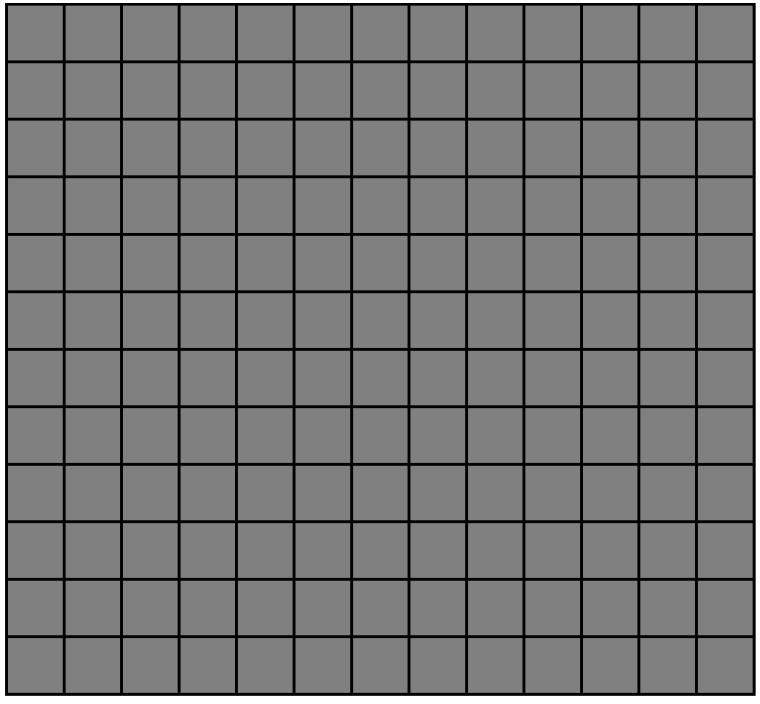
MOBILE ROBOTICS

EXPLORATION ALGORITHM

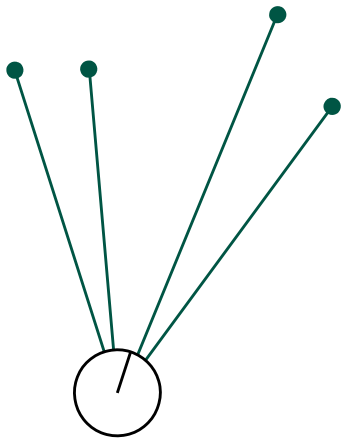
MAPPING

Grid map:
Obstacle = 1
Free cell = 0
Unknown = 0.5



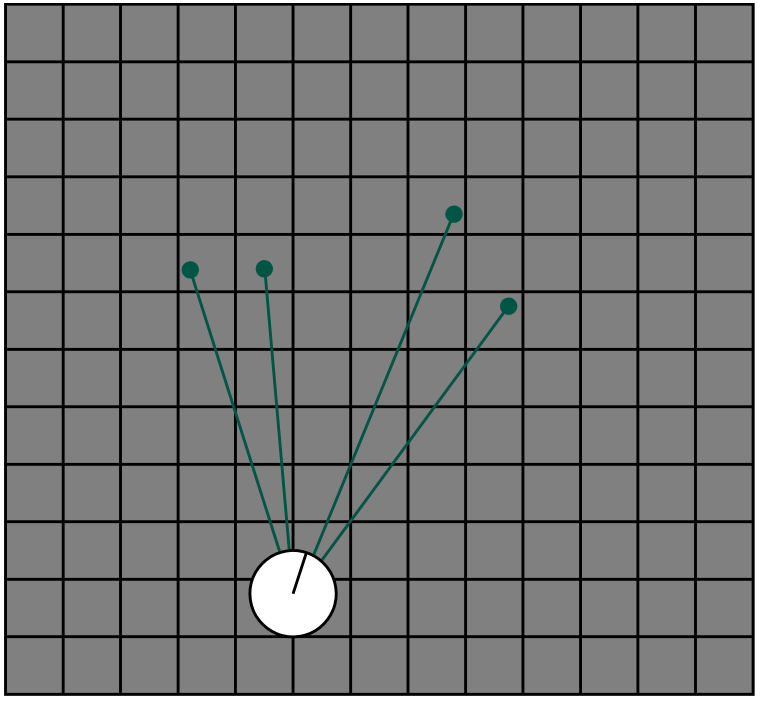
MAPPING

LiDAR measurement



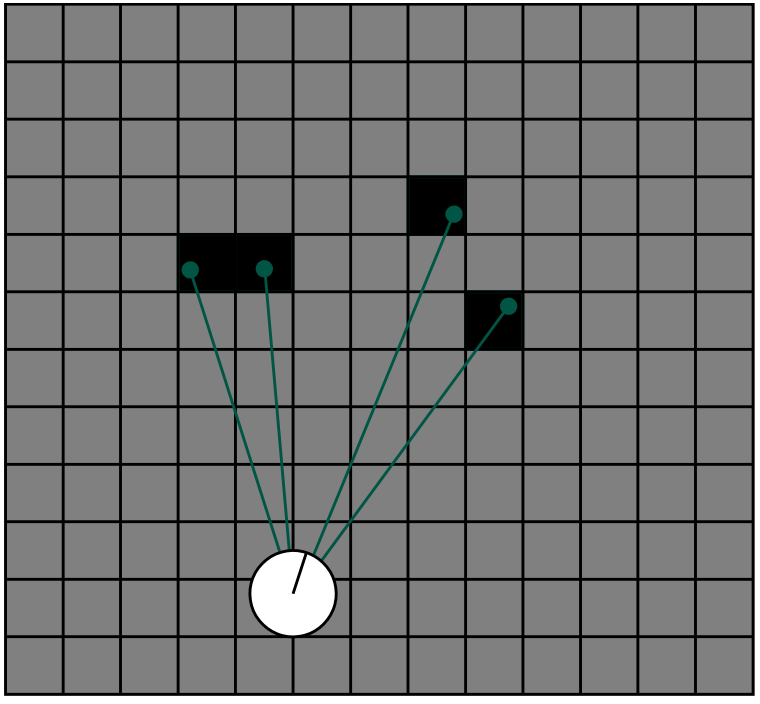
MAPPING

Build the map assuming that the robot's pose is known



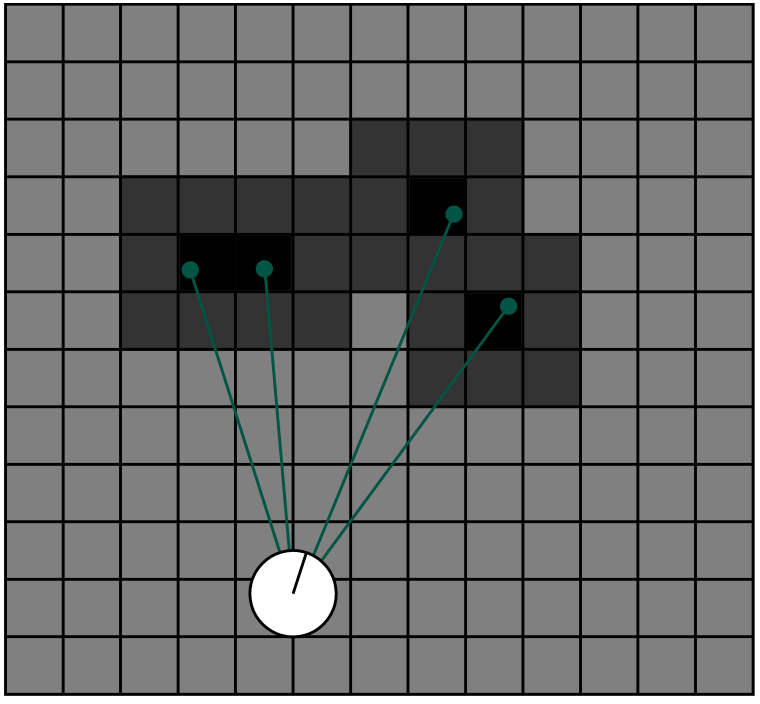
MAPPING

Add the detected obstacles in the map
Cell value : 1



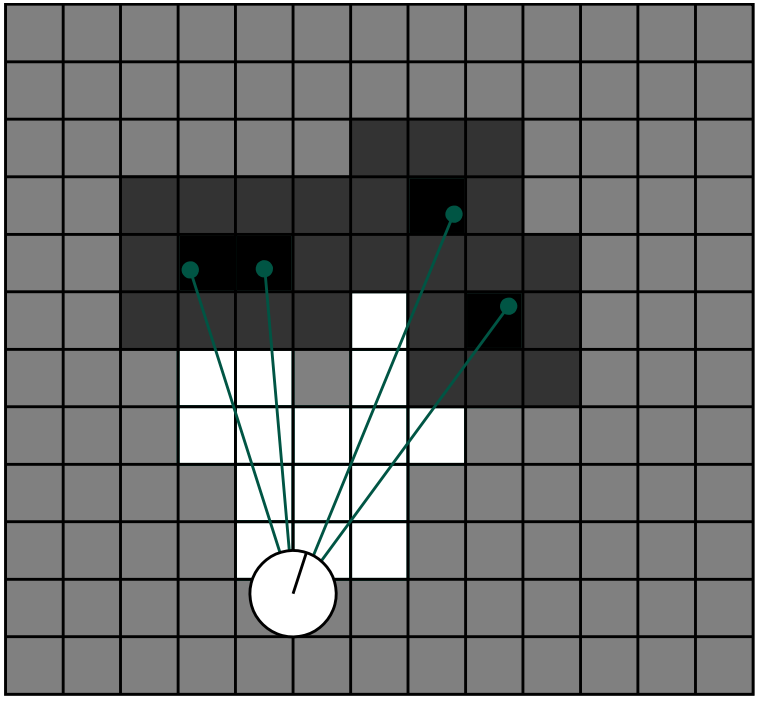
MAPPING

Enlarge the detected obstacles so the robot will not move too close to the obstacles



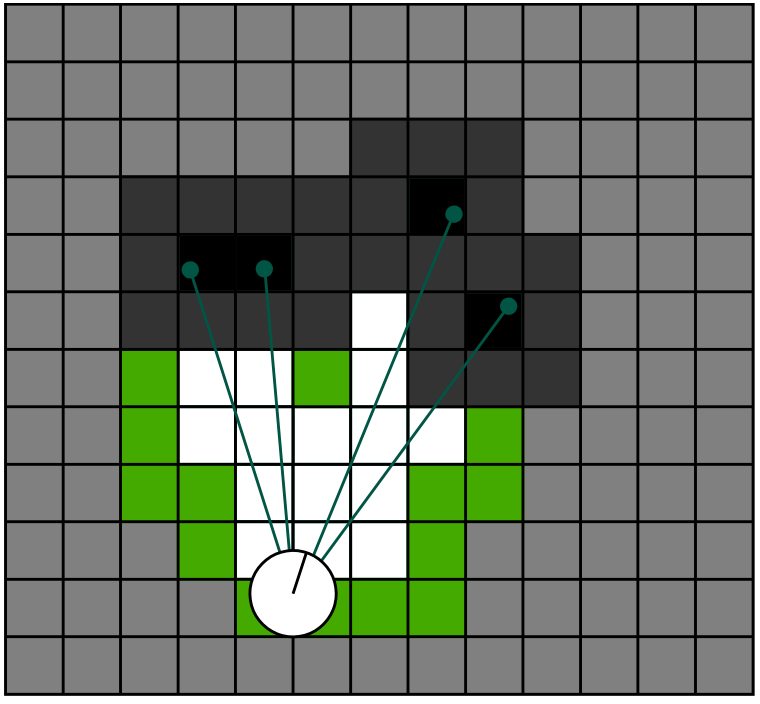
MAPPING

All the cells between the robot and the detected obstacles can be marked as free cells



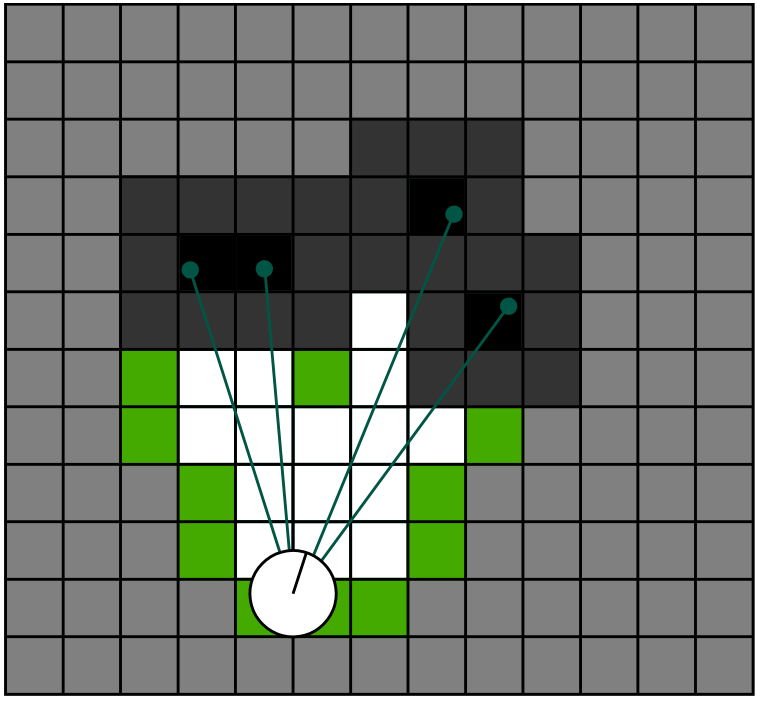
MAPPING

All the unknown cells that are next to a free cell are tagged as frontiers



MAPPING

Note that all the detected frontiers that are not next a free cell can be untagged as frontier



Work to do

- Files to send by email
  - map.py
  - word2do.py
- Warnings
  - Test your code functions after functions, you can use the “Test” button as you want
  - Do not modify the other files
    - You may however modify the btn_test_event() function in simulator.py for you tests
  - Do not add any library (numpy for instance)
- https://youtu.be/B-dSyKx4Fsc

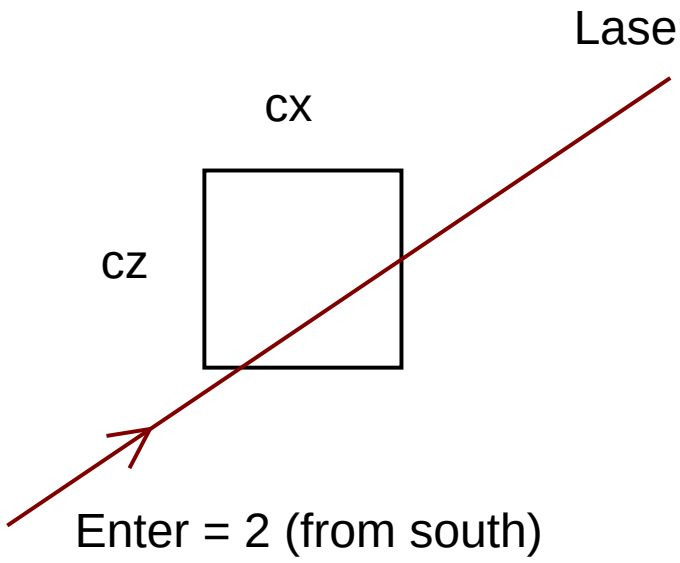
1.1 enlarge_obstacle()

- map.py
- Function to enlarge the obstacles so that the robot will not move too close to the obstacles
- All the cells that have an obstacle as neighbor must have a 0.7 value
  - Cell value 1 : obstacle
  - Cell value 0.7 : enlarged obstacle
  - Cell value 0.5 : unknown
  - Cell value 0 : obstacle free

1.2 update_cell()

- update_cell(cx, cz, laser, enter)

The cell cx, cz can be noted as free and not to be a frontier

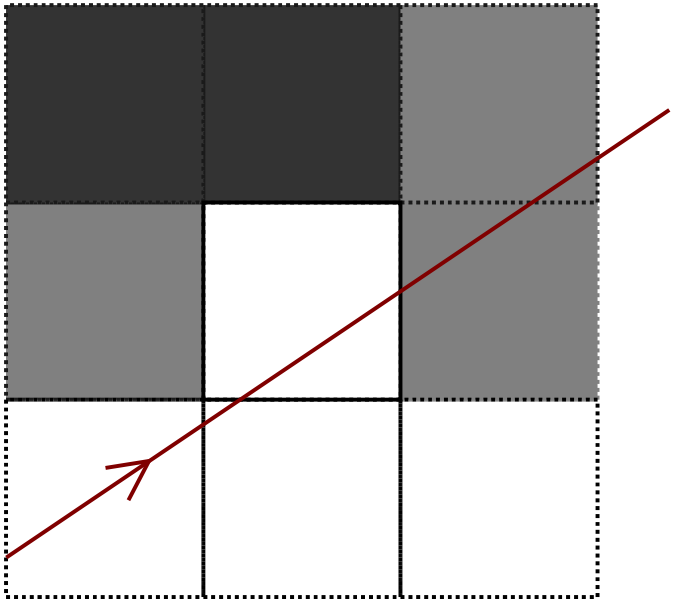


Enter value:
0: first cell to be tested
1: enter from north
2: enter from south
3: enter from east
4: enter from west

1.2 update_cell()

- update_cell(cx, cz, laser, enter)

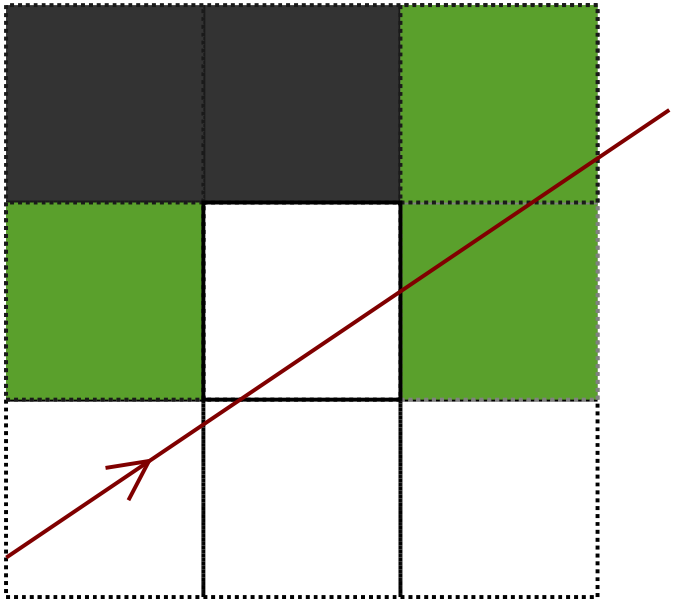
Check the neighbor of the tested cell



1.2 update_cell()

- update_cell(cx, cz, laser, enter)

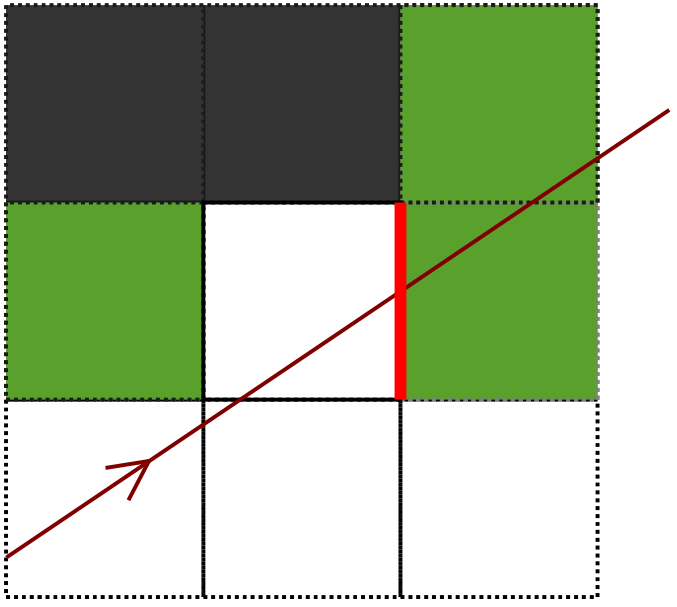
All the neighbor that are unknown can be tagged as frontier



1.2 update_cell()

- update_cell(cx, cz, laser, enter)

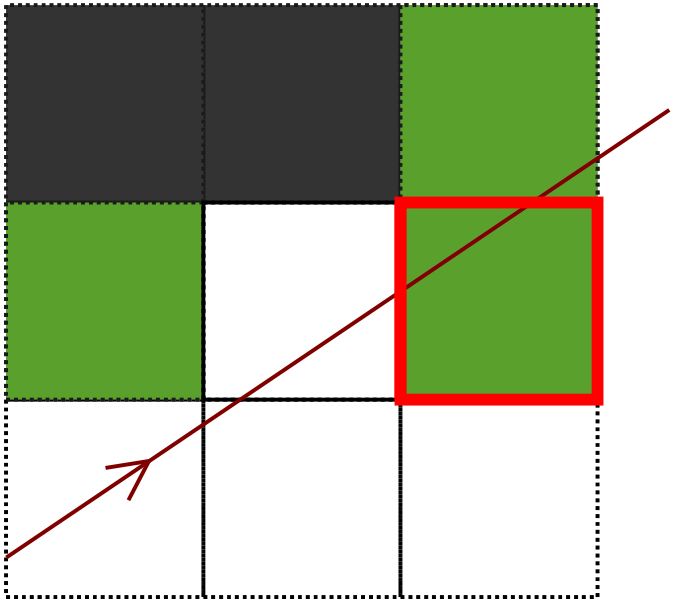
Check the side of the cell that is intersected by the laser ray



1.2 update_cell()

- update_cell(cx, cz, laser, enter)

The next cell to be tested, with enter = 4 (enter from west)



1.2 update_cell()

- Recursive function (until there is a cell to update – until the end of the segment “laser” has been reached)

1.3 free_path()

- Function that marks all the cells between the robot and the detected obstacle as a free cell (without obstacle) - it uses the update_cell() function
  - rx, rz : the robot's position
  - obsx, obsz : the obstacle's position

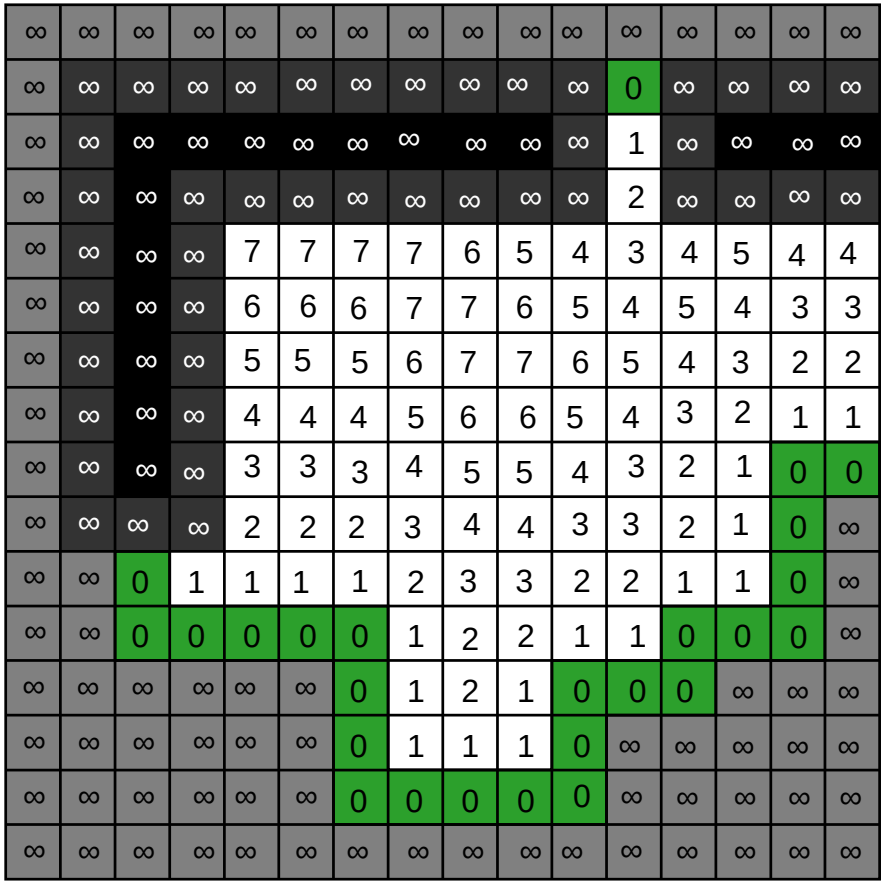
1.4 add_obstacle()

- function to add an obstacle in the map, it uses the free_path() and enlarge_obstacle() functions
  - rx, rz : the robot's position
  - obsx, obsz : the obstacle's position

1.5 remove_isolated_frontiers()

- remove the isolated frontiers, a frontier must have at least one free neighbor (a cell with a value 0)

COSTMAP


∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ 0 ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ 1 ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ 2 ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ 7 7 7 7 6 5 4 3 4 5 4 4
∞ ∞ ∞ ∞ 6 6 6 7 7 6 5 4 5 4 3 3
∞ ∞ ∞ ∞ 5 5 5 6 7 7 6 5 4 3 2 2
∞ ∞ ∞ ∞ 4 4 4 5 6 6 5 4 3 2 1 1
∞ ∞ ∞ ∞ 3 3 3 4 5 5 4 3 2 1 0 0
∞ ∞ ∞ ∞ 2 2 2 3 4 4 3 3 2 1 0 ∞
∞ ∞ 0 1 1 1 1 2 3 3 2 2 1 1 0 ∞
∞ ∞ 0 0 0 0 0 1 2 2 1 1 0 0 0 ∞
∞ ∞ ∞ ∞ ∞ ∞ 0 1 2 1 0 0 0 ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞ ∞ 0 1 1 1 0 ∞ ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞ ∞ 0 0 0 0 0 ∞ ∞ ∞ ∞ ∞
∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞ ∞

2.1 init_costs()

- All the costs are initialized as infinity costs
- The “changed” variable can be initialized to True

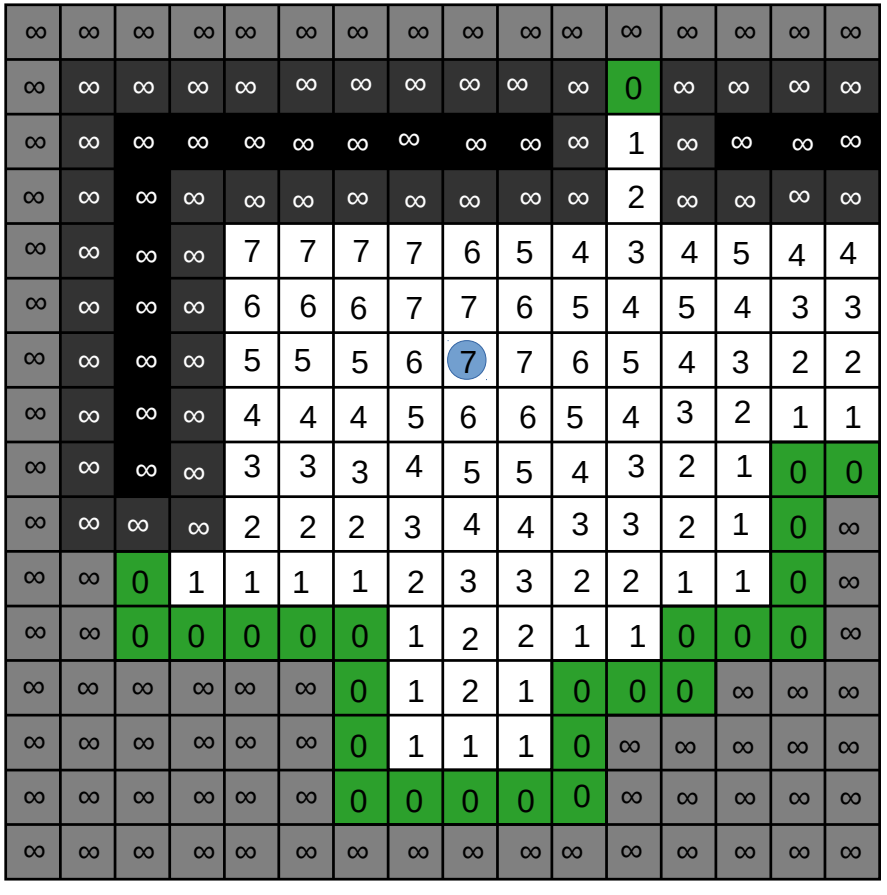
2.2 compute_nsew_nsew()

- Compute the cost using diagonals
  - NW/SE, NE/SW, SW/NE, SE/NW
- Each frontier must have a cost of 0
- Only the costs of the free cells are computed
- If a cell cost is updated the “changed” variable has to be updated to True

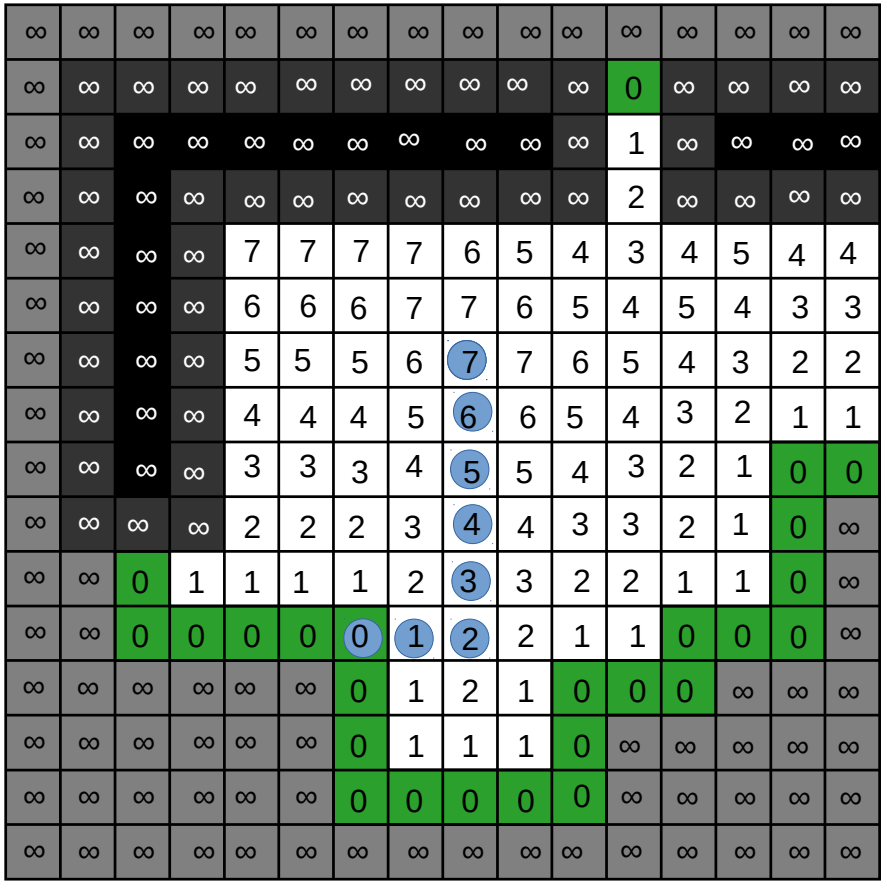
2.3 compute_costs()

- This function uses the compute_ nsew_nsew() functions (until there is no update anymore, use the “changed” variable to detect that)
- The cost of a cell is its distance to the closest frontier

MOVE THE ROBOT



MOVE THE ROBOT



3.1 set_target()

- word2do.py
- Return a target (Point2D) corresponding to the center of the next cell the robot has to go (to get closer to a frontier)

3.2 go_2_target()

- word2do.py
- Return the command U = [ul, ur] to lead to the robot to the target (Point2D)
- Note that the robot can either rotate or go forward