Assignment 1, Part B: Scorecard

(5%, due 11:59pm Sunday, September 9th, end of Week 7)

Overview

This is the second part of a two-part assignment. This part is worth 5% of your final grade for IFB104. Part A which preceded it was worth 20%. This part is intended as a last-minute extension to the assignment, thereby testing the maintainability of your code from Part A and your ability to work under time presssure. If you have a neat, clear solution to Part A you will find completing Part B easy. For the whole assignment you will submit only one file, containing your combined solution to both Parts A and B, and you will receive one grade for the whole 25% assignment.

Motivation

One of the most common tasks in "Building IT Systems" is modifying some existing code. In practice, computer programs are written only once but are subsequently modified and extended many times during their operational lifetime. Code changes may be required in response to internal factors, such as the need to correct design flaws or coding errors, or external factors, such as changes in consumer requirements. A common situation is where some new feature must be added to an existing program. This is the scenario simulated in this part of the assignment.

This task requires you to extend your solution to Part A of the assignment by adding an additional feature. It tests:

- Your ability to work under time pressure; and
- The quality and clarity of your code for Part A, because a well-written solution to Part A will make completing this part of the assignment easy.

Goal

In Part A of this assignment you were required to develop a program which could draw a "treasure map" by following a set of steps encoded as a randomly-generated Python list. The "treasures" found along each path were represented by five "tokens". In this part of the assignment you will extend the treasure map to include a summary of how many tokens were found. Specifically:

- For each type of token, from Token 0 to Token 4, inclusive, you must display on the map how many tokens of this type were found along the current path displayed; and
- You must display the total number of tokens found along the current path.

To complete this additional task you must extend your solution to Part A. You have a free choice of how to display this additional information about the treasure hunt as long as it is clear and neat. No additional Python template file is supplied for this part. As per Part A, your Part B solution must work for any randomly-generated path that can be returned by the provided function `random_path`.

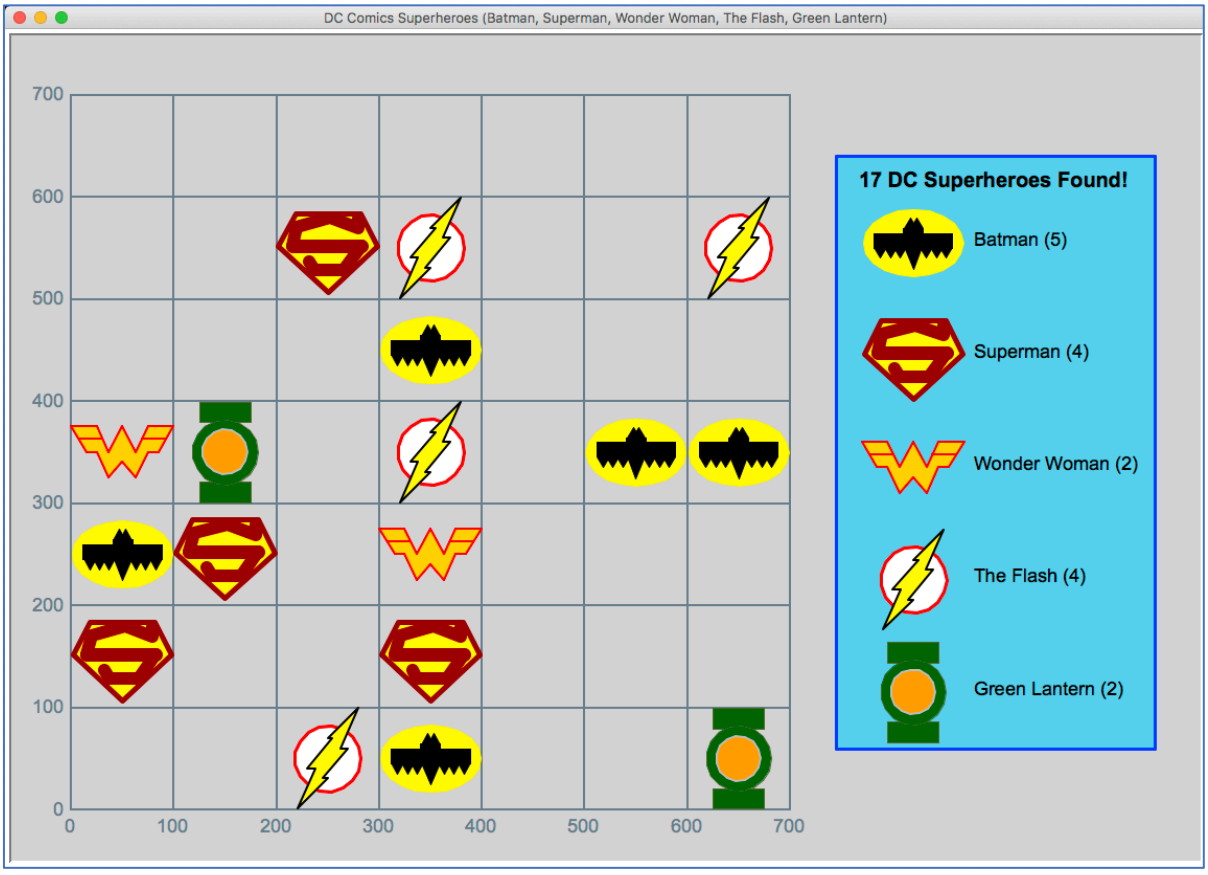
Illustrative example

To illustrate the requirement we'll continue our example from the Part A instructions. Recall that we developed a solution which drew five styles of token inspired by DC superheroes. To show the number of tokens found we have chosen to extend our "legend", although this is not the only possible approach.

For instance, consider the following data set, generated by calling function `random_path`.

```
[['Start', 'Bottom right', 4], ['North', 3, 0],
 ['West', 1, 0], ['West', 2, 3], ['South', 3, 0],
 ['West', 1, 3], ['North', 5, 1], ['East', 4, 3],
 ['West', 3, 3], ['South', 1, 0], ['South', 3, 1],
 ['West', 3, 1], ['North', 2, 2], ['East', 1, 4],
 ['South', 1, 1], ['West', 1, 0], ['East', 3, 2]]
```

Having drawn this path on the treasure map, our extended solution also displays the total number of tokens found, and the number of each token found, as part of the legend on the right.

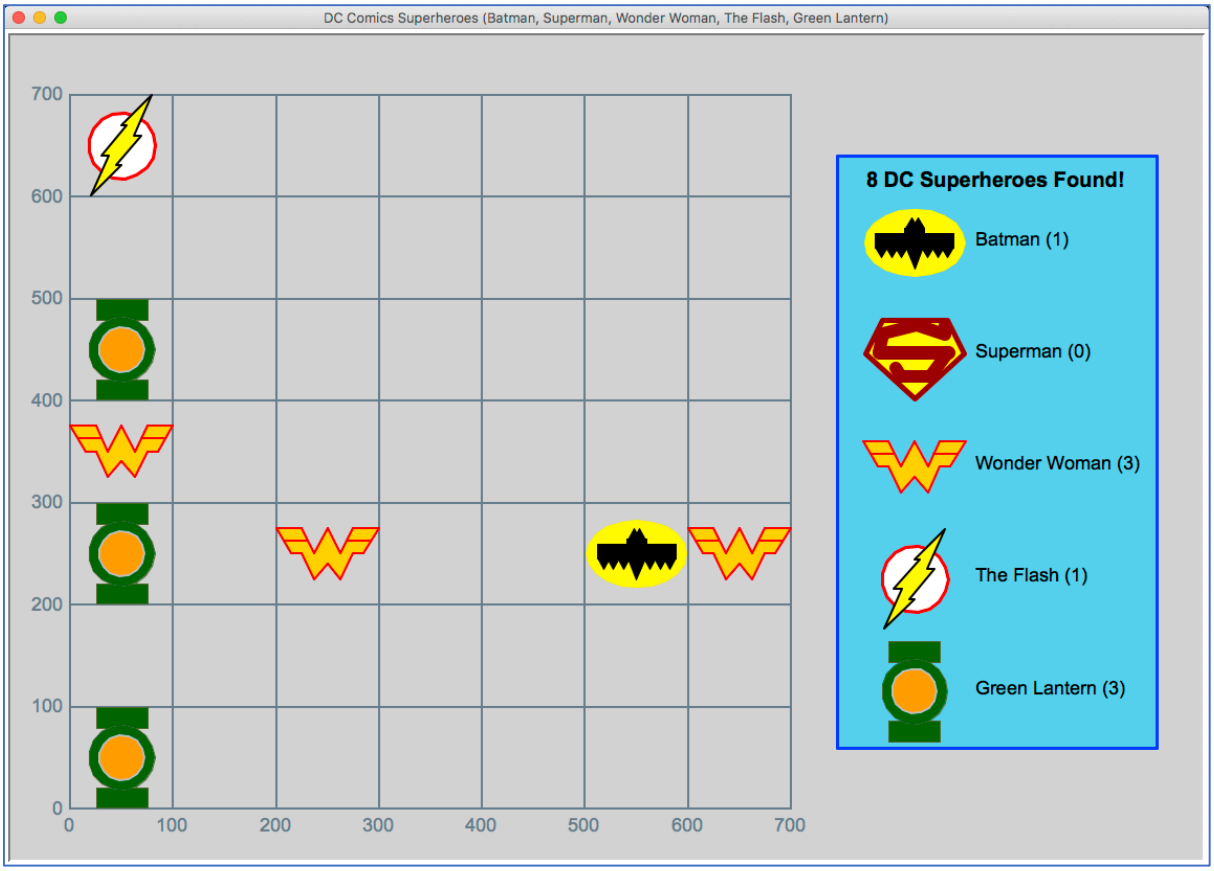


In this case it shows that the path followed encountered 17 superheroes in total, with five instances of Batman, four of Superman, and so on.

As another example, consider the following much shorter randomly-generated path.

```
[['Start', 'Top left', 3], ['South', 3, 2],
 ['North', 1, 4], ['South', 4, 4],
 ['North', 2, 4], ['East', 6, 2],
 ['West', 4, 2], ['East', 3, 0]]
```

In this case only eight tokens were found in total and no 'Superman' tokens were encountered at all.



Note that even though Superman was not found on this particular path his emblem is still displayed in the legend so that we can still tell which symbol is Token 1. (You may choose to visually show that a token was not found at all, e.g., by striking through it, but you should still display all five tokens so that the treasure map's reader can tell which symbol is "Token 0", which is "Token 1", etc.)

Requirements and marking guide

To complete this task you are required to extend your Part A `treasure_map.py` file by modifying the code so that as well as displaying the path followed and the legend, the numbers of tokens encountered are displayed.

Your submitted solution for both Parts A and B will consist of a *single Python file*. Your Part B extension must satisfy the following criterion. Marks available are as shown.

1. **Total and individual token scores are displayed clearly and correctly (5%).** Your program must be capable of displaying on the treasure map both the numbers of each token type found on the current path, as well as the overall total number of tokens found. This data must be displayed clearly, neatly and accurately on the treasure map (and not, for instance, merely printed to IDLE's shell window).

You must complete the assignment using basic Turtle graphics and maths functions only. You may not import any additional modules or files into your program other than those already included in the given `treasure_map.py` template.

Development hints

- It should be possible to complete this task merely by *adding* code to your existing solution, with little or no change to the code you have already completed.
- If you are unable to complete the whole task, just submit whatever part you can get working. You will receive *partial marks for incomplete solutions*. Try to ensure that your program runs when submitted, even if it is incomplete.

Deliverable

You must develop your solution by completing and submitting the provided Python 3 file `treasure_map.py` as follows.

1. Complete the "statement" at the beginning of the Python file to confirm that this is your own individual work by inserting your name and student number in the places indicated. *We will assume that submissions without a completed statement are not your own work!*
2. Complete your solution by developing Python code to replace the dummy `follow_path` function. You should complete your solution using only the standard Python 3 modules already imported by the provided template. In particular, you must *not* use any Python modules that must be downloaded and installed separately because the markers will not have access to these modules. Furthermore, you may *not* import any image files into your solution; the entire image must be drawn using Turtle graphics drawing primitives.
3. Submit *a single Python file* containing your solution for marking. Do *not* submit multiple files. Only a single file will be accepted, so you cannot accompany your solution with other files or pre-defined images. **Do not submit any other files! Submit only a single Python 3 file!**

Apart from working correctly your program code must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant code segments and *helpful* choices of variable and function names. *Professional presentation* of your code will be taken into account when marking this assignment.

If you are unable to solve the whole problem, submit whatever parts you can get working. You will receive *partial marks for incomplete solutions*.

Security warning and plagiarism notice

This is an individual assessment item. All files submitted will be subjected to software plagiarism analysis using the MoSS system (http://theory.stanford.edu/~aiken/moss/). Serious violations of the university's policies regarding plagiarism will be forwarded to the Science and Engineering Faculty's Academic Misconduct Committee for formal prosecution.

As per QUT rules, you are not permitted to copy *or share* solutions to individual assessment items. In serious plagiarism cases SEF's Academic Misconduct Committee prosecutes both the copier and the original author equally. It is your responsibility to keep your solution secure. In particular, **you must not make your solution visible online via cloud-based code development platforms such as GitHub**. Note that free accounts for such platforms are usually public. If you wish to use such a resource, do so only if you have a *private* repository that cannot be seen by anyone else. For instance, students can apply for a *free* private repository in GitHub to keep their work secure (https://education.github.com/pack). However, we recommend that the best way to avoid being prosecuted for plagiarism is to keep your work well away from the Internet!

How to submit your solution

A link is available on Blackboard under *Assessment* for uploading your solution file before the deadline (11:59pm Sunday, September 9th, end of Week 7). You can submit as many drafts of your solution as you like. You are strongly encouraged to *submit draft solutions* before the deadline as insurance against computer and network failures. If you are unsure whether or not you have successfully uploaded your file, upload it again!

Students who encounter problems uploading their Python files to Blackboard should contact the *IT Helpdesk* (ithelpdesk@qut.edu.au; 3138 4000) for assistance and advice. Teaching staff will *not* answer email queries on the weekend the assignment is due, so ensure that you have successfully uploaded at least one solution by close-of-business on Friday, September 7th.