Roomba

This project simulates the functioning of a Roomba vacuuming device in a room, comparing different cleaning strategies. The simulation relies on cellular automata models, and, in this case, the roomba occupies one cell only and traverses cell by cell throughout the whole grid in the room. As it traverses around the room, the roomba tries to clean as many cells as possible without hitting an obstacle throughout the way (represented as blocked cells).

Model

The models assume that the roomba is located in a static room, that is, no movable objects are present in this scenario. Additionally, it assumes that the roomba is always placed at a square room, with dimensions provided as an input to the model. The roomba cleans a randomly selected percentage of the total dirt in the cell, trying to mimic the real life scenario in which the dirt might be something more difficult (something stuck at the floor) or easier (light dust) for the roomba to clean. The threshold for the roomba to stop cleaning is when the room is 80% cleaner compared to when the vacuum started to work. The assumption is that a room 80% clean is an acceptable scenario, and any effort from the roomba to clean more than that does not bring enough utility to justify the resources required to do so.

As mentioned before, the roomba is assumed to be randomly placed into the room each time a new simulation starts, which seeks to capture the uncertainty present in a real life scenario. Additionally, the simulation assumes that the roomba can incorporate different scenarios based on the following inputs:

- Room size;
- Dirt density of the room;
- Density of obstacles in the room.

The dirt density is calculated based on the space available in the room. For instance, if the room has 8 cells and 4 of them are occupied by obstacles; inputing a dirt density of 50% will spread the dirt among the 2 unoccupied cells (out of a total of 4 unoccupied cells). Each cell has a percentage of how dirty it is (0% to 100%), and all the dirty cells and how much dirt each holds have to sum up to the total dirt density inputted into the model. The amount of dirt that each cell holds is randomly determined by the model.

When initializing the simulation, there is a chance that the roomba is placed in the middle of several objects and is unable to move. In this case, the simulation is reseted and the code interprets this situation as having the roomba cleaning 0% of the room.

This project implements four different cleaning strategies, and the efficiency of each one is analyzed by a combination of how clean the room is after the roomba runs through it and how many times in total the cells were visited more than once by the roomba. As for the visualization, the ‘x’ represents obstacles; the ‘.’ represents a dirty cell (no matter how dirty it is); and the ‘R’ represents the roomba position.

Strategy 1: 8 neighbors

In this strategy, the roomba moves up, down, left, right, and diagonally to all directions. However, as it is the case for all the other strategies, the roomba only moves one cell at a time. The roomba randomly selects one of its 8 neighboring cells to be its next destination, not considering obstacles as available cells. After every move, the roomba analyzes its current cell and, if there is any amount of dirt, the roomba will clean anything from 0 to 100% of the cell (the percentage value is randomly selected by the model).
Example:

Steps 1 and 2

```
[x, ., ., x, ., .]
[., ., ., ., R, x]
[x, ., x, ., x, .]
[., x, x, ., ., x]
[x, ., ., ., x, x]
[x, x, ., ., ., .]
```

```
[x, ., ., x, ., .]
[., ., ., ., ., x]
[x, ., x, ., x, R]
[., x, x, ., ., x]
[x, ., ., ., x, x]
[x, x, ., ., ., .]
```

Steps 3 and 4

```
[x, ., ., x, ., .]
[., ., ., ., ., x]
[x, ., x, R, x, .]
[., x, x, ., ., x]
[x, ., ., ., x, x]
[x, x, ., ., ., .]
```

```
[x, ., ., x, ., .]
[., ., ., R, ., x]
[x, ., x, ., x, .]
[., x, x, ., ., x]
[x, ., ., ., x, x]
[x, x, ., ., ., .]
```

Strategy 2: 4 neighbors

Similarly to the first strategy, under this configuration, the roomba will randomly select one of its four available neighboring cells to be its destination; moving only up, down, left, and right in any direction. The actual cleaning of the cell is the same as in the first strategy

Example:

Steps 1 and 2

```
[x, ., x, ., x, .]
[., ., ., x, ., .]
[., ., ., ., R, .]
[x, x, ., x, ., .]
[x, x, ., ., x, .]
[., ., ., ., ., .]
```

Steps 3 and 3

```
[x, ., x, ., x, .]
[., ., ., x, ., .]
[., ., ., ., R, .]
[x, x, ., x, ., .]
[x, x, ., ., x, .]
[., ., ., ., ., .]
```

```
[x, ., x, ., x, .]
[., ., ., x, R, .]
[., ., ., ., ., .]
[x, x, ., x, ., .]
[x, x, ., ., x, .]
[., ., ., ., ., .]
```

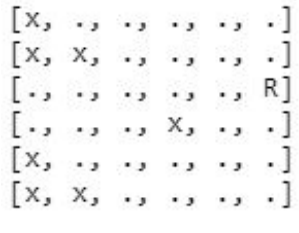
```
[x, ., x, ., x, .]
[., ., ., x, ., .]
[., ., ., ., ., R]
[x, x, ., x, ., .]
[x, x, ., ., x, .]
[., ., ., ., ., .]
```

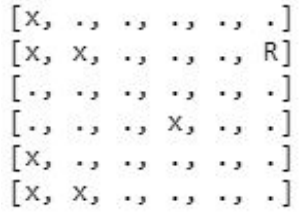
Strategy 3: Prioritize straight lines up and down

In this strategy, the roomba will run up until it hits the north wall, turn right, and then go all the way to the south wall, repeating these steps until it hits the wall on the right. When this happens, the roomba repeats its up and down pathway, but this time always turning left when it hits the north or south walls. Whenever the roomba cannot move north or south (depending on which direction it is determined to go), it will randomly pick one of its 8 neighboring cells to move to and proceed with the cleaning, going either up or down. This process is repeated until the room is a minimum of 80% cleaner compared to when the roomba started running.

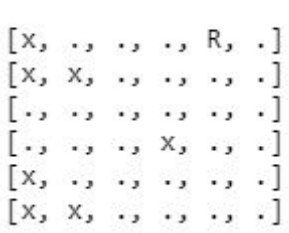
Example:

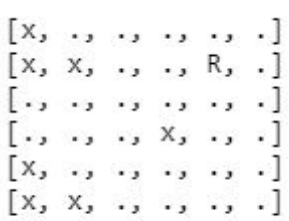
Steps 1, 2 and 3

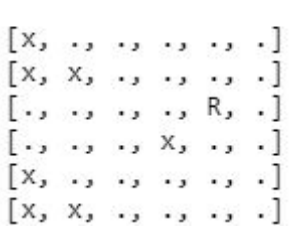




Steps 4, 5, and 6







Strategy 4: Prioritize walls

In this strategy, the roomba is allowed to move to each of its 8 neighboring cells, always prioritizing the cells neighboring the walls. In this case, the simulation assigns a higher probability of having the roomba at one of those cells (70% chance) in comparison to cells not neighboring the walls of the room (30% chance).

Example:

Steps 1, 2, and 3

```
[., ., ., ., ., .]
[., x, x, ., x, .]
[., ., ., x, ., x]
[., ., ., ., ., .]
[x, ., ., ., ., .]
[., ., x, ., R, .]
```

```
[., ., ., ., ., .]
[., x, x, ., x, .]
[., ., ., x, ., x]
[., ., ., ., ., .]
[x, ., ., ., ., R]
[., ., x, ., ., .]
```

```
[., ., ., ., ., .]
[., x, x, ., x, .]
[., ., ., x, ., x]
[., ., ., ., ., .]
[x, ., ., ., R, .]
[., ., x, ., ., .]
```

Steps 4, 5, and 6

```
[., ., ., ., ., .]
[., x, x, ., x, .]
[., ., ., x, ., x]
[., ., ., ., ., .]
[x, ., ., ., ., R]
[., ., x, ., ., .]
```

```
[., ., ., ., ., .]
[., x, x, ., x, .]
[., ., ., x, ., x]
[., ., ., ., R, .]
[x, ., ., ., ., .]
[., ., x, ., ., .]
```

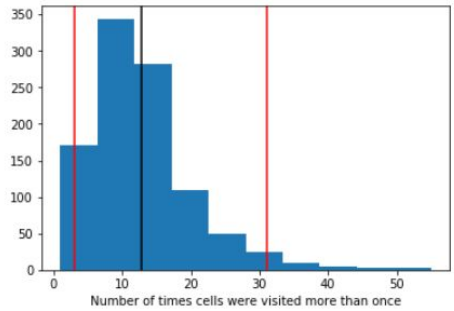
```
[., ., ., ., ., .]
[., x, x, ., x, .]
[., ., ., x, ., x]
[., ., ., ., ., R]
[x, ., ., ., ., .]
[., ., x, ., ., .]
```

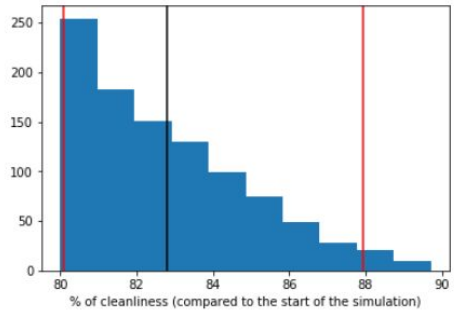
Simulation

The simulation was run, in average, 1500 times for each obstacle density value, going from density of 0 to 0.9, increasing by 0.1 for each 1000 runs. To account for the cases where the roomba can be initialized in the middle of objects and thus get stuck; the simulation runs a minimum of 1000 times for each obstacle density value, and a maximum of 2000 times. For every obstacle density value, the roomba moved 500 times, stopping once it cleaned more than 80% of the room or after the pre-determined 500 moves. The room size and the dirt density of the room were the same for all the runs, with a room size of 6 by 6 and dirt density of 0.5. The following graphs show hows how clean the room was after the roomba went through it, as well as how many times the roomba revisited cells.

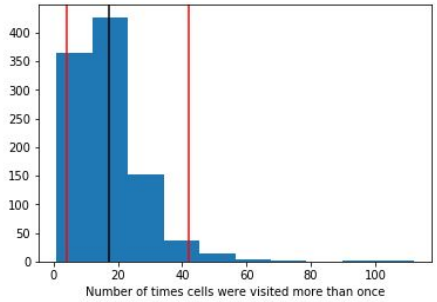
Best-case scenario: No obstacles (no objects in the room)

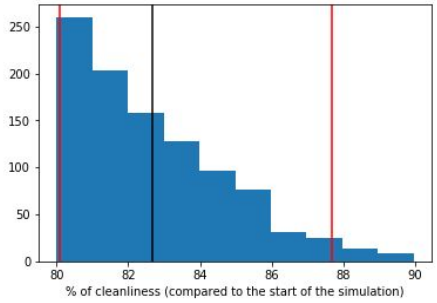
```
##################################################
Strategy 1
##################################################
Density: 0.0
Cancelled Turns: 0
```



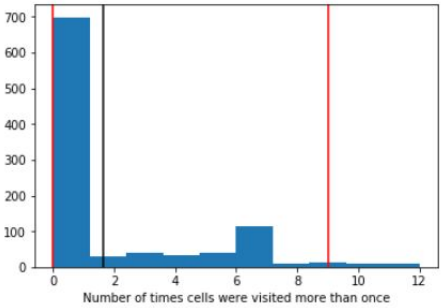


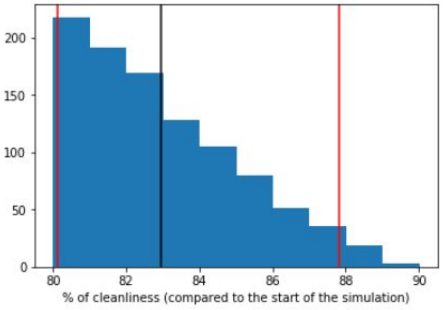

```
##################################################
Strategy 2
##################################################
Density: 0.0
Cancelled Turns: 0
```



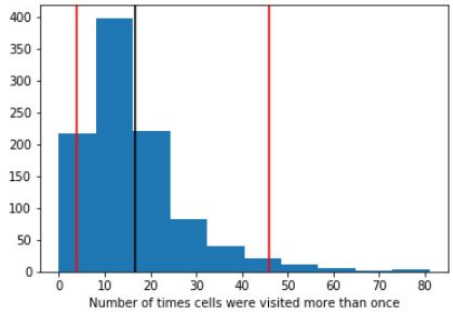


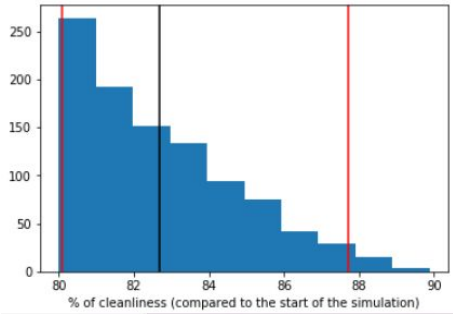

```
##################################################
Strategy 3
##################################################
Density: 0.0
Cancelled Turns: 0
```





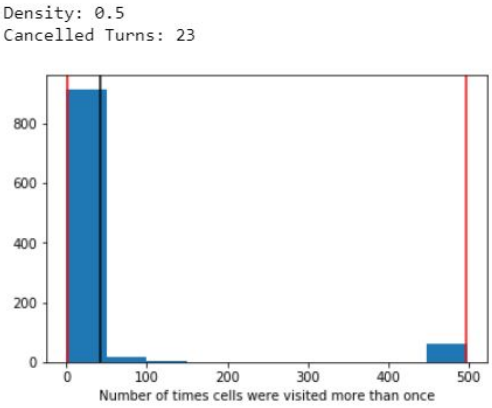

```
##################################################
Strategy 4
##################################################
Density: 0.0
Cancelled Turns: 0
```

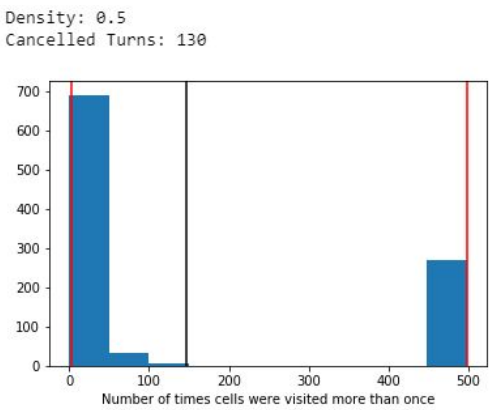


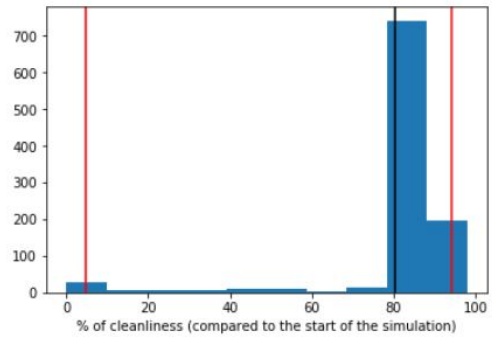


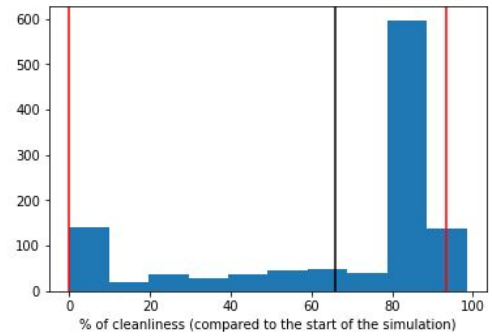
In the best case scenario, the room has no obstacles, so the roomba can move freely and only obey to its own strategy guidelines. In this scenario, all strategies clean almost the same amount of dirt, leaving the room, on average, 83% cleaner. However, the third strategy is the most efficient when it comes to revisiting cells, having a mean value of 1 revisit per simulation run. Although visually it seems that we have a wide confidence interval for the cleanliness of the room, in fact we have somewhat of a tight interval, with most of the simulation runs returning a room 80 to 88% cleaner with all the strategies.

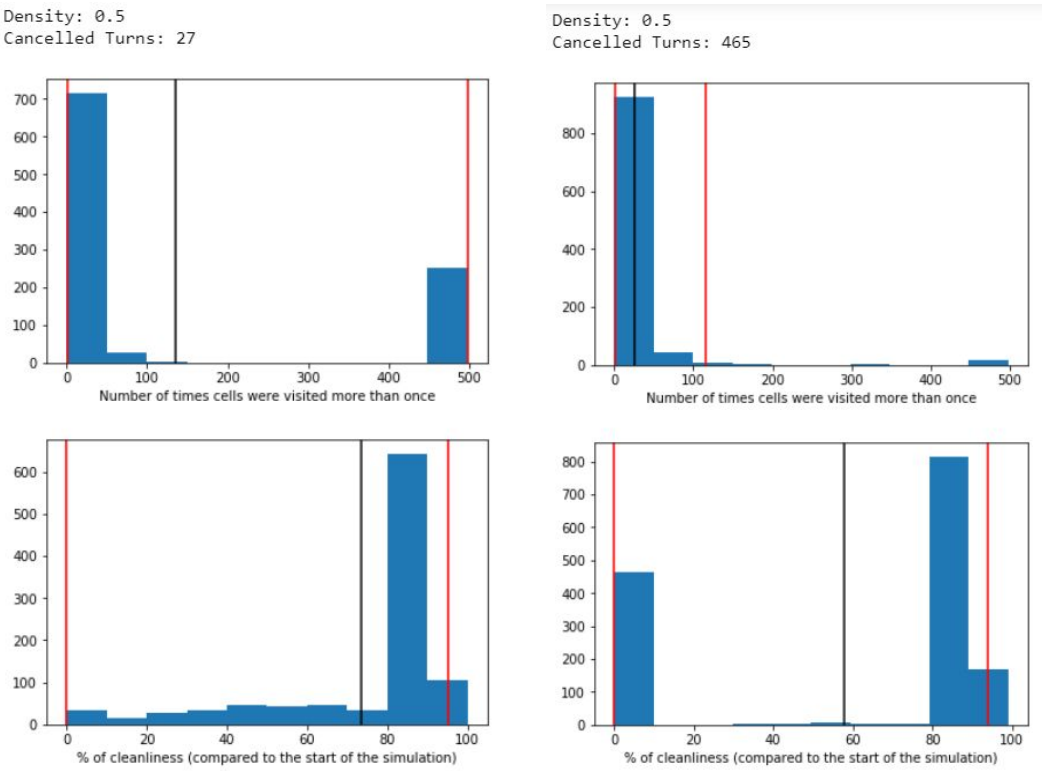
Average scenario: 50% of the room is occupied by obstacles









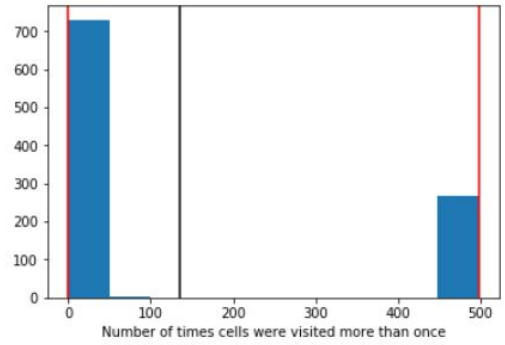


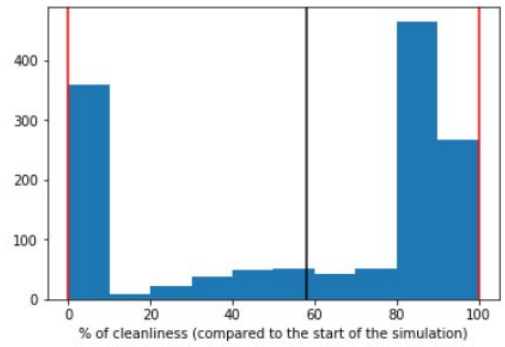
In the average case, distribution of the number of revisited cells and well as the distribution of the cleanliness of the room are quite irregular and hard to predict. Additionally, the confidence intervals for all the graphs, with the exception of the revisited cells' distribution for the fourth strategy, show a wide range of values most likely to be picked, incorporating the whole distributions; which makes it hard to narrow down our predictions for both the number of revisited cells and well as how clean the room can be. The fourth strategy, however, shows a tight confidence interval, which can indicate a more reliable strategy for average cases. However, it is the strategy that presents the highest number of simulation runs that had to restarted because the roomba could not move, 475 times to be more specific, which makes the roomba under this strategy more of a burden than an useful home appliance.

We can also notice that the values tend to group at both edges. In general, roomba either cleans a lot or doesn't clean at all; and either revisits all the cells or doesn't revisit at all. These clusterings can happen if the roomba is initialized surrounded by objects, and thus being forced to move around the same place repeatedly; or if the roomba is initialized far away from objects and the objects themselves are initialized all clustered together, freeing up the space for the roomba.

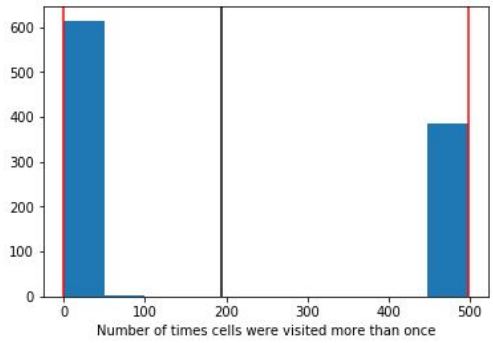
Worst-case scenario: 80% of the room is occupied by obstacles

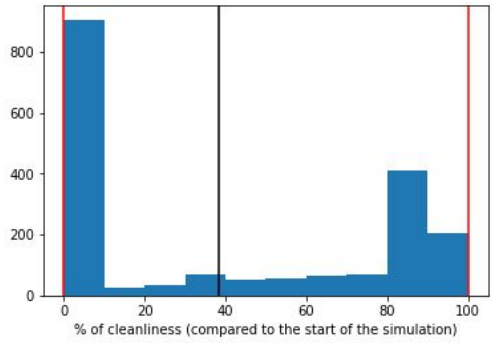
Density: 0.8
Cancelled Turns: 354

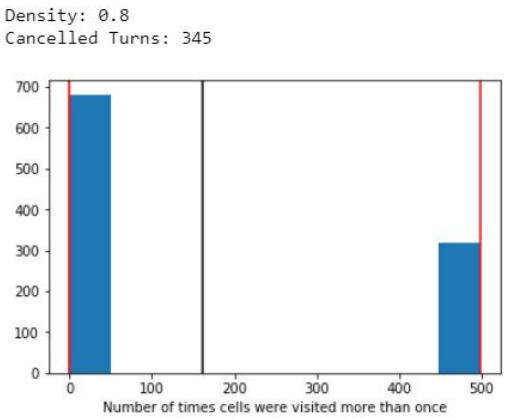


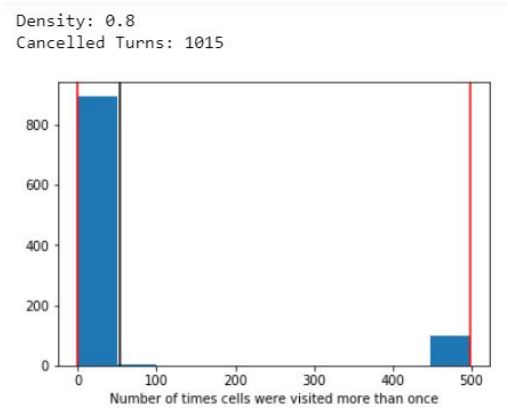


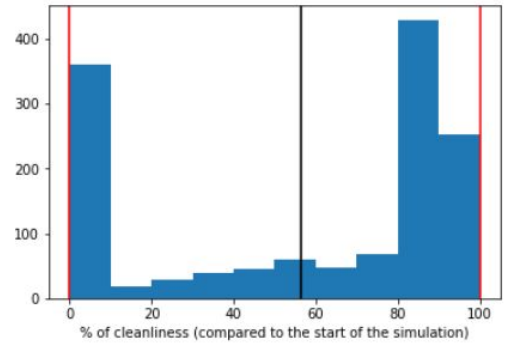
Density: 0.8
Cancelled Turns: 893

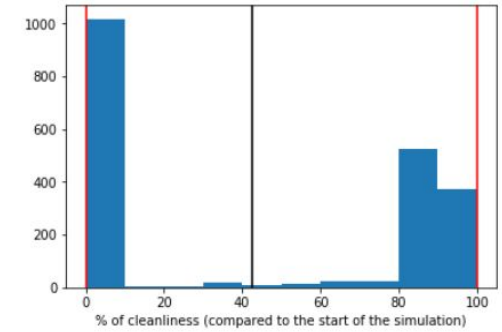










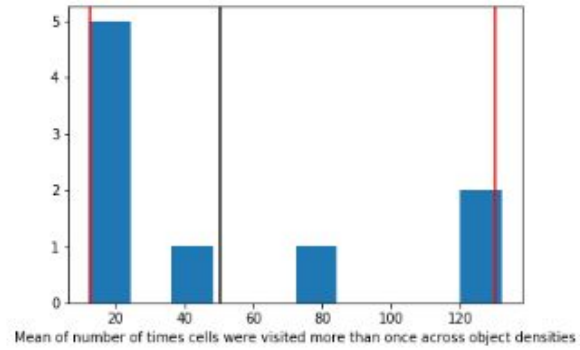


In the worst case scenario, we expect the room to be as full as it is possible while having the roomba running around at least once every 2000 simulation runs. Again, the fourth strategy presents the worst performance in terms of having the roomba unable to clean, with a total of 1015 simulation runs having to be cancelled because of the roomba's inability to move. None of the strategies perform particularly well, but the first and third strategy present the highest cleanliness, leaving the room an average of 60% cleaner. However, the cleanliness distribution for both of these strategies are clustered at the edges, presenting a wide confidence interval; and thus we can't guarantee that the average cleanliness of 60% is well representative of a real life scenario.

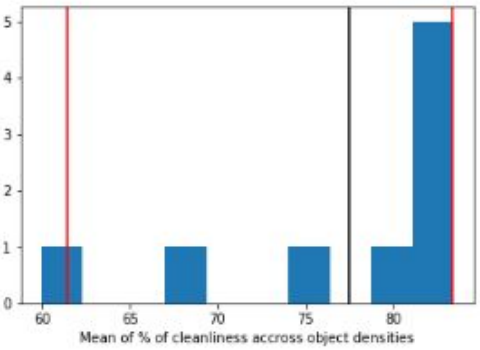
Summary

First strategy:

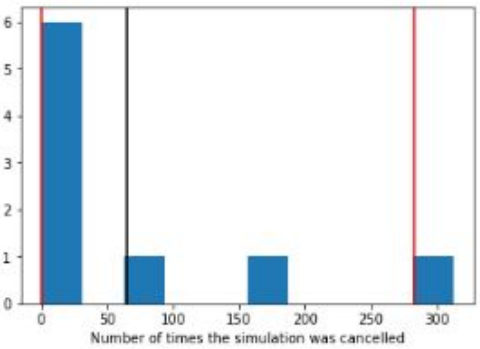
Run Summary:



Mean of number of times cells were visited more than once across object densities



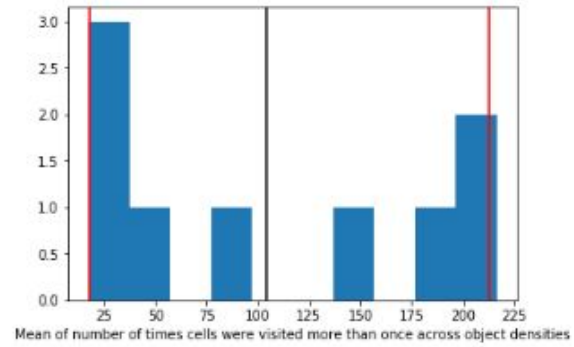
Mean of % of cleanliness accross object densities

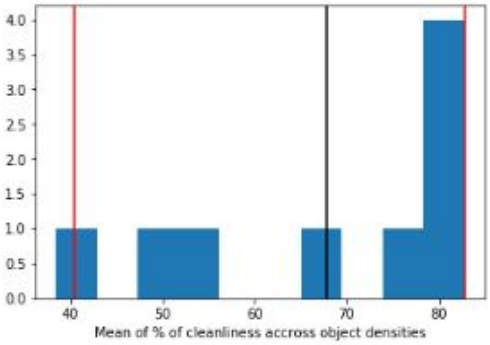


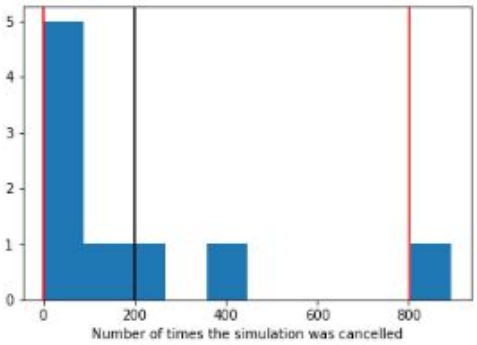
Number of times the simulation was cancelled

Second strategy:

Run Summary:

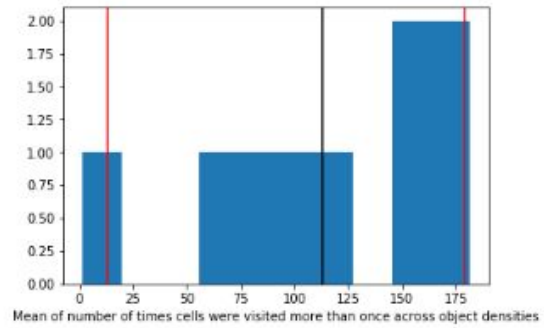




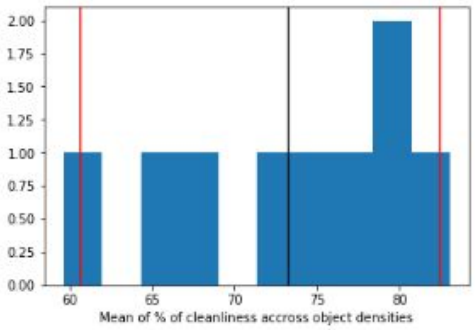


Third strategy:

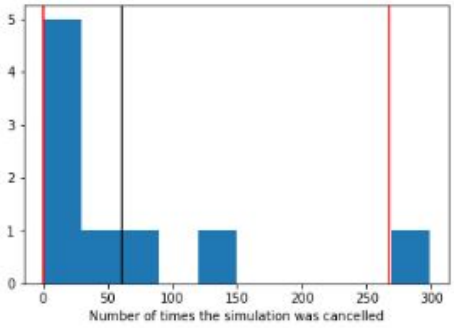
Run Summary:



Mean of number of times cells were visited more than once across object densities

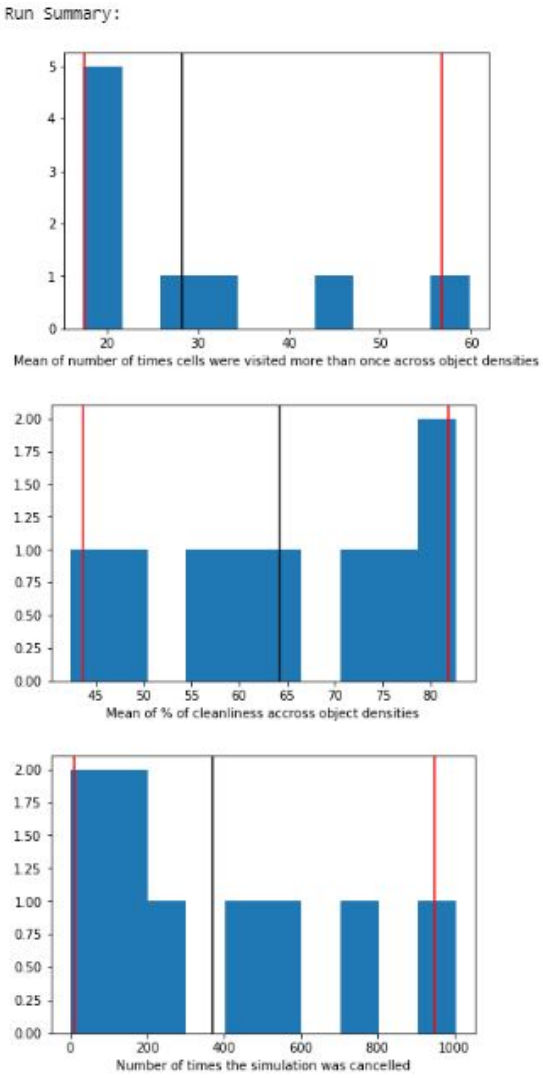


Mean of % of cleanliness accross object densities



Number of times the simulation was cancelled

Fourth strategy:



As the wide confidence intervals shows, it is hard to predict with certainty how well each strategy will perform in terms of room cleanliness and revisiting cells. However, of all the strategies, the first one presents is the one that seems most promising. It shows the highest number of times when the room is between 85% and 95% cleaner; presenting also the lowest number of average revisited cells. The fourth strategy, on the other hand, presents the worst performance. Not only its cleaning factor is unreliable (it can clean 45, 50, 55, 60, 65, 70, and 75% of the room with equal probabilities) and it presents the highest number of cancelled simulations; with an average of 400 simulations that had to be restarted because the

roomba couldn’t even make the initial move. Out of the 4 strategies, the first one presents the most utility achieved by the roomba, and thus should be prioritized over the other strategies.

It is important to note, however, that this simulation only consider a simplified version of the environment that the roomba would be exposed to in real life, such as having moving objects. Also, it doesn’t take into consideration the life battery of the roomba, which might run out before the roomba is done cleaning. Finally, the roomba is restricted to a small 6 by 6 room, and larger rooms might make the roomba escalate highly undesired behaviors that would turn it unusable.