Assignment: SPACE MOVER
Client's Briefing Part 2 – Transcript

Hello again. By now you should have chosen and drawn a symbol of your choice, of a size to fit exactly over the top of a cell on the supplied Turtle canvas. The code to produce your symbol should be encapsulated in a function that can be called as required, in order to produce that symbol anywhere on that Turtle canvas' grid.

This part of the specifications contains the remainder of your client's requirements.

Part 2

Your client is interested in visualising an entity moving around in a confined space. This confined space is represented by the supplied Turtle canvas' grid.

When you run our Python template you will notice that one of the cells is marked by a dot.

[Runs the template file from the last briefing]

This cell, in position D1, marks the starting location of the entity (i.e., your drawn symbol). Instructions supplied by the data generation module will indicate the path that the entity should travel from the starting location.

To provide you with data sets to visualise, our data analysts have created a Python module called `config.py` which you can find accompanying this briefing. You are welcome to study this code, but do not change any of it.

[*Displays the* `config.py` *module*]

If you put this module into the same folder as the original Python template you have been working on, it will cause the program to generate a new data set each time it is run.

[*Runs the template file a few times with the data module in place*]

As you can see, a different data set is printed to the shell window each time. These are the data sets of interest to our analysts, but in this form they're very hard to understand. It's not immediately obvious from the sequence of individual moves where the entity goes, and where it ends up on the grid.

Your job is to create a visual representation of the behaviours encoded in the data sets, using the symbol you have already created for us, so that we can see clearly the path it travels and where it finishes.

Each data set is a list of moves, with each move itself being a list containing a number and a string. These values tell you how the entity should move. Let's look at an example.

[*Shows and explains a typical example of one of the data sets*]

In summary, therefore, each data set is a list of zero or more instructions to move the entity. Each instruction contains a number representing how many cells to move, and a string representing the direction in which to move.

The movement of the entity is restricted by the following rules:

- the entity must be drawn at every cell it moves to, while following an instruction;
- apart from in the legend, the entity may only move around inside the grid;

- If a move would otherwise cause the entity to move outside the grid, that move should be ignored;
- the entity must only be permitted to visit perimeter cells of the grid a maximum of 10 times; (the perimeter is defined as any cell on the grid in column: 1 or 7, or in row: A or G)
- a count must be displayed on the right hand side of the grid, showing the number of visits to cells on the perimeter;
- if the maximum number of visits to perimeter cells is reached, the processing of all further instructions is to be aborted;
- if the processing of further instructions is aborted, a suitable message is displayed above the grid;
- if all instructions are able to be processed without aborting, a different message should be displayed above the grid to indicate this.

In every case the entity starts at cell D1 and then moves as far as it can given the instructions in the data set and the above restrictions.

It is often extremely difficult to predict the path from the instructions, which is why your visualisation is so important to help our data analysts interpret the data sets.

To show you what's expected our back-room boffins have extended their sample solution, so I'll run their demonstration on a few different data sets.

[*Runs demo with empty data set*[1]]

As you can see, this data set contained NO instructions, so the entity did not move at all.

[*Runs demo that processes all instructions, with NO attempt to move outside grid*[2]]

If the instructions are to move to a cell that already has the entity on it, it makes no difference. The entity must be drawn again regardless. The entity drawn over the top must completely obscure that drawn underneath.

[*Runs demo that processes all instructions that include attempt to move outside grid*[3]]

If a step in the instruction is to move to outside the bounds of the grid, that step must be ignored. Processing continues with the next instruction, if any.

[*Runs demos that abort due to too many perimeter moves*[4]]

Regardless of how many instructions have been processed, if there is an attempt to exceed the maximum number of steps on the perimeter or in fact if there are ANY instructions following the maximum perimeter count being reached, the program should end, with a suitable message displayed on the canvas.

In summary, therefore, your updated program must implement the following behaviours. Rather than attempting to do everything at once, our boffins tell me that you'll find the job much easier if you tackle one thing at a time.

[1] Seed 31
[2] Seed 15
[3] Seed 18
[4] Seeds 80, 36

1. Before processing the first move in the data set, you must draw the entity in its home cell, marked by the dot, in D1.
2. Before processing the first move in the data set you should also make a note that the entity has so far stepped on the perimeter one (1) time – as the cell D1 is on the perimeter. This perimeter count must be displayed on the right hand side of the grid where indicated. Each further time the entity steps on a perimeter cell, the number displayed must be updated. That is, the total number of times the perimeter has been stepped on should always be displayed.
3. For each instruction, you must draw the entity in each cell in that path. It should not move if it is currently on the perimeter and instructed to move out of the grid.
4. You must stop processing moves if there is an attempt to exceed the maximum number of steps on the perimeter or if indeed the maximum number of steps has been reached.
5. A final message must be displayed on the Turtle canvas above the grid to indicate if all moves were processed, or whether they were aborted.

To help you, each data set generated is printed to the shell window, as you have just seen. More importantly, however, the data set is also returned as a list by function `actions`, which makes it available to your `start_moving` function.

[*Shows the call to* `actions` *in the template's main program*]

This function call returns the same list that is printed in the shell window, so you will use the value returned by `actions` as the basis for your visualisation in function `start_moving`. You must not write code to call the function `actions` itself, because our template already does so.

Furthermore, so that you can access the list returned by `actions`, all of your code must go in, or be called from, function `start_moving`. Do not put code anywhere outside the template's "Student's Solution" section and do not change any of the code in any Python file given to you, except as explicitly allowed by the comments.

Overall, therefore, your solution needs to do a job equivalent to our demonstration, using the symbol you created for us previously. See the Part 1 client briefing.

Finally, I fully appreciate how difficult it is to work with large data sets that change randomly. To help you develop your program with some unchanging data sets, our boffins have therefore incorporated a feature into the Python template that allows the data sets to be fixed temporarily. If you add an integer argument in the call to function `actions` in the main program it will always generate the same data set. For instance, using seed value 30 produces a data set with many instructions.

[*Shows how to use Seed 30 to produce this behaviour*]

In this instance the path the entity followed took it across the grid to the eastern edge then aborting before processing all instructions because the maximum perimeter count had been reached.

It's also a complicated example because it backtracks a few times and attempts to move outside the grid too, so our analysts may want to study this particular data set in detail. Fortunately, by supplying the seed value 30 to the function `actions` we can replay this path as often as we like.

[*Runs the program using Seed 30 again*]

In the `config.py` module we've listed many such numbers you may find helpful because they produce data sets having already known behaviours.

[*Shows the "seed" values documented in the data source module*]

Of course, when you submit your finished program, it must work for any random list that can be generated by function `actions`. The marker will test your program with no argument in the call to function `actions`.

[*Removes the seed value in the call to* `actions` *and runs the program again*]

When you've finished your program, please upload it to our organisation's IT system. We only need the single Python file that you have completed, that is, the supplied template `space_mover.py` with your solution added in the appropriate place. Do not submit the `config.py` module; we already have it and will use our own copies to test your code.

Also, you are encouraged to submit multiple drafts as you develop your solution, as insurance against technical failures, floods, pandemics, and so on leading up to the deadline. If you can't complete the whole job in time, submit what you have, potentially for part marks.

That's all for this set of requirements. Our organisation is looking forward to receiving your updated program at your earliest convenience.