagonally in two. Thus direction number 1 will be from 0 to 45 degrees (0 degrees is in the positive direction of the x-axis and angles increase anticlockwise). Direction 2 is from 45 to 90 degrees, direction 3 from 90 to 135 degrees and so on, all the way to direction 8 from 315 to 360 degrees.

Thus `player one` can request to move 5 units in direction 3 which can be achieved by translating `[-3,4]`, but moving 5 units in direction 4 requires `[-4,3]`. `Player two` moving 13 units in direction 5 needs a translation of `[-12,-5]` while moving 13 units in direction 6 needs `[-5,-12]`. Note that when your program runs, you will simply enter `13 <space> 5` or 13 `<space> 6` to specify distance and direction. Make sure you fully understand how all of this works - discuss it with classmates before you start your design work.

By the way, these translations will be in our imaginations at first. Your algorithms will simply be manipulating coordinates and printing out the locations of the players and the destination. You are not required to visually display the Cartesian plane and players as they move around (this will be an optional extra at the end).

To simplify your task, you will be provided a list with the first 127 primitive Pythagorean triples sorted by the hypotenuse (see section 3.2 for more information). Do not add to this list. To earn full marks in this section, you need to think carefully about how to move a distance of `c` units in direction `n` using a pythagorean triple `(a, b, c)` from the list. Using look-up tables can help, where you can somehow link directions to translations. Your design will eventually be packaged into a reusable Python function that can be called to execute a required translation. Your design work here is central to the entire investigation - make sure to document what you decide to do (and exactly how it works) in your presentation slides. If you choose to implement these conversions another way other than using the look-up tables, describe in your presentation what you did and how it works.

1.2 Use functions, validate input and document imports (10 Marks)

As you work on this investigation, make sure you do not simply import a library (either built-in, or from the internet) and use it to do all of the work for you. For instance, doing the following to calculate distance will not earn any marks:

```
import math
print(math.dist([2,-4],[-5,3])
```

The intent is for you to write your own versions of the functions (in this case, `dist()`) that perform the required operations so that you gain a deep understanding of how to think about, and design computational algorithms. This knowledge will also allow you to be able to understand and explain code that you obtain from sources on the internet as you work on this task.

If you do import some libraries (such as ***`numpy`*** *or* ***`scipy`*** *or* ***`pandas`****), make sure to explain in your slides exactly how you are using them, making it clear that they only play a supporting role and that they do not implement entire parts of your investigation.*

It is essential that you use functions throughout (worth 5 marks) and you must demostrate this during your presentation. You should also validate all input data such as rejecting negative lengths, and directions beyond 1 and 8. Document all the kinds of validation that you do (worth 5 marks).

2 PART I: CREATE AND PLACE ENTITIES

2.1 Represent the two players and the destination as dictionaries (5 Marks)

An efficient design strategy for Computational and Algorithmic Thinking (CAT) is the packaging of information about an entity in one place. You are going to do this using Python dictionaries (you can use classes if you are familiar with them however this is not required and does not attract additional marks).

Thus to this end, you will create three dictionaries; `player_one`, `player_two` and `destination`. The player dictionaries will contain the following information: current coordinates, distance from the destination, midpoint coordinates to the other player, the gradient of a line connecting the player to the destination, and personal space buffer. The destination dictionary should contain its current coordinates and its personal space buffer. Use a buffer of 10 units for all three entities - more on this later. Display your dictionaries in your slides and describe them during your presentation.

Below is an example of how to use a dictionary to store relevant information:

```
person_one = {
    'name': 'Rene Descartes',
    'birthplace': [47.3, 0.67],     # Approx Long and Lat
    'trips_around_the_sun': 53,
}

print(person_one['name'], "was born", person_one['birthplace'][1],"degrees north.")
```

If you need more practice with dictionaries, consult Grok Academy's Introduction to Programming 2 (Python) Course, module 2. Document how you designed your three dictionaries in your slides.

2.2 Randomly place the players and the destination (5 Marks)

The size of the cartesian plane on which the program events unfold will be from -800 to +800 in both the x and y directions. When your program runs, it should generate three sets of random integer coordinates and assign them to `player one`, `player two` and the `destination` by storing them in the relevant dictionaries. You should import and use the Python `random` module to achieve this and document your work for this section in your presentation slides.

2.3 Calculate distance, midpoint and gradient (10 Marks)

Since you need to know the distance and gradient (both to 1 decimal place) of each player to the destination, as well as the midpoint coordinates of the two players, create functions that calculate these. Store the outputs of the functions in the relevant dictionaries created above. You will be calling these functions repeatedly as is about to be explained shortly. Remember not to simply import modules and use their functions - write your own from scratch. The approach and formulas you use need to be evident and documented in your code and slides.

2.4 Display information about players and the destination (5 Marks)

You will need to print current information about the players and the destination as necessary during gameplay. You therefore need to write a function that prints the following information about `player one` when called:

```
PLAYER ONE Location: (-90, 112)

Distance to destination: 385.1 units

Gradient with destination: 2.5

Midpoint with Player Two: (10.4, -70.8)
```

You should have another function to do the same for player two. For full marks you should reuse one function to print information about each player as required. Document how you do this in your slides. Finally, write a function to print information about the destination location.

...

Some hints: You might find it useful to use Python's f-strings as shown below when summarising.

```
name = 'Leonhard Euler'
num = 2.71828
print(f"{name}'s favourite number is {round(num,2)} (2 d.p.)")
```

Make sure to investigate Python's rounding and check that it works as you expect.

By the way, when using VS Code or PyCharm, you can print statements in colour by first installing `termcolor` using `pip install termcolor`. Below is how you use its `cprint()` function:

```
import termcolor
termcolor.cprint('hello world','red')
termcolor.cprint('in colour!','blue')
```

Printing in colour is optional and does not attract any additional marks.

3 PART II: TAKE TURNS MOVING PLAYERS

3.1 Use the personal space buffer to identify a winner (5 Marks)

As you've already worked out, the aim of the program is for the two players to take turns and compete to see who can reach the destination first. A player is deemed to have reached the destination if their coordinates are within the personal space buffer of the destination. You should have a function that can be called to check for this and indicate that the player has won. You will need to think about how you can ascertain when a player has reached the destination's personal space and document your thoughts and designs/algorithm in your slides.

Since each human player can see information about the other player's location and the midpoint coordinates between them, a player can move towards another. If a player reaches another player's personal space buffer, the player who made the move wins. It is therefore important for players to avoid each other as they advance towards the destination. Once again, document your design work for this in your slides and make note of any ways that you have reused your previous computational algorithms.

3.2 Move players along the hypotenuse of a right-angled triangle (10 Marks)

Attached to this investigation is a file containing a list of 127 primitive Pythagorean triples. Do not add to this list. The list (of lists) looks like this:

`triples = [ [3, 4, 5], [5, 12, 13], [8, 15, 17] ... [555, 572, 797] ]`

The list has been doubly sorted for you, in ascending order by the length of the hypotenuse, and within each triple, the lengths are also in ascending order so that the last number is always the hypotenuse. Keep this in mind as you design your algorithms. Below are some examples of how you can query the list:

```
print(triples[2])      # prints the third triple


# to loop over each triple
for triple in triples:
    print(triple)


# to access only the hypotenuses...
for triple in triples:
    print(triple[2])


# to access only the hypotenuse of the second triple
print(triples[1][2])
```

Remember that a player can effectively move along the hypotenuse `c` by using a translation involving `a` and `b`. It is therefore unlikely for a player to take a straightforward path to the destination - it will have to be a zig-zag path that sometimes doubles back. Clearly, not every integer length from 1 to 800 is available. Thus if a player requests to move distance 15 in direction 2, the requested distance should be interpreted as 'at most 15 units' meaning you should pick the closest hypotenuse that is at most 15. In this case, allow them to only move 13 units in direction 2.

You should therefore write a function that accepts the requested distance and direction, and only move the player the allowed distance in the requested direction. Remember, the actual translation that achieves this goal is `[5,12]`. Detail your thoughts and design work in your slides and make sure you can explain your algorithm.

After a player moves, your program should call the relevant function and display their updated information, as well as the destination's information (hopefully in different colours). The second human player can make use of this knowledge during their turn.

4 PART III: ADVANCED FEATURES

4.1 Introduce a time penalty (1 Mark)

Players should not have unlimited time to make a move. If a player takes more than 10 seconds to decide, your program should pick a random Pythagorean triple instead, as well as a random direction from 1 to 8. This information should override the late player's request if they've made one. You can import the Python `time` module to help you and you are not required to implement a countdown, although you can if you wish. Make sure to document your design decisions.

4.2 Design and implement player three as an NPC (5 + 2 Marks)

You can choose to think about how to design an algorithm for the computer to control a Non-Player Character (`player three`) so that they also get a turn during gameplay. Remember, *a plausible algorithm (even if not coded) will be awarded 5 marks*, therefore you can give details in your slides and convince your classmates that your NPC algorithm will work.

4.3 Visual depiction of gameplay (2 Marks)

You can make the program more visually appealing by providing a visual representation of the cartesian plane and how the players are moving after each turn. This can be implemented using any method you choose as long as it uses Python code. The recommendation is to use Pygame as it will simply display the data that you have already computed as you worked on this investigation.

5 PART VII: DOCUMENT YOUR CODING PROCESS WITH FREQUENT AND DETAILED COMMITS (30 MARKS)

Once you complete the design process, you are required to implement it in code using the Python programming language. You are expected to use code comments and functions extensively and there will be marks allocated for these. Consult the year 8 CAT resources in Compass under School Documentation | Curriculum and assessments | Mathematics (more in in Grok Academy's "Introduction to Programming 2" intermediate course, module 7).

Do NOT use Grok to write code for this investigation. You must use VS Code (or PyCharm) and you need to have installed Git so that you can document your coding process with frequent commits ('diary entries'). Writing a separate diary of your progress elsewhere other than by 'making commits' as instructed will not be awarded any marks. If any of these instructions are unclear, make sure to contact Mr. Kigodi during term 1 week 9.

Before you start coding, create a new folder called Y8 CAT Inv1 Name Surname (with your name and surname), open it in VS Code or PyCharm and initialize a new repository (follow the instructions in the year 8 Compass CAT resources). Create a main.py file to hold your code and save your design slides into the same folder that contains main.py. You should then immediately make your first commit with a diary entry saying you have just created a new project with your design slides and a blank file to hold your code. Do not move files or delete them once you have initialised a repository – if you are not careful, you might erase your commits (and lose the 30 marks).

As you code, debug and test what you have written, and once you have made a notable addition that works (such as a welcome message or a new function), commit it with a detailed description of what you've done. Continue adding code as per your design, and each time you write enough

lines to make a notable change, make another commit with a detailed description. If you make a mistake or change your mind about some of your code or design, make the necessary changes and commit again with a description of what you've done. You will not be penalised for this – just make sure to adjust or refine and commit your design slides to reflect any new ideas you come up with as you code (this is normal). The code and the design in your slides must match.

The code at each commit checkpoint needs to be able to run and you should be able to describe the features that the code implements up to that point. Remember that frequent and detailed commits are required to earn the marks in this section (Git records the date and time of each commit). Inserting tens or hundreds of lines of code per commit will not meet the requirement. Essentially, your commit history should make it very clear that you have been gradually working on your investigation and that you understand intimately the code that you are committing. Remember that you will need to show your commit history when presenting to a random group of classmates.

6 SUBMIT YOUR WORK

The due date for the slides and the code produced for this investigation is Friday 26 April 2024 (end of term 2 week 2). Make sure your presentation slides are in the same folder that contains your code and make the final commit by the due date. The code and slides in this final commit are what you will present to your classmates and the timestamp will be used to confirm that the submission was made on time.

In addition to making the final commit, you will also upload your code and slides to an online submission form that will be made available the week the investigation is due. Full submission instructions will be issued in week 1 of term 2.

During week 3, you will do a 12-minute presentation of your computational and algorithmic thinking processes to a random group of peers with teacher supervision. You will use VS Code (or PyCharm) to present your code and gameplay. You will also need to showcase the slides saved during the final commit documenting the process you went through and the designs you generated. Email isaac.kigodi@education.wa.edu.au or come to the Maths office if you need assistance. You can type into VS Code (and run) `import this` to get additional guidelines on what constitutes good Python code.