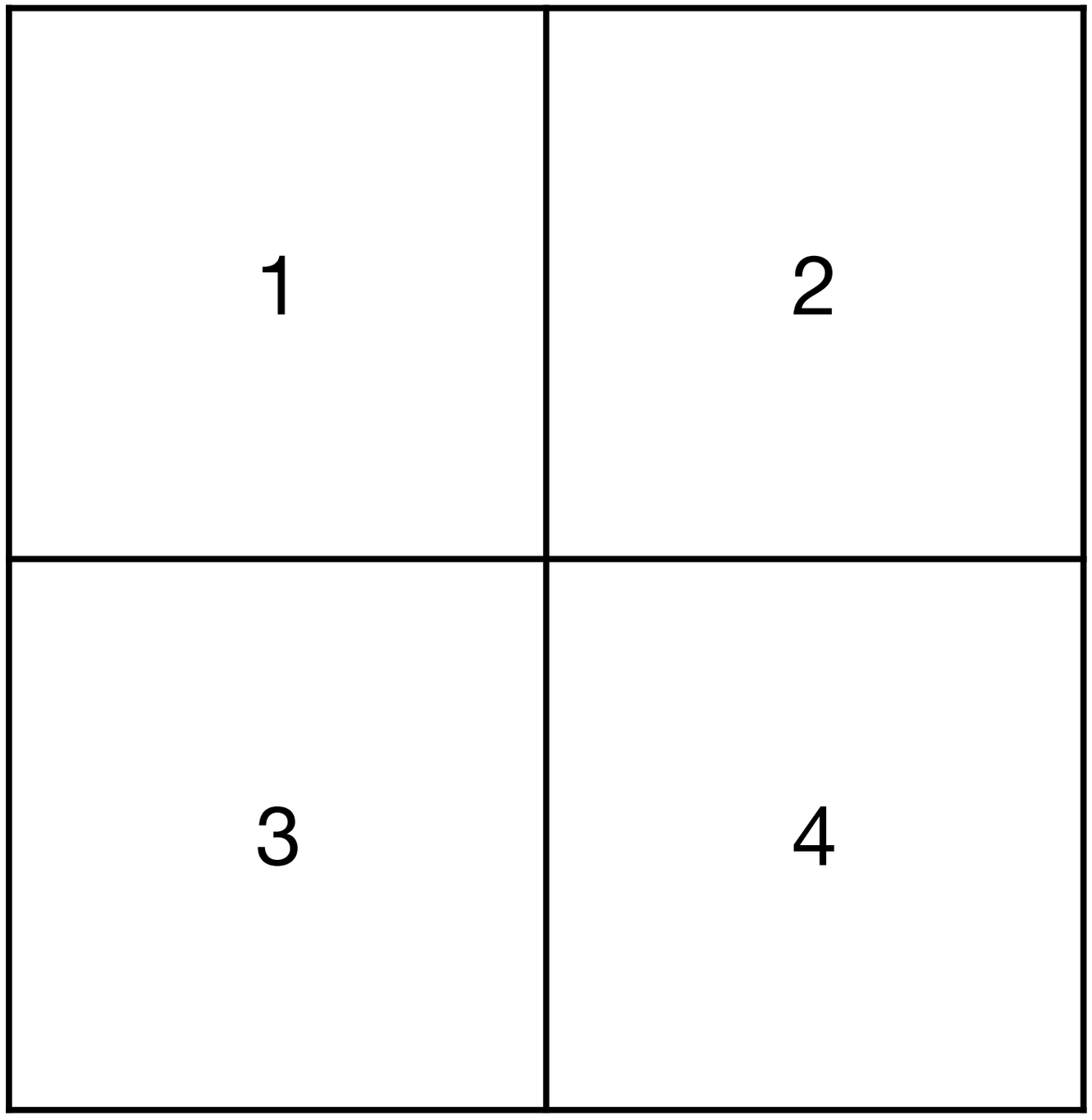
Exercise 1:

Imagine a very simple environment formed by 4 locations (identified by 1, 2, 3, and 4) as in the figure below:



A vacuum-cleaner robot should be programmed in AgentSpeak to maintain the environment clean.

The available actions for the robot are:

- **suck**: remove dirt at the robot's position;
- **left**: move the left;
- **right**: move to right;
- **up**: move up;
- **down**: move down.

To help the robot decide what action to take, the following percepts are given:

- **dirty**: the robot is in a dirty location;
- **clean**: the robot is in a clean location;
- **pos(X)**: the location of the robot is X (1 <= X <= 2).

Download environment:

https://github.com/AngeloFerrando/website/blob/master/assets/courses/jason-exercise/Exercises.zip

Some tips:

You can start programming your agent by thinking about how it should react to the available perception. For instance, what it should do when it perceives "dirty"? **The action "suck", of course!**

In AgentSpeak, we program this reaction by means of a plan as follows:

```
+dirty <- suck. // when dirty is perceived, do the action suck
```

So, an initial and very reactive agent can simply react to every perception and be programmed as shown below (replace "someaction" for the action you think is the most suitable, you might also want to remove some of the plans):

```
+dirty  <- someaction.
+clean  <- someaction.
+pos(1) <- someaction.
+pos(2) <- someaction.
+pos(4) <- someaction.
+pos(5) <- someaction.
```

First solution

```
-------------------------------------
+dirty <- suck.

+pos(1) <- right.
+pos(2) <- down.
+pos(3) <- up.
+pos(4) <- left.
-------------------------------------
```

*** comments**

- this is a reactive agent, which is easy to implement with Jason, but note that the language is meant for cognitive agents
- as a consequence, all plans have an empty context (normally used to check the situation currently believed by the agent)

*** problems**

- the robot may leave dirt behind, and you will get messages like
  [VCWorld] suck in a clean location!
- reason: the events related to location ("+pos(...)") are selected before the "dirty" event, so the suck action is performed after the move action in a possibly clean location

Second solution

```
-------------------------------------
// plans for dirty location
+pos(1) : dirty <- suck; right.
+pos(2) : dirty <- suck; down.
+pos(3) : dirty <- suck; up.
+pos(4) : dirty <- suck; left.

// plans for clean location
+pos(1) : clean <- right.
+pos(2) : clean <- down.
+pos(3) : clean <- up.
+pos(4) : clean <- left.
-------------------------------------
```

*** comments**

- again a rather reactive agent
- the selection of plans is based on context (perceptual beliefs in this case)
- it solves the problems of the previous solution

*** problems**

- the moving strategy is coded in two sets of plans, so to change the strategy we need change all plans
- if you leave the agent running for a long time, it eventually stop. the reason is that sometimes the robot does not perceive neither dirty nor clean and thus no plan are selected. The following code solves that:

Second solution (revisited)

```
---------------------------------------
// plans for dirty location
+pos(1) : dirty <- suck; right.
+pos(2) : dirty <- suck; down.
+pos(3) : dirty <- suck; up.
+pos(4) : dirty <- suck; left.

// plans for other circumstances
+pos(1) : true <- right.
+pos(2) : true <- down.
+pos(3) : true <- up.
+pos(4) : true <- left.
---------------------------------------
```

Third solution

```
---------------------------------------
+pos(_) : dirty <- suck; !move.
+pos(_) : true <- !move.

// plans to move
+!move : pos(1) <- right.
+!move : pos(2) <- down.
+!move : pos(3) <- up.
+!move : pos(4) <- left.
---------------------------------------
```

*** comments**

- the moving strategy is re-factored to use a (perform) goal (the goal '!move')
- this agent reacts to the perception of its location, but the reaction creates a new goal (to move)
- to change the moving strategy we only need to change the way the goal "move" is achieved (the plans for triggering event '+!move')

*** problem**

- suppose that actions may fail, in this case, after performing 'up', for example, the agent may remain in the same place and then no new location is perceived: the agent will stop moving
  (this is only conceptually a problem since the environment was not coded to simulate action failures, i.e., the environment model is deterministic).

Fourth solution

```
---------------------------------------
!clean. // initial goal

+!clean : clean <- !move; !clean.
+!clean : dirty  <- suck; !move; !clean.
-!clean          <- !clean.

+!move : pos(1) <- right.
+!move : pos(2) <- down.
+!move : pos(3) <- up.
+!move : pos(4) <- left.
---------------------------------------
```

* comments

- this agent is not reactive at all; it has no behaviour which is triggered by an external event, that is, a perception of change in the environment (note however that BDI agents typically have both goal-directed and reactive behaviour)

- instead, the agent has a *maintenance goal* (the '!clean' goal) and is blindly committed towards it (if it ever fails, the goal is just adopted again -- see below)

- this goal is implemented by a form of infinite loop using recursive plans: all plans to achieve 'clean' finish by adding 'clean' itself as a new goal

- if anything fails in an attempt to achieve the goal 'clean', the contingency plan (-!clean) reintroduces the goal again at all circumstances (note the empty plan context); this is what causes the "blind commitment" behaviour mentioned above

- this agent is thus more 'robust' against action failures

Exercise 2:

Now the vacuum-cleaner robot must be able to clean a grid of any possible size.

Implement all the necessary plans to make it independent from the size!

Update:

+!move:

Add:

+!go_home:

Hint:

Keep track of the spaces already visited!!!