National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2022/2023

Side Quest 8.1
To Infinity and Beyond!

Release date: 19th September 2022
Due: 4th October 2022, 23:59

Required Files

- sidequest08.1-template.py
- planets.py

This mission consists of **two** tasks.

Introduction

You are working in a space agency and are appointed as the engineer to plan out the route for the curiosity probe to visit Mars. Now that you have learnt so much about Python, you know that this will be an easy feat!

Firstly, let's plan the space flight. Gravity will be the main driver of the object motion in space, as it is not feasible to carry a large amount of fuel there. Thus, we have to depend on the gravity assist of planets [1] in space to alter the path and speed of your spacecraft in order to reach your desired destination.

Mathematics

The force of a planet acting on a spacecraft can be defined by the following formula:

$$F = \frac{G \times M \times m}{r^2} \tag{1}$$

where F is the gravitational force, G is the gravitational constant, M is the mass of the planet, m is the mass of the spacecraft, and r is the distance between the planet and the spacecraft.

By knowing the force acting on the spacecraft at each moment in time, we could effectively plot the trajectory of a spacecraft and determine where and how fast a spacecraft should be launched from Earth.

To solve this problem, we will use numerical methods, which is to break the task into small time components, and compute the values we need at each time point. For example, we can consider a simple case below:

At time = 0, Russell is standing at position 0. Given that his speed at that time is $1m/s$, and his acceleration is $2m/s^2$.

[1] A more accurate term to use would be *celestial bodies* since the moons of planets might be involved in the gravity asssist too. However, for expedience, we will refer to them collectively as planets.

```
position => 0
speed => 1 m/s
acceleration => 2 m/s^2
```

We can then approximate Russell's position and speed at the next second by:

```
position => old_position + time * speed
         => 0 + 1 sec * 1 m/s
         => 1

speed    => old_speed + time * acceleration
         => 1 m/s + 1 sec * 2 m/s^2
         => 3 m/s
```

If we know the acceleration of Russell at this new position, we can compute his position and speed for the next second. Hence, we can plot out the path of Russell through this method!

This could be done for the spacecraft's motion as well, by considering the motion in two perpendicular axes (namely, x and y). We can keep track of all this variables in a vector:

$$Y(t) = \left[r_x(t), r_y(t), v_x(t), v_y(t)\right] \tag{2}$$

where $r_x(t)$ is the position in the x-axis, $r_y(t)$ is the position in the y-axis, $v_x(t)$ is the velocity in the x-axis and $v_y(t)$ is the velocity in the y-axis

To track the rate of change of each variable at a specific time, we could define a function f as described by the equation below:

$$f(t, Y(t)) = \frac{\Delta Y(t)}{\Delta t} \tag{3}$$

The function f takes in a time t and a vector $Y(t)$ and returns the differential of each of its components with respect to time at that particular time t. Thus $f(t, Y(t))$ can also be expressed as:

$$f(t, Y(t)) = \left[r'_x(t), r'_y(t), v'_x(t), v'_y(t)\right] \tag{4}$$

$$= \left[v_x(t), v_y(t), a_x(t), a_y(t)\right] \tag{5}$$

where $a_x(t)$ and $a_y(t)$ represent the acceleration in the x-axis and y-axis respectively.

Returning to our gravitational formula above, we can find the acceleration at each position by dividing the force by the mass of the spacecraft. Hence, we can find the acceleration in the x-axis with the following formula:

$$a_x(t) = \frac{G \times M \times r_x}{r^3} \tag{6}$$

where r_x is the distance in the x-axis between the spacecraft and the planet, and r is the distance between the spacecraft and the planet (r and r_x will not be the same). We can do the same for the y-axis.

Task 1: Setting up (8 marks)

Before you get started, take a quick look on `planets.py` first. This file contains all the implementation to represent a planet and functions to plot the spacecraft later.

If you look at the last line, you can see there's a tuple `planets` which consists of the Earth, the Moon, and Mars. Take note of this tuple for the rest of this side quest!

(a) Define the function `get_velocity_component` which calculates the x and y velocity components of the spacecraft and returns them in a tuple. This function takes in the spacecraft's velocity and its angle in degrees. You should make use of the functions in the `math` library. (2 marks)

(b) Define the function `calculate_total_acceleration` that computes the x and y acceleration of the spacecraft. This function will take in a tuple of planets, as well as the current x and y position of the spacecraft. The attributes of each planet can be retrieved by the following accesors:

```
get_x_coordinate(planet): returns the x_coordinate of planet
get_y_coordinate(planet): returns the y_coordinate of planet
get_mass(planet): returns the mass of planet
```

The function will then return the x and y values of the acceleration in a tuple. You should make use of formula 6. The constant G has already been defined for you in `planets.py`. (4 marks)

(c) Lastly, define the function $f(t, Y)$ described in equations 3 and 4. This will be used to compute the spacecraft's position and velocity at each time point. (2 marks)

Task 2: The journey (3 marks)

Once you have defined the functions above, you can make use of the `matplotlib` and `numpy` libraries to solve the whole problem! They provide the functions to plot out the path of the spacecraft in the same way that we have done for Russell. These functions have been set up for you in the template.

Your task is to find out the best angle and speed to launch an imaging spacecraft. The spacecraft should pass by Mars within a given range so as to take a close-up photo of the planet's surface and return back to Earth after it is done. However, there are a few constraints on this mission:

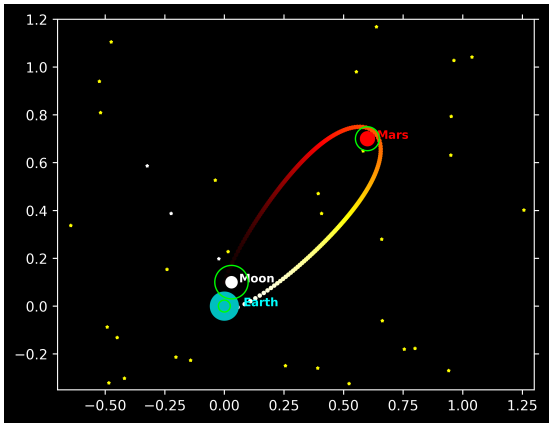
- You are to complete the journey within at most 365 days.
- You shall not crash on the any of the planets (Earth, Moon, and Mars) because we won't be able to recover from the crash's financial loss.
- Once you manage to take a picture of Mars' surface, you have to return to Earth safely without going through the vicinity of the Moon because an enormous amount of space junk is expected to appear near the Moon during the mission's halfway so we decided to play safe.
  We definitely do not want to damage the spacecraft just because of space junk.
- Do not go out of the plot's bounds as we will not be able to track the spacecraft's trajectory.

If you completed the mission properly, you will see three green circles once the animation stops. See the picture of a successful simulation on the next page.

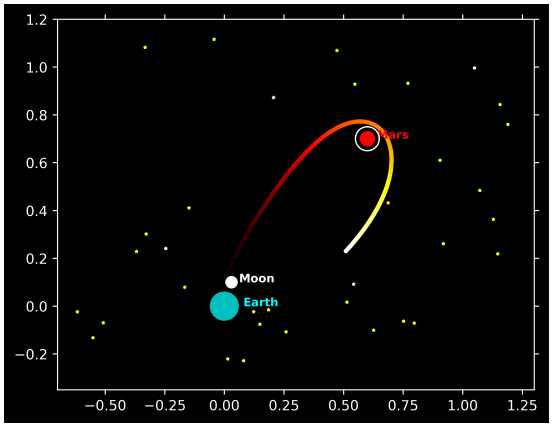
Now, you can change the path of spacecraft by using different values for the function `get_velocity_component` in the template.

```
vx, vy = get_velocity_component(..., ...)
```

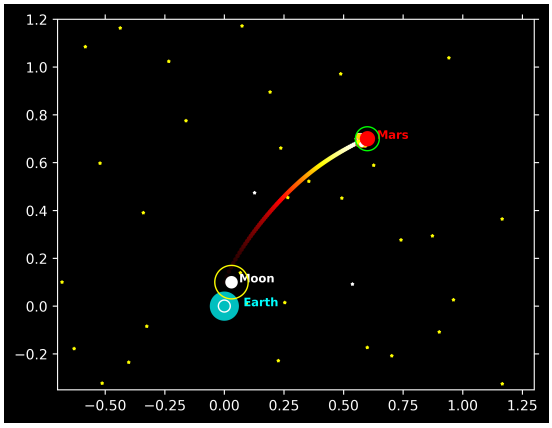
To test your values, uncomment the line, `start_spacecraft_animation`, at the end of the code. Sample trajectories are shown in Figure 1 below.



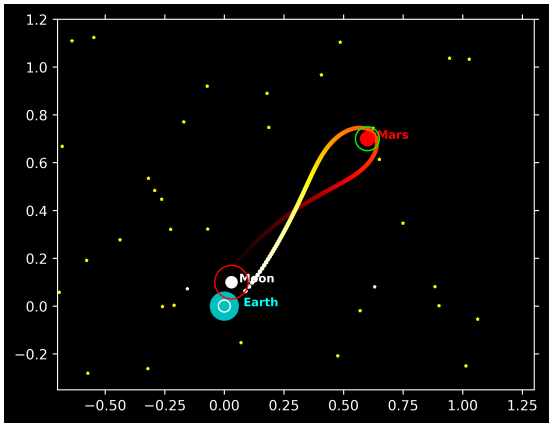
Successful simulation



Unsuccessful simulation



Crashed on Mars



Hit the Moon's vicinity

Figure 1: Sample simulations (your space might be different)

Extra: Saving the animation

You can actually save the animation by using the `.save` method. In order to save it, go to `planets.py` and look at the `start_spacecraft_animation` function.

Simply change the line `plt.show()` into this!

```
anim.save("animation.gif", fps=30)
```

You may modify the filename and the number of frames per second (fps) as you wish. Have fun!