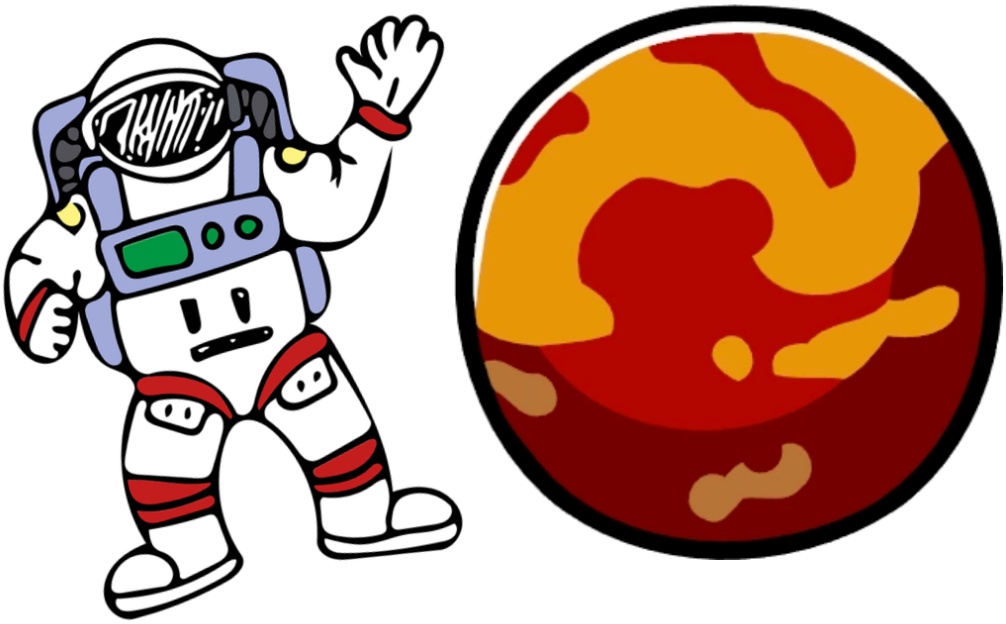
Mission to MARS

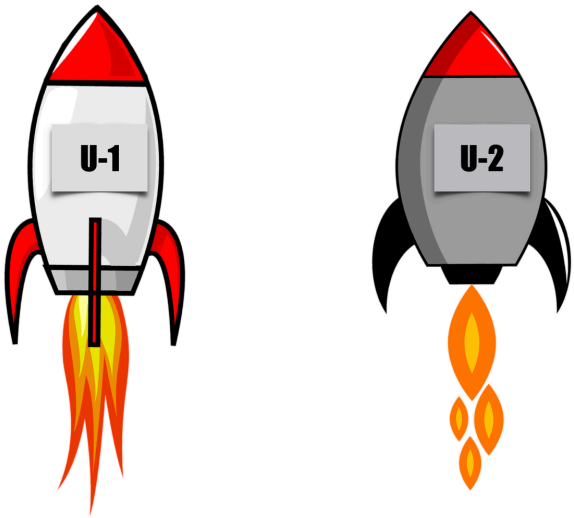


Space Challenge

In this project, you will build a simulation that will help us with our mission to Mars!

The mission is to send a list of items (Habitats, bunkers, food supplies, and rovers) to Mars, but we need to run some simulations first to pick the correct fleet of rockets.

We've already designed 2 rocket prototypes, but we need your help to design and run some simulations to help us decide which type to use.



U-1

The U-1 Rocket is light weight, agile and pretty safe, but can only carry a total of 18 tonnes of cargo. It costs $100 Million to build and weighs 10 tonnes. It has a slim chance of crashing while landing but a bigger chance of exploding when launching, both chances depend on the amount of cargo carried in the rocket.

U-2

The U2 Rocket heavier than the U-1 but much safer and can carry a lot more cargo; to a total of 29 tonnes. However, it costs $120 Million to build and weighs 18 tonnes. It has a greater chance of crashing while landing than while launching, but just like the U-1 both chances depend on the amount of cargo carried.

The Mission

The mission consists of 2 phases:

Phase-1:

This phase is meant to send building equipment and construction material to help build the colony. In the resources tab, you will find a text file that contains the list of all items that we need to send called 'Phase-1.txt`. Each line in the file contains the item name as well as its weight in Kgs.

The file has:

```
building tools=2000
building tools=2000
building tools=2000
building tools=5000
building tools=5000
building tools=2000
building tools=1000
building tools=5000
building tools=6000
shelter equipment=5000
construction equipment=5000
plants=1000
steel=8000
books=1000
water=5000
```

Phase-2:

This phase is meant to send the colony of humans along with some food resources. In the resources tab, you will find a text file that contains the list of all items that we need to send called 'Phase-2.txt`. Each line in the file also contains the item name and its weight in Kgs.

The file has:

```
habitat=3000
colony=5000
```

```
food=5000
habitat=3000
colony=5000
food=3000
colony=5000
food=3000
```

Your job is to run some simulations and test both rocket types for each phase separately.

In a second part, we will use the software as a decision-making tool.

Ready? let's have a look at the details ...

Rocket Specifications

U-1

```
Rocket cost = $100 Million
Rocket weight = 10 Tonnes
Max weight (with cargo) = 18 Tonnes
Chance of launch explosion = 5% * (cargo carried / cargo limit)
Chance of landing crash = 1% * (cargo carried / cargo limit)
```

U-2

```
Rocket cost = $120 Million
Rocket weight = 18 Tonnes
Max weight (with cargo) = 29 Tonnes
Chance of launch explosion = 4% * (cargo carried / cargo limit)
Chance of landing crash = 8% * (cargo carried / cargo limit)
```

Here's what you need to do:

PART 1: The design

1. Create an `Item` class that includes a `String` name and an `int` weight that will represent an item to be carried by the rockets
2. Create a `SpaceShip` Interface that includes the **definitions** of these methods:

- `launch`: a method that returns either `true` or `false` indicating if the launch was successful or if the rocket has crashed.
- `land`: a method that also returns either `true` or `false` based on the success of the landing.
- `canCarry`: a method that takes an `Item` as an argument and returns `true` if the rocket can carry such item or `false` if it will exceed the weight limit.
- `carry`: a method that also takes an `Item` object and updates the current weight of the rocket.

3. Create a class `Rocket` that implements the `SpaceShip` Interface and hence implements all the methods above.

- `launch` and `land` methods in the `Rocket` class should always return `true`. When U1 and U2 classes extend the Rocket class they will override these methods to return true or false based on the actual probability of each type.

- `carry` and `canCarry` should be implemented here and will not need to be overridden in the U1 and U2 classes

4. Create classes `U1` and `U2` that extend the `Rocket` class and override the `land` and `launch` methods to calculate the corresponding chance of exploding and return either `true` or `false` based on a random number using the probability equation for each.

Test the software

We want to be sure that our software works correctly. To do so, we will implement unit tests using JUnit framework.

1. Test carrying features (if an item has been loaded in the rocket, is the carried cargo well incremented? Can we exceed total cargo weight? etc…)
2. How can we test the probabilistic distribution of explosion chances?

PART 2: The simulation

Create a `Simulation` class that is responsible for reading item data and filling up the rockets. The Simulation class should include these methods:

- `loadItems`: this method loads all items from a text file and returns an ArrayList of Items:
- Each line in the text file consists of the item name followed by `=` then its weigh in kg. For example:

```
habitat=100000
colony=50000
food=50000
```

- `loadItems` should read the text file line by line and create an Item object for each and then add it to an ArrayList of Items. The method should then return that ArrayList.
- `loadU1`: this method takes the ArrayList of Items returned from `loadItems` and starts creating U1 rockets. It first tries to fill up 1 rocket with as many items as possible before creating a new rocket object and filling that one until all items are loaded. The method then returns the ArrayList of those U1 rockets that are fully loaded.
- `loadU2`: this method also takes the ArrayList of Items and starts creating U2 rockets and filling them with those items the same way as with U1 until all items are loaded. The method then returns the ArrayList of those U2 rockets that are fully loaded.
- `runSimulation`: this method takes an ArrayList of Rockets and calls `launch` and `land` methods for each of the rockets in the ArrayList. Every time a rocket explodes or crashes (i.e if `launch` or `land` return false) it will have to send that rocket again. All while keeping track of the total budget required to send each rocket safely to Mars. `runSimulation` then returns the total budget required to send all rockets (including the crashed ones).

Run the simulation

Create a `Main` class with the main method and start running the simulation:

1. Create a `Simulation` object

2. Load Items for **Phase-1** and **Phase-2**
3. Load a fleet of U1 rockets for Phase-1 and then for Phase-2
4. Run the simulation using the fleet of U1 rockets and display the total budget required.
5. Repeat the same for U2 rockets and display the total budget for that.
6. Edit Phase1 and Phase2 data files, and test other items list

PART 3: Building a decision-making tool

By running some tests, you should see that regarding the number of items to be carried to Mars, few rockets are subject to explosion. But as it cost a lot of money, we should know the probabilistic distribution for the cost of the project (e.g.: the mission has 87% of chance to be conducted without any loss, 8% of chance for one rocket to explode, causing an extra cost for the mission, etc…). A way to get those data is to run a big number of simulations, and compute statistics.

The idea is to explore some strategies to minimize the cost of the mission, and maximize the mission success chances.

Here are some ideas for the different parameters and strategies to implement. Feel free to implement all those parameters/strategies or not, and to propose other ideas (write them in the description of your personalized project):

- The number of simulations to run to compute the statistic for a given configuration
- Test different Phase1 and Phase2 data files, with more items, and with more diversity regarding weight of item.
- In a first version, chances of explosion were subject to a linear distribution correlated to the carried cargo. Try and implement other distributions (where chances of explosions with no cargo is very low, and chances of explosions with maximal cargo is very high).
- Use of only U1 rockets or only U2 rockets, or the “best” rocket regarding items to carry?
- In the context of a non-linear chance of explosion distribution for the rockets, shouldn’t we explore some other strategies where we try to reduce the cargo to avoid being near the maximal cargo?
- By running some simulations, you should have noticed that the algorithm to load items in the rockets is not optimal. Which other algorithms could we use?
- Any item may have a cost. How can we modify input files to introduce a cost? That cost should be integrated to mission cost (don’t forget to take under account destroyed rockets).
- Take human lives under account: how to minimize human loss (rockets transporting colonies should have a very few chances to explode)? What is the impact of the decisions you incur in the cost of the Colonies?

Your tool should have a graphical interface where all those parameters and chosen strategies should be entered or modified. All parameters and input data for a simulation should be encoded in a scenario (a `Scenario` class?). You may introduce persistency for a scenario (load/save features for a scenario).

Your tool should also graphically display the result of a scenario (the statistics regarding the simulations).

There is no optimal solution. You are facing a classical engineering problem where requirements are continuously changing as the users discover their needs while using the tool. “The problem is not hard, but not well defined/completed”. In that context, it is really important to have a good software architecture which is able to evolve while requirements and requested features are continuously changing. Agile methods are also a good methodology in this context.