Task 1 - Include more inderterminism in the enviroment.

1.a - Randomly scattering some pieces of garbage on the grid:

At the constructor method of the `MarsModel` class, we removed the 5 lines that created the garbage in a default position. Then, we replace them with a loop that places garbage in random positions with: `add(GARB, x, y)`, where `x` and `y` are the random positions. Before this coordinates are selected for the garbage dropping, we check that there is no previous garbage, this is performed with: `hasObject(GARB, x, y)`.

1.b - Randomly placing the incinerator (together with r2) on the grid.

At the constructor method of the `MarsModel` class, we removed the line that places the r2 agent in the default position. The same process as the previous exercise, placing the r2 agent with: `setAgPos(1, r2Loc)`, where `r2Loc` is the new random position.

1.c - Randomly placing r1 on the grid.

The same as the incinerator (r2) random placing, but with the r1 agent: `setAgPos(0, r1Loc)`, where `r1Loc` is the new random position.

Task 2 - Modify r2 agent to fail as r1 when picking up the garbage.

R2 agent could fail up to 3 times when trying to pick up the garbage that r1 agent left in his position.

We define in `MarsModel` a new variable named `burnerr`, that saves the number of tries of burning garb are made by the incinerator.

The incinerator can fail (random), and the maximum number os fails permited is defined by the variable `MErr`.

The `burnGarb()` function:

```
void burnGarb() {
    // r2 location has garbage
    if (model.hasObject(GARB, getAgPos(1))) {
        // TASK 2: r2 fail as r1 when picking up the garbage
        if (random.nextBoolean() || burnerr == MErr) {
            remove(GARB, getAgPos(1));
            burnerr = 0;
        } else {
            burnerr += 1;
            System.out.println("[LOG] Failed burning garbage " + burnerr + " times.");
        }
    }
}
```

In `r2.asl` file we defined a new behaviour: the agent will recursively active the `!ensure_burn(S)` plan until the call to the `burn(garb)` function actually burns the garbage.

```
+garbage(r2) : true <- burn(garb); !ensure_burn(r2).

+!ensure_burn(r2) : garbage(r2) <- burn(garb); !ensure_burn(r2).

+!ensure_burn(_).
```

Task 3 - Modify r1 agent for changing its searching path.

3.a - Scanning top-down instead of left-right.

For this task we modify the `nextSlot()` function in the `MarsModel` class. The function takes the position of the r1 agent and changes for one below, and when the last position of the grid (last row) is reached, we continue from the first position of the next column.

```
void nextSlot() throws Exception {
    Location r1 = getAgPos(0);
    // TASK 3 (a): move top-down
    r1.y++;
    if (r1.y == getHeight()) {
        r1.x++;
        r1.y = 0;
    }
    ...
}
```

3.b - Scanning continuosly.

We use the same `nextSlot()` function used in the previous task. For moving continuosly r1 will behave using "top-down" strategy but without stopping the searching process in the last position of the grid. So, every time it arrives the last row of the last column, it will move to the first column of the first row to restart the searching process.

`nextSlot()`:

```
void nextSlot() throws Exception {
    Location r1 = getAgPos(0);
    // TASK 3 (a): move top-down
    r1.y++;
    if (r1.y == getHeight()) {
        r1.x++;
        r1.y = 0;
    }
    // finished searching the whole grid
    // TASK 3 (b): move continously
    if (r1.x == getWidth()) {
        r1.x = 0;
        r1.y = 0;
    }
    ...
}
```

Task 4 - Include a new crazy Robot r3 that moves and produces garbage randomly.

We define the r3 agent in the `mars.mas2j` file, then we create a `r3.asl` file that will contain its plans and the "beliefs".

`r3.asl`:

```
!wander(S).

+!make_garb(S): not garbage(r3)
    <-
    ?pos(r3, X, Y);
    gen_garb(X, Y);
    !wander(S).

+!make_garb(S)
    <-
    !wander(S).

+!wander(S)
    <-
    move_around(S);
    !make_garb(S);
    !wander(S).
```

This agent will use the `moveAround()` function to move randomly, and will use the `genGarb()` function to generate garbage randomly.

The process can be summarized as follows:

- Set the initial location of the r3 agent randomly in the constructor method of the `MarsModel` class: `setAgPos(2, r3Loc)`.
- Move the agent continuosly using the `moveAround()` function.
- Generate garbage where there is no previous garbage and a probability of 10%, using: `if (!model.hasObject(GARB, loc) && random.nextFloat() < 0.1) add(GARB, loc);`

Task 5 - Include a new task at your choice.

For this additional task we create a r4 agent that is a battery charger, so the r1 agent will have a battery level that decreases with every move. When the r1 agent has no more battery, it will go to the charger (r4) location.

In the constructor of the `MarsModel` class we added a r4 agent at a initial random position.

We added the "battery logic" in the `r1.asl` file:

- The r1 agent will have an initial belief of `battery(30)`.
- The `check(slot)` plan will reduce the battery each time this plan is executed, only if the `charge` belief is not included.
- When the battery is empty, we remove all the intentions remaining and all the `at(), pos()` beliefs. After that, we activate the plan `!charge_agent(charge)`.
    - `!charge_agent(charge)`: move the r1 agent to the charger location, activating the `charge` belief. When the battery is full, move to the original position and reactivate the `check(slots)` plan.