**Unit 4 Project - Mars rover exploration**

# Instructions

The distance between Mars and Earth makes it difficult for humans to study the planet. In an effort to solve this problem, The Space Terrain Advancement Reserves (STAR) has sent a series of unmanned vehicles called “rovers” to explore the Martian surface collecting data and samples. In this project, you will use graph algorithms to help STAR explore the Martian surface by programming its rovers to find an optimal path between each of STAR's areas of interest.

### Deliverables:

* Checkpoint, due 4/15
* Project Submission, due 4/21
* TA Meeting, by 4/24

### Evaluation Criteria:

Your submission will be graded according to the following rubric:

| Checkpoint | 10% |
| --- | --- |
| Decomposition | 15% |
| Implementation | 50% |
| Explanation | 25% |

## Teamwork and Roles:

This project is to be completed individually. You may discuss any aspects of the project with course staff. You may discuss the project at a high level with classmates, but you should not share code or answers to specific questions.

### Use of genAI:

* **Category 1 - Free use of AI tools**
  + You are encouraged to use generative AI to help with this project, especially to generate Python functions from your own descriptions. You can find more details below.

## 

## Understanding The System

The two main classes you will be using are hidden - you will need to use them, but you will have to use the APIs provided below instead of reading through the code yourself. Trust that the functions work as described!

The PlanetIntel class holds all of the information about the planets you are exploring, but not all of the information is available to you. The PlanetIntel class has the static functions get\_planet\_1(), get\_planet\_2(), and get\_planet\_3(), which will return 3 Planet objects that you will need to explore. Planets are represented as 2-dimensional grids, with different types of terrain occupying different cells on the planet.

Your main tool for exploring planets is the Rover class, which has the following methods:

| \_\_init\_\_(self, planet, battery\_life=20) | **constructor**: initializes a rover on the provided planet with a default battery life of 20. The rover's initial location will be the home location of the planet.  In order to create a Rover with unlimited battery life, you can pass in math.inf as the battery\_life param (note that you must first import math). |
| --- | --- |
| get\_current\_location(self) | **returns:** the terrain type at the current location, represented as a character |
| get\_battery\_life(self) | **returns:** the charge remaining on the battery. This will be a non-negative integer. |
| move(self, direction) | Attempts to move in the specified direction.  **direction:** 'N', 'E', 'S', or 'W', representing North, South, East, or West.  **returns**: a tuple of (bool, String), from the following options:   * (True, "Success") - the rover successfully moved in the requested direction * (False, "Invalid Direction") - an input other than 'N', 'E', 'S', or 'W' was provided * (False, "Obstructed Space") - the rover can not currently move in the requested direction. This could be because of an object in the way, or a map boundary. * (False, "Drained Battery")   If the first element of the tuple is False, the rover did not move, and is still in the same location from before move was called.  The rover's battery power will be reduced by 1 only if it successfully moves. |

### Creating Test Planets

Throughout the development process of this project, you may find it useful to create a test planet, so that you can know the layout of the terrain in advance or test specific features. To do this, you can construct a test planet by passing in a list of Strings to PlanetIntel's createTestPlanet function, like so:

| rows = [  "...w",  "X.XX",  "....",  "H...",  "...."  ]  test\_planet = PlanetIntel.create\_test\_planet(rows) |
| --- |

This creates a planet whose layout is a 5-by-4 grid (5 rows by 4 columns). The current system uses the '.' character to represent an empty space, a 'w' character to represent water, an 'H' character to represent the home base of the rover (where it will be initialized), and the 'X' character to represent an obstruction. Note that these are case-sensitive.

## Checkpoint: Mapping the Martian Surface

Your first task is to map the surface of planets 1, 2, and 3. Implement the map\_surface function in main.py using a single rover object. It should return a 2-dimensional list (a matrix) of characters: '.', 'w', 'H', and 'X' as specified above, and '?' for potentially unknown spaces.

Some other things to keep in mind:

* You may create any additional helper methods or classes that would be useful
* You must display the result of the map\_surface function in a meaningful way so that you can show the surface layout visually, such as printing the map's characters to the terminal in a grid format
* You may only initialize one rover within the map\_surface function
* It is perfectly reasonable, for this checkpoint, to implement a naive algorithm, such as randomly moving 100,000 times and mapping as you go along. The checkpoint is meant to make sure you can use the rover's functions properly to get meaningful information from the planet.

For debugging / verification purposes, this is how Planet 1 is created:

| def get\_planet\_1():  rows = [  "..X..",  "X....",  "..X..",  "ww.H.",  "w...X"  ]  return Planet(rows) |
| --- |

**For your checkpoint, you must turn in:**

* An implementation of the map\_surface function according to the above specifications
* Maps of planets 2 and 3 (in a .txt file added to your submission, or similar)
* A decomposition diagram for Project Submission: Range Constraints below

## Project Submission: Range Constraints

Implement a new function map\_surface\_with\_battery\_constraint that maps the surface of a planet, but this time the rover only has a battery range of 20. For this section, you must do better than repeated random walks.

The battery logic is already handled within the Rover class. Every time the rover successfully moves, its battery life is reduced by 1. It can't move if its battery is at 0. **Whenever the rover returns to the home location, its battery is restored to its full initial value.**

For this section, your algorithm cannot use knowledge of the layout of the planets you discovered in the checkpoint, nor from previous times running your program. Your function must map the surface to the best of its ability from scratch, so that it could even map a hypothetical 4th planet if asked.

You must turn in:

* A decomposition diagram
* A full implementation

**Note:** It is okay if you are not able to fully map the largest planets. It may not even be possible! Be able to defend your algorithm in the meeting with your TA. We won't tell you exactly how much of Planet 3 you need to explore, but you should be able to explain to your TA why you selected that algorithm and why you believe it's sufficient.

**Another Note:** It is up to you whether or not the robot needs to return to the home base at the end.

***Hint:*** *What if you go out one space, and then return? You can map all the spaces one away from your home base. Now can you do all the spaces within 2? Then 3? When you reach a new space this way, based on the rules of BFS, you've found the most efficient path to that space.*

***Efficiency Improvement Hint:*** *What if you go out as far as you can with enough battery life to return to recharge? You've learned a lot and now you can explore out to that point more efficiently next time.*

## Extra Credit: Research Questions

You can earn up to 5 points of extra credit per research question (maximum of 10 extra credit points).

Propose 1 or 2 additional research questions and answer them by writing programs. Here are some possible research questions:

* What is the minimum battery life needed to map each of the planets' surfaces?
* Design and implement a way to animate your rover's movement (in the terminal, no graphics library needed) so that you can watch your algorithm in action
* Can you design a more optimal algorithm for the battery-constrained robot?
* Can you select the ideal home base location for a given planet, based on some metrics of your design (e.g. the location with the most water within X distance)?

You can also propose your own research questions, but they must be non-trivial (ask an instructor or TA). As a general rule, answering your question should require using one of the graph algorithms from class.

The PYC files were created with Python 3.13.3