Melody Mars Mission App

Requirement summary

- Manage Robots and other vehicles for space exploration
- Move around surface of Mars
- Surface represneted by Plateau
- Plateau can be rectangular for this but should alllow for other shapes
- Rovers can have cameras and robot arms to collect samples
- Use TDD, Unit tests, production quality, expandable!
- Employ structured file system for project
- Keep Mars Rover logic separate to UI code
- Include a README to document key features and user guide
- Plateau is divided into a grid, x,y coords, NSEW
- Rotate Rover L and R by 90 degrees
- Move Rover 1 grid unit at a time

UI Inputs – keep separate from Mars Rover

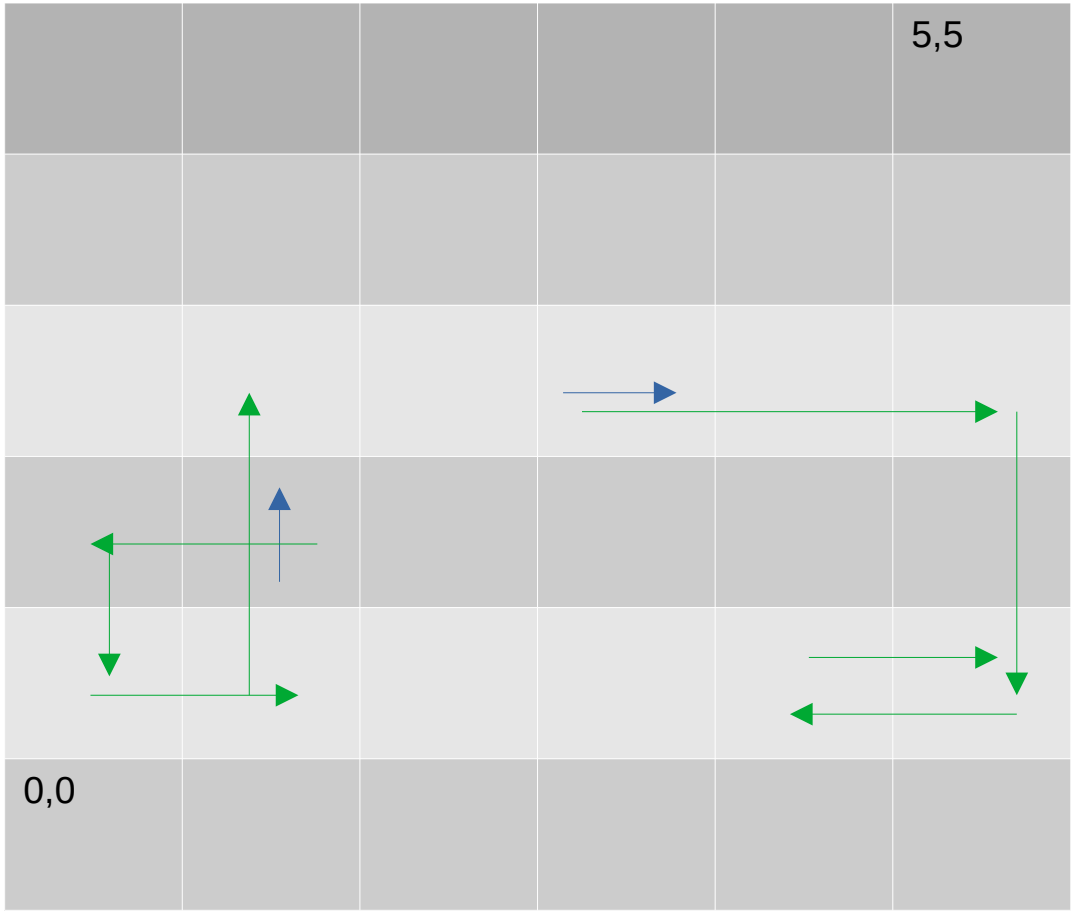
- First input is plateau dimensions (or no of grid points) e.g 5 5 indicates
- Plateau max coord is (5,5) (assume min (0,0)
- Create a plateau object with the given dimensions
-
- Then 2 lines of input per Rover (can repeat):
  - Line 1 gives Start position and Orientation e.g 1 2 N
  - Need to check this is valid and available. Create a new rover
  - Line 2 gives instruction to move Rover around e.g LMMLMRMMML
  - Move the Rover and return the resulting position and direction
  -
- At the end of moving the Rover, the app must be able to return the
- Location and Orientation of the Rover in its finished postion
-

Assumptions and Questions

- For this project I'm going to assume that the plateau is fenced, so that the Rover cannot move outside of the boundary. So if the Rover attempts to move outside I think that movement is blocked and we move onto the next instruction. Alternatively we could raise some sort of exception
- We need to be able to accommodate alternate plateau shapes, even if these aren't implemented
- The Rover is only one type of a vehicle, so it should be possible to create other types of vehicle
- Vehicles should be capable of hosting "instruments" like camera and robot arm
- Possible to use alternate units to XY, e.g Lat, long?
- Does Mars have a NSEW or would using degrees of rotation work better?

Lets try out the example given

1. Input 5 5 – so here is the 6x6 grid from (0,0) to (5,5)



2. Input 1 2 N – so place the Rover in the grid facing N

3. Input LMLMLMLMM – so lets draw that in

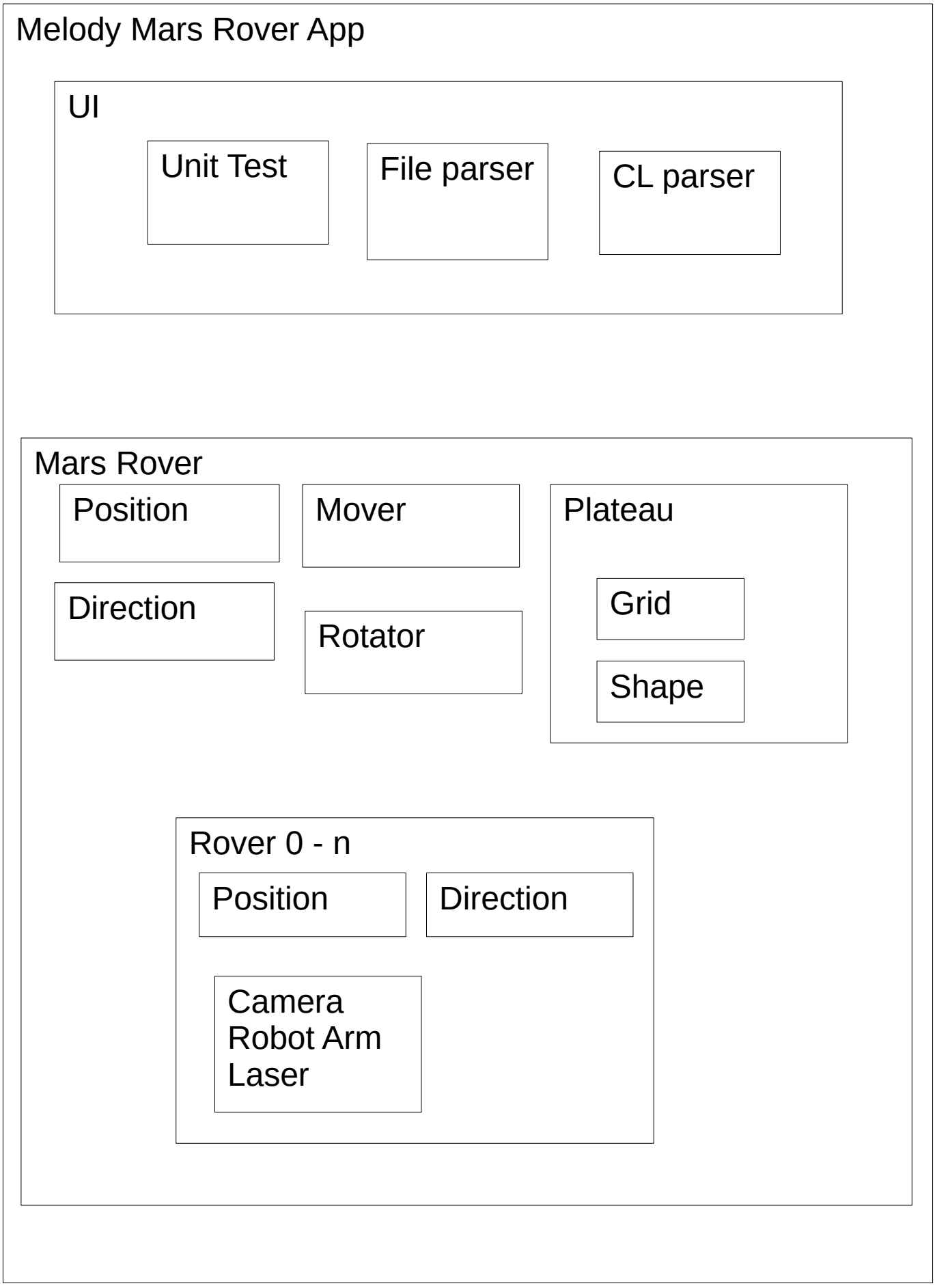
Result is 1 3 N

4. Input 3 3 E – so place 2nd Rover in the grid facing E

5. Input MMRMMRMRRM – draw in

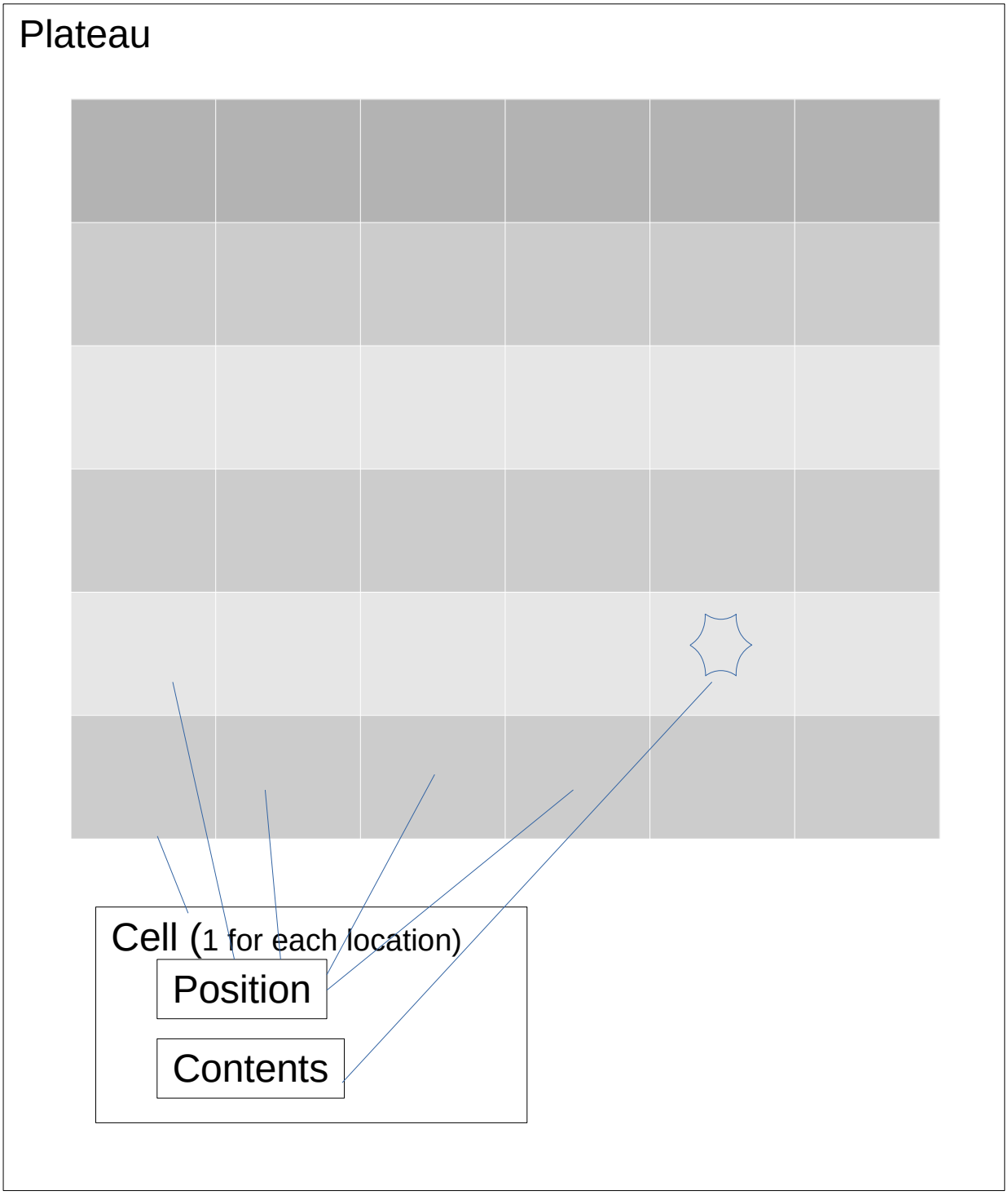
Result is 5 1 E

Melody Mars Rover Initial Overview of Componants



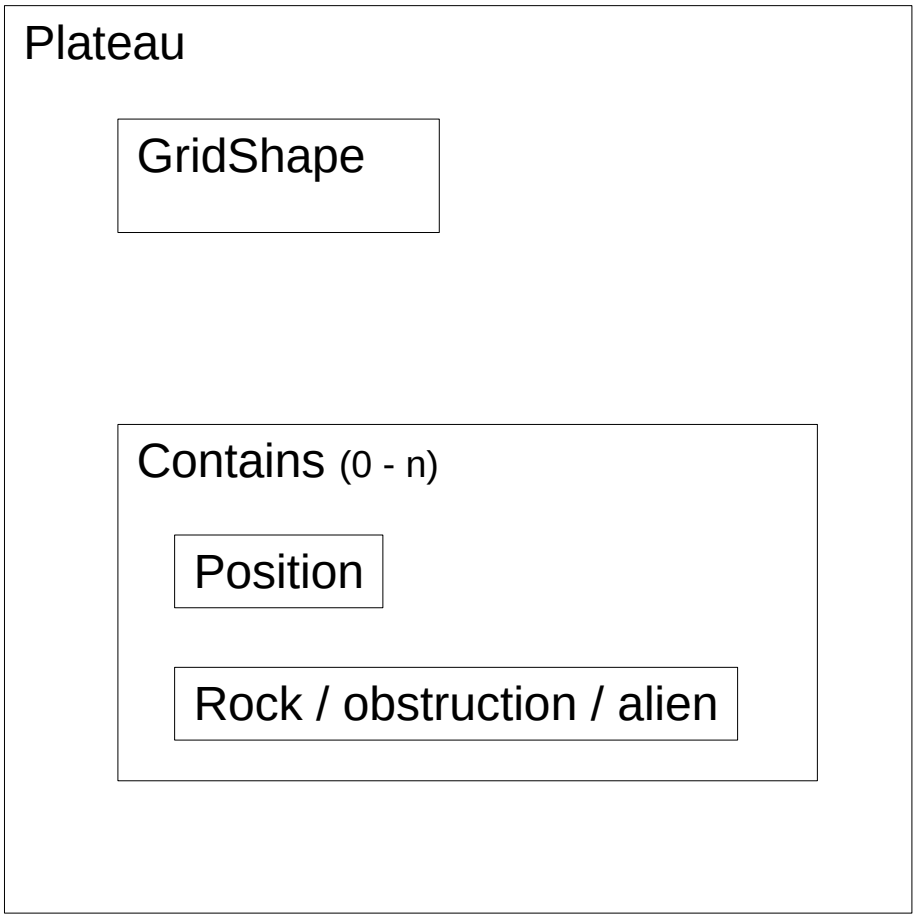
1st Plateau Plan

- Plateau contains a grid within a shape boundary
- Possibly pass a GridShape object to Plateau constructor
- Initial thought was to create a Cell for each location containing its position and any Content located there e.g Obstruction or Rock
- These Cell objects would be created when the Plateau was instanced



This approach of having a cell object for each location would probably work OK on a small grid, eg 6x6 but its not very scaleable eg to a 1000x1000 grid = 1,000,000 cell objects!

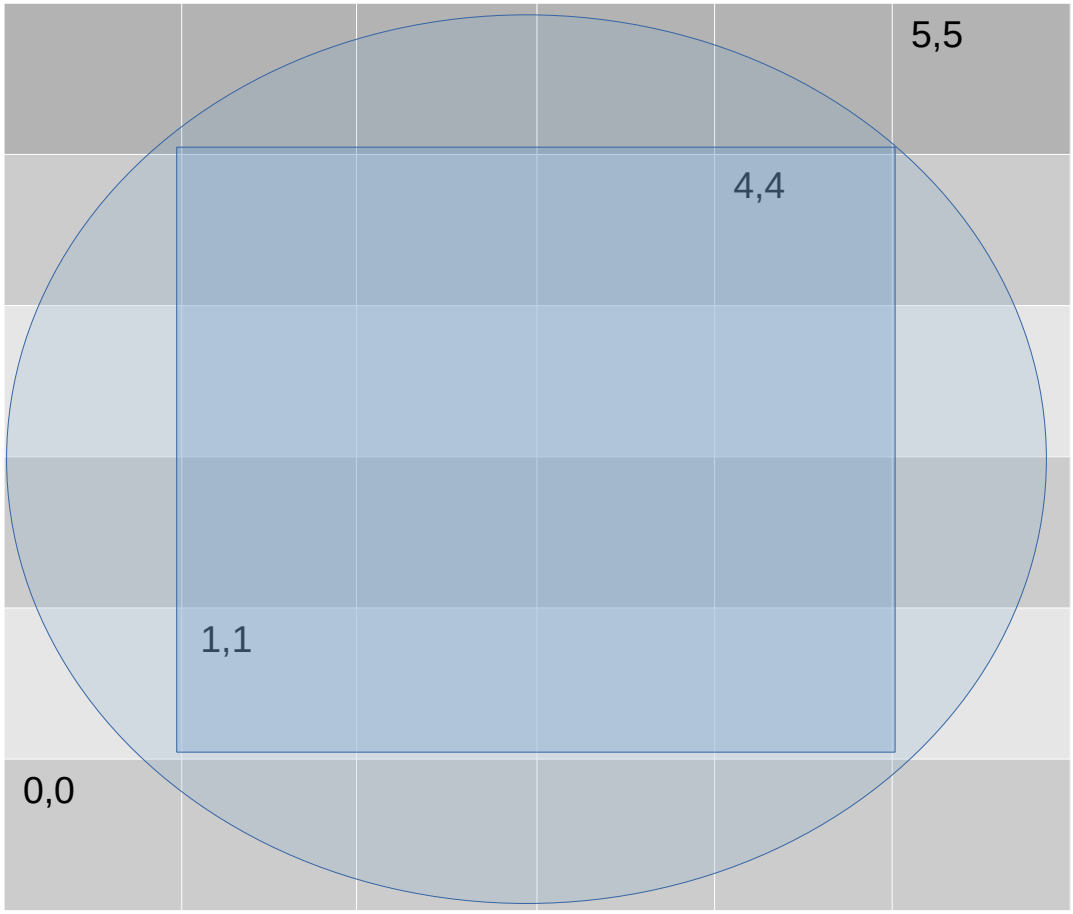
Revised Plateau plan



- The Plateau is instanced with a GridShape
- The Plateau Contains a list of <someclass> which
- represents an object e.g a rock at a particular location

- The GridShape is responsible for determining whether, given a Position, it is inside or outside of the shape boundary

Dealing with other shapes for the Plateau

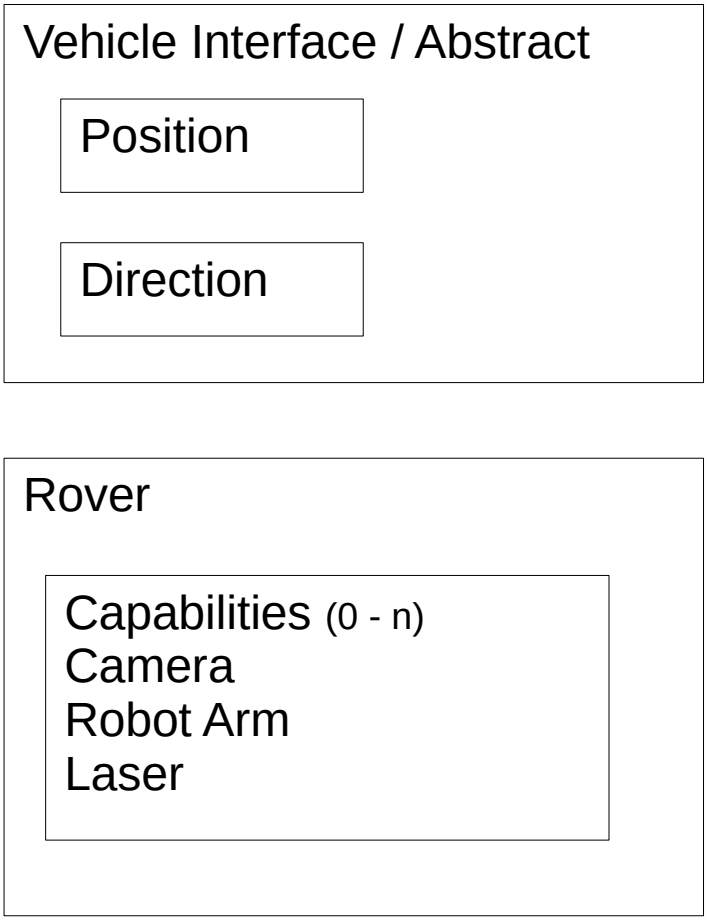


The GridShape Interface inside of the Plateau has a method: IsPositionWithinBoundary (Position) which returns Boolean

This means that we can create different subclasses from GridShape, such as RectangleGrid, CircleGrid etc where each is responsible for its own implementation of IsPositionWithinBoundary() method

For this example we are only concerned with creating a rectangle grid, thus avoiding the complcation of implementing the IsPositionWithinBoundary for more complex shapes, e.g for a CircleGrid we could only use Positions (1,1) through (4,4)

The Rover



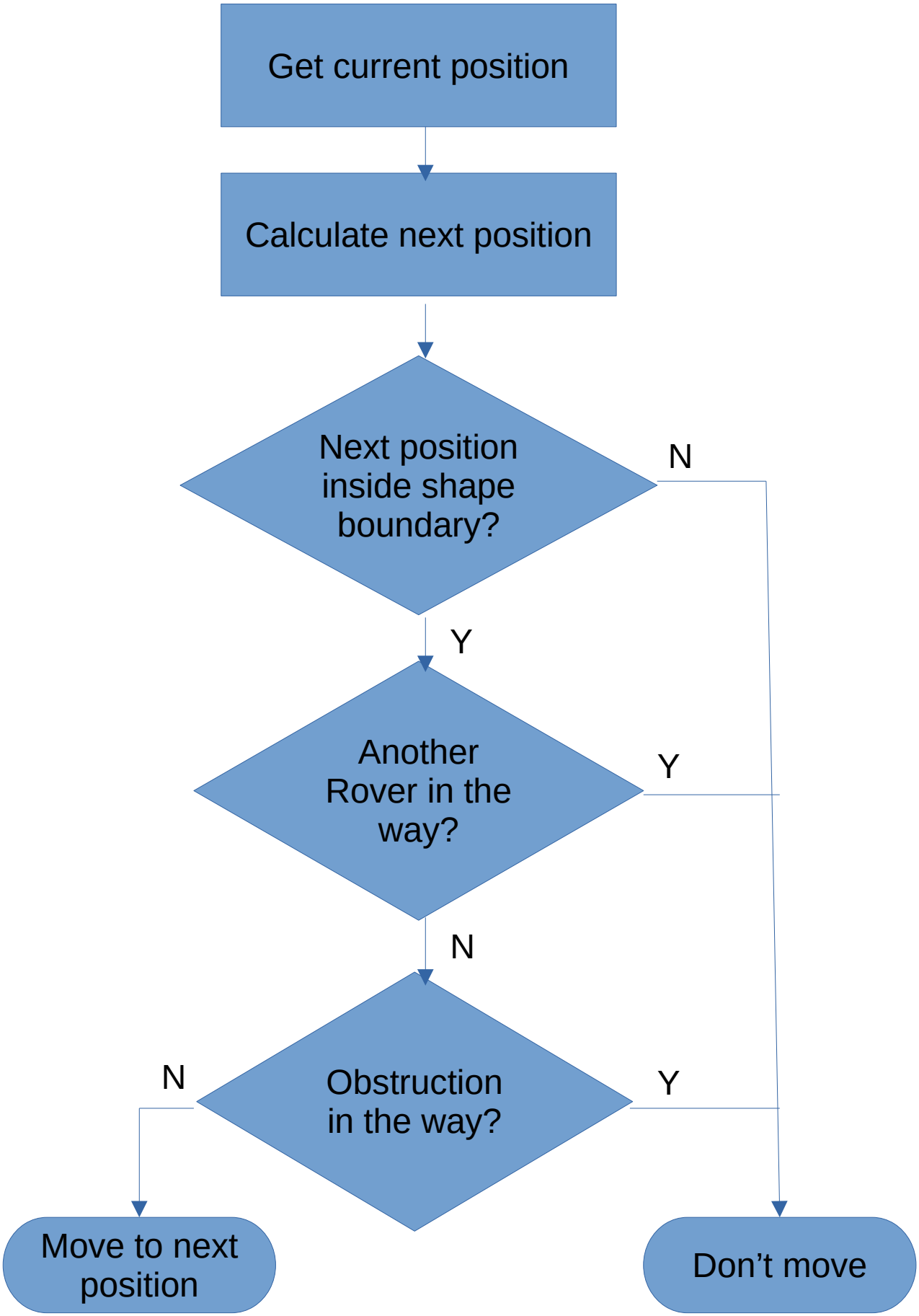
The Rover (derived from a Vehicle class or Interface) contains a Position, a Direction and (optionally) a List of Capabilities which we may use in the future to collect a rock or take a picture (or zap an alien!) for instance

Is the vehicle class responsible for mandating a Move and Rotate method, or do we use seperate Mover and Rotator Classes to do this ?

There is no restriction on performing a change to the Direction

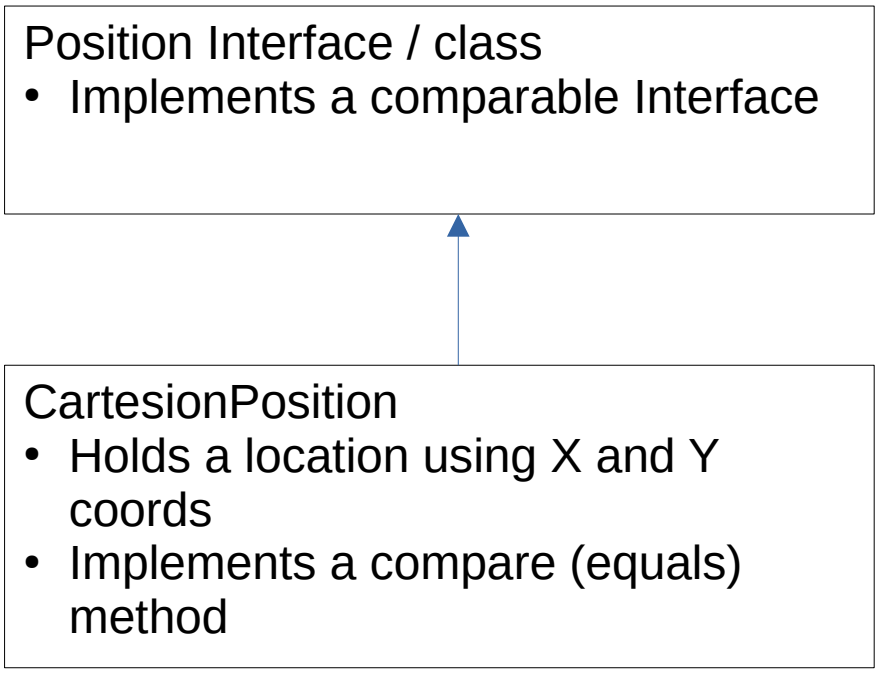
There is a process to follow in order to Move the Rover: (There is also be a similar process to this when positioning a Rover / Obstacle at the start)

Logic to move a Rover (or other Vehicle derived object)



Position Class functionality

- Define a location using whichever coordinate system we've selected
- Maybe implement as an interface so that if an alternative coordinate system is required we just derive another type of positioning system
- So therefore we should derive a CartesionPosition class from this interface for our example
- Has the ability to perform a compare to another Position class using one of the comparitor interfaces

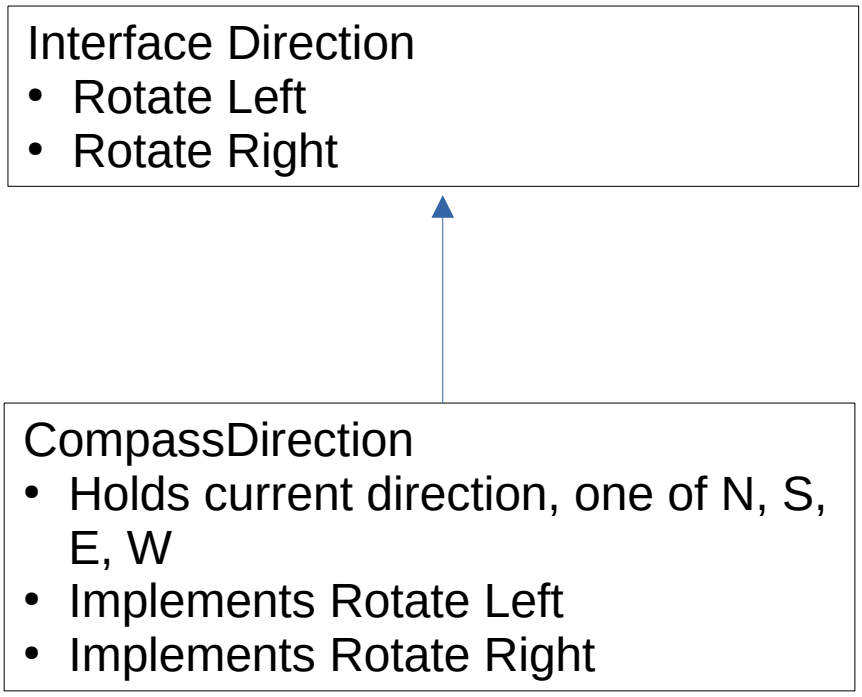


Should a CartesionPosition class be able to handle negative numbers e.g if a circle shape is defined for the Plateau then the centre of the circle may be (0,0) and so left or below may contain negative grid squares?

Should a Position class be able to Move itself?

Direction class functionality

To futureproof this, do we create an Interface and derive a Compass class from this?
What if we wanted to rotate by 45 degrees in the future?
Or by specifying an angle?
Which way is UP on MARS?
Does MARS have a NORTH? - According to Google it does! (earthsky.org)



Position Mover

- This needs a bit of thinking about
- Is the Position object responsible for this?
- How about the Vehicle, does this implement a Mover Interface?
- Do we pass a Position into a Mover class, or a Rover class?
- Do we have a separate PositionCalculator class to calculate the next position, given a current position and a Direction? If so then how? - Does Direction know how to calculate a new position – that kind of makes sense – give a Position to Direction and it calculates a new Position
- Once we've calculated a new position, where is the Moving logic on page 7 carried out?
-

Interface Direction (revised)

- Rotate Left
- Rotate Right
- CalculateNextPosition (Position) : Position

So do we create a Mover class and pass in a Vehicle?
To perform the move logic we need access to:

- the Plateau class, (are we inside the boundary, are there any obstructions in the Postion)
- The list of Rovers (is there another rover in this Position)
- Again for flexibility, do we create a Mover Interface and then implement a PostionClear function from there?
- Sounds like a plan, lets try it

Mover Interface

- PositionClear (Position) : Boolean (or ENUM reason)

Seems like the Mover interface should be implemented by the Rover / Vehicle class

If the Vehicle / Rover class implements a move() method and a PositionClear() method then it requires some kind of link to the Plateau to pass in the Position for testing

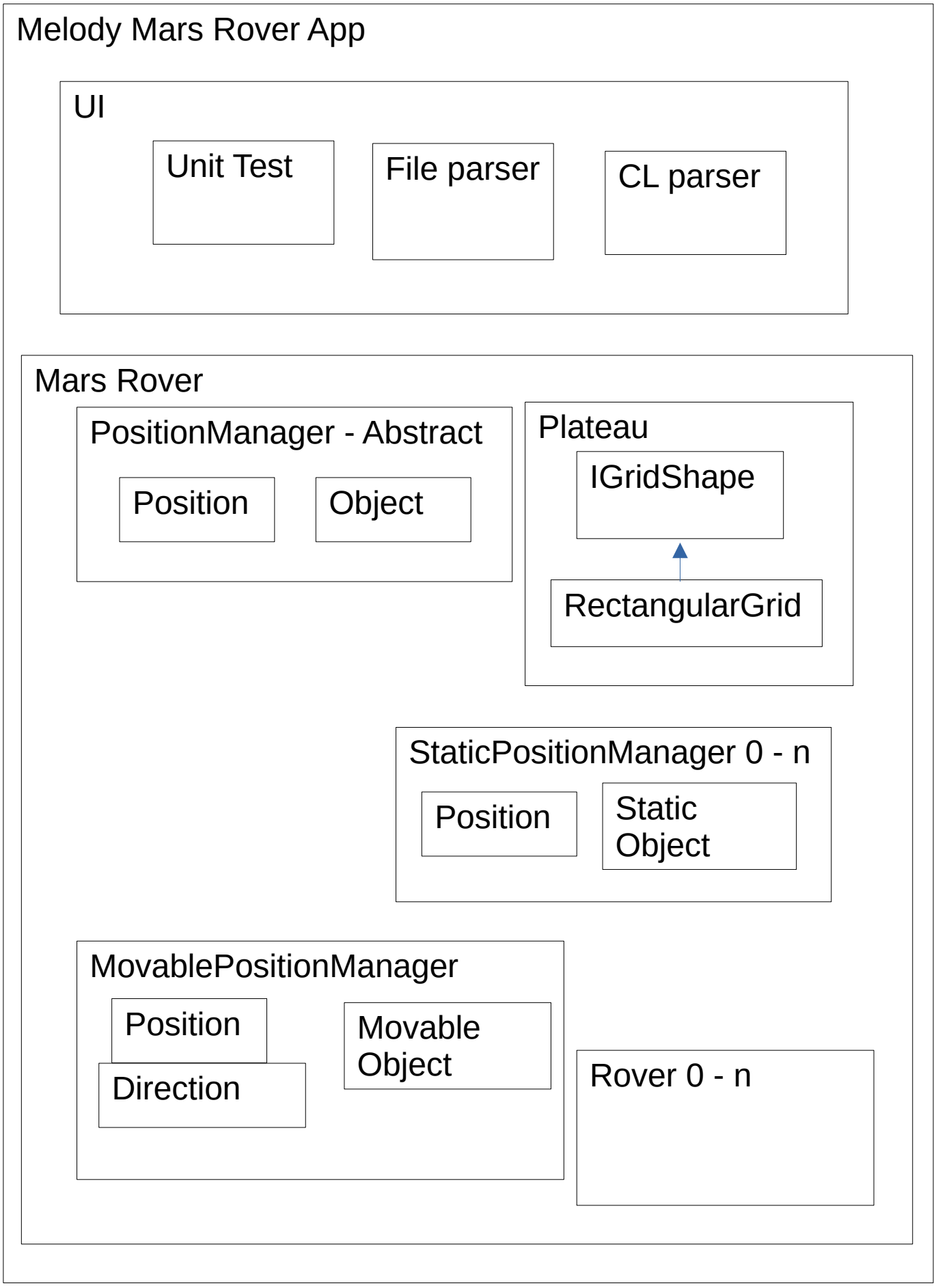
Vehicle Interface / Abstract implements Mover

Position

Direction

- IsPositionClear() : Boolean / ENUM reason for can’t move / List of contents
- Move()
- Rotate Left()
- Rotate Right()

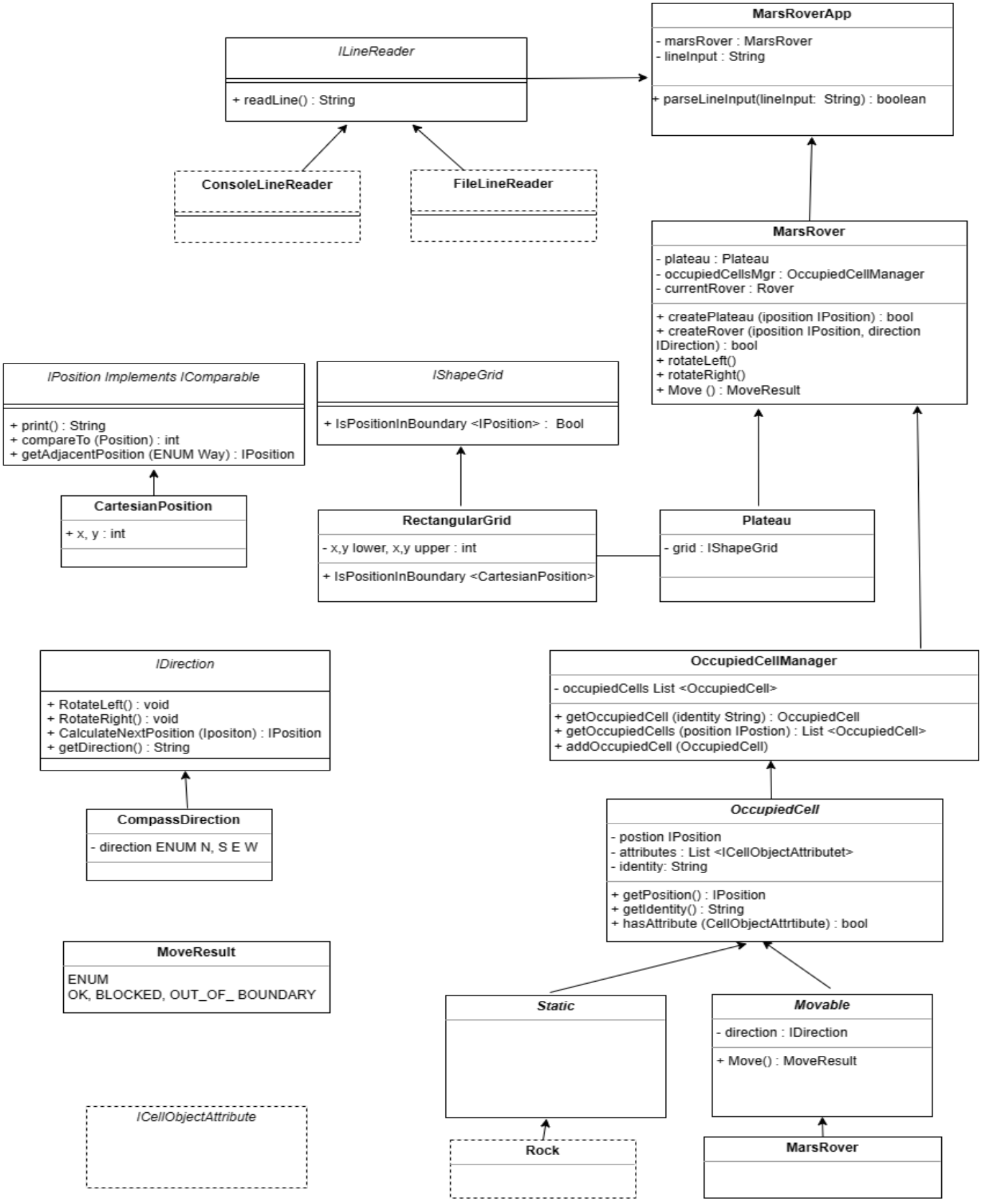
Melody Mars Rover Basic - Revised Overview of Componants



Object Attributes

- Movable
- HasCamera
- HasRoboticArm
- BlocksMove
- Collectable

Lets try and mock up a UML diagram



Project Folders Structure Plan