1_simple-randomized-agent_LOWSLEY

September 14, 2020

1 Intelligent Agents: Vacuum-cleaner World

Implement a simulator environment for a vacuum-cleaner world and a set of intelligent agents.

1.1 PEAS description

Performance Measure: Each action costs 1. The performance is measured as the sum of the cost to clean the whole environment.

Environment: A room with $n \times n$ squares where $n = 5$. Dirt is randomly placed on each square with probability $p = 0.2$. For simplicity, you can assume that the agent knows the size of the layout of the room (i.e., it knows n and where it starts).

Actuators: The agent can `clean` the current square or move to an adjacent square by going `north`, `east`, `west`, or `south`.

Sensors: Four bumper sensors, one for north`,`east`,`west`, and`south‘; a dirt sensor reporting dirt in the current square.

The easiest implementation for the environment is to hold an 2-dimensional array to represent if squares are clean or dirty and to call the agent function in a loop untill all squares are clean or a predefined number of steps have been reached.

1.2 Define the agent program for a simple randomized agent

The agent program is a function that gets sensor information (the current percepts) as the arguments. The arguments are:

- A dictonary with boolean entries for the for bumper sensors `north`, `east`, `west`, `south`; not specified bumpers are assumed to be `False`. E.g., if the agent is on the north-west corner, `bumpers` gets `{"north" : True, "west" : True}` or if the agent is not close to a border then it gets `{}`.
- The dirt sensor produces a boolean.

The agent returns the chosen action as a string.

Here is an example implementation for the agent program of a simple randomized agent:

[28]:
```
from numpy import random

actions = ["north", "east", "west", "south", "suck"]
```

```
def simple_randomized_agent(bumpers, dirty):
    return random.choice(actions)
```

[29]:
```
simple_randomized_agent({"north" : True}, True)
```

[29]:
```
'south'
```

1.3 Simple environment example

The environment is infinite in size (bumpers are always `False`) and every square is dirty. We run the agent for 10 times steps.

[30]:
```
for i in range(10):
    print(simple_randomized_agent({"north" : False, "south" : False, "west" :␣
 ↪False, "east" : False}, True))
```

```
south
north
east
north
south
east
west
suck
south
suck
```

2 Tasks

Submission Instructions: Use this notebook to prepare your submission. Complete this section with your code and results. You can use Markdown blocks for your description, comments in the code and use mathplotlib to produce charts. If you use external code files then you can include them with

```
from notebook import psource
psource("your_file.py")
```

Note: Try to keep the code simple! In this couse, we want to learn about the algorithms and we often do not need to use object-oriented design.

2.1 Task 1: Implement a simulation environment

Your environment simulator needs to create squares, make some dirty, and proivde the agent function with the sensor inputs. The environment needs to evaluate the performance measure. It needs to track the agent's actions until all dirty squares are clean and count the number of actions it takes the agent to complete the task.

The simulation environment needs to work with the simple randomized agent program from above.

[46]:
```
import numpy as np
import copy
import matplotlib.pyplot as plt
```

[33]:
```
# define choices
dirty_choices = [True, False]

def CreateEnvironment(agent, n = 5, p = .2, position = [0,0], max_steps = 1000,␣
 ↪debug = True):
    global memory
    global pos
    if(p > 1):
        print("Error, dirty prob is > 1")
    else:
        # set up 2d matrix w/ random dirty squares
        # squares = [[random.choice(dirty_choices, p=[p, 1-p]) for x in␣
 ↪range(n)] for y in range(n)]
        squares = np.random.choice(a=[True, False], size=(n,n), p=[p,1-p])
        # find total number of dirty squares where true = 1
        need_clean = np.sum(squares)
        if(position != [0,0]):
            pos = position
        else:
            pos = [0,0]
        if(debug):
            print(need_clean, " dirty squares")
            print("Starting at {}".format(pos))
            for row in squares:
                print(row)
            print("Begin")
        memory = []
        # start
        for i in range(max_steps):
            # set location bumpers
            bumpers = {
                "north": pos[0] == 0,
                "west": pos[1] == 0,
                "south": pos[0] == n-1,
                "east": pos[1] == n-1
            }
            # locate square and find if dirty
            dirt = squares[pos[0]][pos[1]]
            if(debug):
                print("--------------------------")
                print("Iteration: ", i)
                print("Current location: ", pos)
                print("Bumper status: ", bumpers)
                print("Square dirty? ", dirt)
            # call to agent function
            action = agent(bumpers, dirt)
            if(debug):
                print("robot did action: ",action)
            # move robot according to action
            if(action == "north" and pos[0]>0):
                pos[0] = pos[0]-1
            if(action == "south" and pos[0]<(n-1)):
                pos[0] = pos[0]+1
            if(action == "east" and pos[1]<(n-1)):
                pos[1] = pos[1]+1
            if(action == "west" and pos[1]>0):
                pos[1] = pos[1]-1
            if(action == "suck"):
                squares[pos[0]][pos[1]] = False
            # check number of clean squares
            need_clean = np.sum(squares)
            if(debug):
                print("need to clean {} more squares".format(need_clean))
            # check if done
            if(need_clean == 0):
                if(debug):
                    print("Environment clean!")
                break
        return(i+1)
```

2.2 Task 2: Implement a simple reflex agent

The simple reflex agent randomly walks around but reacts to the bumper sensor by not bumping into the wall and to dirt with sucking.

[34]:
```
def simpleReflexAgent(bumpers, dirty):
    if(dirty):
        return("suck")
    else:
        # NW
        if(bumpers['north'] and bumpers['west']):
            choice = np.random.randint(low = 0, high = 2)
            if(choice == 0):
                return('south')
            if(choice == 1):
                return('east')
        # NE
        elif(bumpers['north'] and bumpers['east']):
            choice = np.random.randint(low = 0, high = 2)
            if(choice == 0):
                return('south')
            if(choice == 1):
                return('west')
        # SW
        elif(bumpers['south'] and bumpers['west']):
            choice = np.random.randint(low = 0, high = 2)
            if(choice == 0):
                return('north')
            if(choice == 1):
                return('east')
        # SE
        elif(bumpers['south'] and bumpers['east']):
            choice = np.random.randint(low = 0, high = 2)
            if(choice == 0):
                return('north')
            if(choice == 1):
                return('west')
        # N
        elif(bumpers['north']):
            choice = np.random.randint(low = 0, high = 3)
            if(choice == 0):
                return('east')
            if(choice == 1):
                return('west')
            if(choice == 2):
                return('south')
        # E
        elif(bumpers['east']):
            choice = np.random.randint(low = 0, high = 3)
            if(choice == 0):
                return('north')
            if(choice == 1):
                return('west')
            if(choice == 2):
                return('south')
        # S
        elif(bumpers['south']):
            choice = np.random.randint(low = 0, high = 3)
            if(choice == 0):
                return('east')
            if(choice == 1):
                return('west')
            if(choice == 2):
                return('north')
        # W
        elif(bumpers['west']):
            choice = np.random.randint(low = 0, high = 3)
            if(choice == 0):
                return('east')
            if(choice == 1):
                return('north')
            if(choice == 2):
                return('south')
        # No bumper reading
        else:
            choice = np.random.randint(low = 0, high = 4)
            if(choice == 0):
                return('east')
            if(choice == 1):
                return('west')
            if(choice == 2):
                return('south')
            if(choice == 3):
                return('north')
```

2.3 Task 3: Implement a model-based reflex agent

This agent keeps track of the location and remembers where it has cleaned. Assume the agent knows how many squares the room has and where it starts. It can now use more advanced navigation.

How will your agent perform if it is put into a larger room, if the room contains obstacles, or it starts in a random square?

[35]:
```
# NOTE: uses global position, size, and memory variables per each run.
def modelBasedAgent(bumpers, dirty):
    # add memory location if not already in list
    if(memory[-1:] != [pos]):
        memory.append(pos.copy())
    # first, if square is dirty, clean and marked as visited location
    if(dirty):
        return("suck")
    # otherwise move forward with decision making process
    else:
        # if west side and NW or SW corner go east
        if(bumpers["west"] and len(memory)==1):
            return("east")
        # if east side and NE or SE corner go west
        if(bumpers["east"] and len(memory)==1):
            return("west")
        # if not against east wall, and prev spot is to west, move east
        if(not(bumpers["east"]) and ((memory[-2:][0])[1] == (pos[1]-1))):
            return("east")
        # if not against west wall, and prev spot is to east, move west
        if(not(bumpers["west"]) and ((memory[-2:][0])[1] == (pos[1]+1))):
            return("west")
        # if against west wall but previous spot does not match y, move east
        if(bumpers["west"] and (memory[-2:][0])[0] != pos[0]):
            return("east")
        # if against east wall but previous spot does not match y, move west
        if(bumpers["east"] and (memory[-2:][0])[0] != pos[0]):
            return("west")
        # if no bumpers east or west but y coord changed, move east or west␣
↪(obstacle was hit)
        if(not(bumpers["east"]) and not(bumpers["west"]) and (memory[-2:
↪][0])[0] != pos[0]):
            return(random.choice(["east", "west"], p=[.5, .5]))
        # if starting on top, move south, else, move north
        if(memory[0][0] == 0 and not(bumpers["south"])):
            return("south")
        elif not(bumpers["north"]):
            return("north")
        # safety condition to at least move robot if stuck
        avail_moves = [k for k,v in bumpers.items() if v == False]
        return random.choice(avail_moves)
```

How will your agent perform if it is put into a larger room? My agent, when put into a larger room, is still able to clean the entire room so long as the minimum steps specified is large enough to cover

x a move for every square $+ y$ the total number of dirty squares $- 2$ (for start and stop)

For example, if the room is 5x5 and has 4 dirty squares, you would need a minimum of $ 25 + 4 - 2 = 27 $ steps. It starts in one corner and will zig zag its way to completion in another corner.

How will your agent perform if the room contains obstacles? The agent was not built with obstacles in mind, it may run randomly until the maximum allowed steps is reached. It will most likely not perform well at all due to them and may never finish cleaning the environment.

How will your agent perform if it starts in a random square? The agent can be placed in any corner square. If it is placed in another square other than those 4 then it will zigzag by going straight horizontally, down or up, and then horizontally again until it reaches the last corner in its direction. When it hits this corner it will get stuck.

2.4 Task 4: Simulation study

Compare the performance of the agents using different size environments. E.g., 5×5, 10×10 and 100×100. Use at least 100 random runs for each.

[42]:
```
# define constraints
number_runs = 100
```

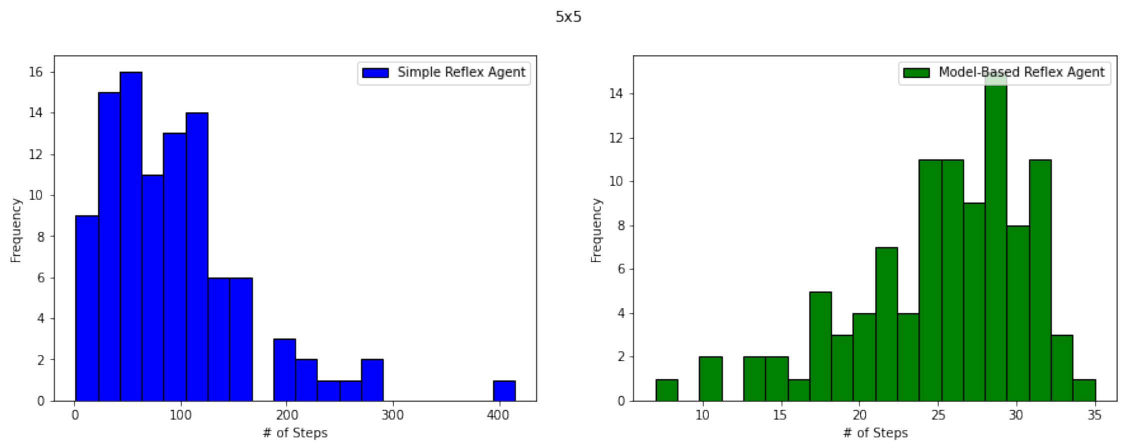
```
number_models = 3
sim_sizes = [5, 10, 50, 100]

# Holds our number of steps taken for each model for the number of runs
steps_taken = [np.repeat(0,number_runs) for i in range(number_models)]
# holds all data for all environments
all_data = {}

# for given environment size
for curr_env in sim_sizes:
    print("Running {}x{}".format(curr_env,curr_env))
    # define max steps to take, will cap at number of moves and cleaning of␣
↪dirt
    #   needed for square if every square was dirty or 10,000
    max_steps_env = max(10000, env**2 + env**2)
    # loop through number of runs and save off results
    for i in range(number_runs):
        steps_taken[0][i] = CreateEnvironment(simple_randomized_agent, n =␣
↪curr_env, max_steps = max_steps_env, debug = False)
        steps_taken[1][i] = CreateEnvironment(simpleReflexAgent, n = curr_env,␣
↪max_steps = max_steps_env, debug = False)
        steps_taken[2][i] = CreateEnvironment(modelBasedAgent, n = curr_env,␣
↪max_steps = max_steps_env, debug = False)
    # save off env data to dictionary
    all_data[curr_env] = copy.deepcopy(steps_taken)
```

```
Running 5x5
Running 10x10
Running 50x50
Running 100x100
```

[47]:
```
for env, steps in all_data.items():
    # visualize results
    print("--------------------------------------")
    print("Environment: {}x{}".format(env,env))
    print("Steps Taken: ")
    print("{:30}{:10}{:10}{:10}".format("Agent", "Min", "Max", "Average"))
    print("{:30}{:10}{:10}{:10}".format("Simple Random Agent", np.
↪min(steps[0]), np.max(steps[0]), np.mean(steps[0])))
    print("{:30}{:10}{:10}{:10}".format("Simple Reflex Agent", np.
↪min(steps[1]), np.max(steps[1]), np.mean(steps[1])))
    print("{:30}{:10}{:10}{:10}".format("Model-Based Reflex Agent", np.
↪min(steps[2]), np.max(steps[2]), np.mean(steps[2])))

    fig_main = plt.figure(figsize=(15,5))
    fig_main.suptitle("{}x{}".format(env,env))
    fig1 = fig_main.add_subplot(1,2,1)
```

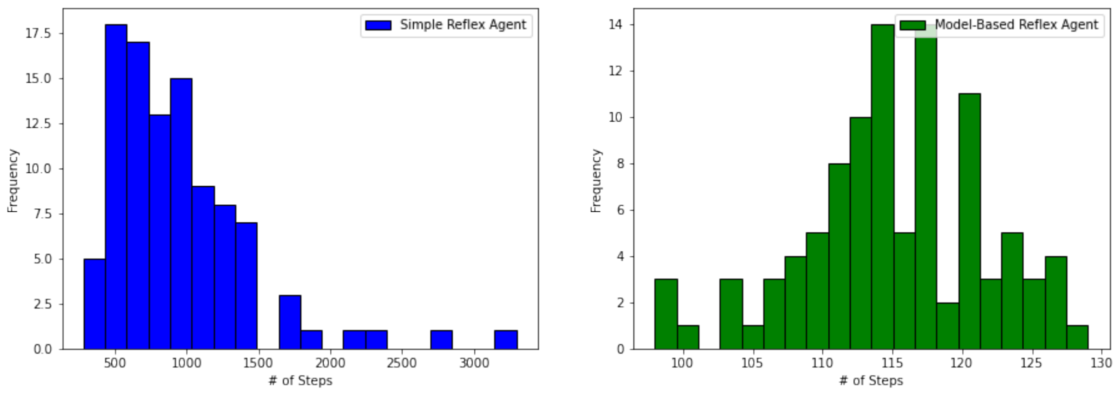
```python
    fig2 = fig_main.add_subplot(1,2,2)

    fig1.hist(steps[1], bins = 20, label = ["Simple Reflex Agent"], color =␣
↪"blue", ec = "black")
    fig1.set_xlabel("# of Steps")
    fig1.set_ylabel("Frequency")
    fig1.legend(loc = "upper right")

    fig2.hist(steps[2], bins = 20, label = ["Model-Based Reflex Agent"], color␣
↪= "green", ec = "black")
    fig2.set_xlabel("# of Steps")
    fig2.set_ylabel("Frequency")
    fig2.legend(loc = "upper right")

    plt.show()
```

```
-------------------------------------
Environment: 5x5
Steps Taken:
Agent                          Min       Max       Average
Simple Random Agent                     1      1504      455.0
Simple Reflex Agent                     1       415      93.59
Model-Based Reflex Agent                7        35      25.14
```



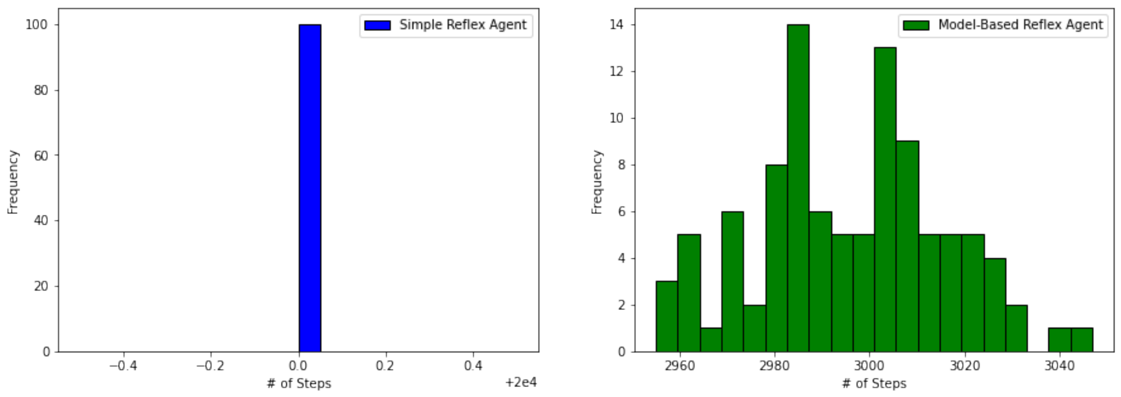
```
-------------------------------------
Environment: 10x10
Steps Taken:
Agent                          Min       Max       Average
Simple Random Agent                   822      9460    3278.32
Simple Reflex Agent                   281      3298     943.45
Model-Based Reflex Agent               98       129     115.03
```

10x10

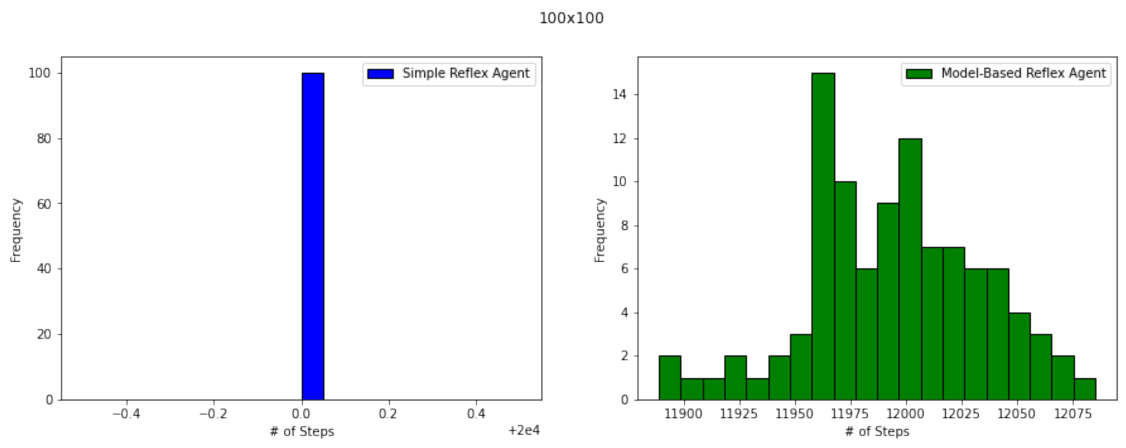


```
-------------------------------------
Environment: 50x50
Steps Taken:
Agent                          Min        Max        Average
Simple Random Agent                20000      20000    20000.0
Simple Reflex Agent                20000      20000    20000.0
Model-Based Reflex Agent            2955       3047    2995.74
```

50x50



```
-------------------------------------
Environment: 100x100
Steps Taken:
Agent                          Min        Max        Average
Simple Random Agent                20000      20000    20000.0
Simple Reflex Agent                20000      20000    20000.0
Model-Based Reflex Agent           11889      12085    11993.5
```

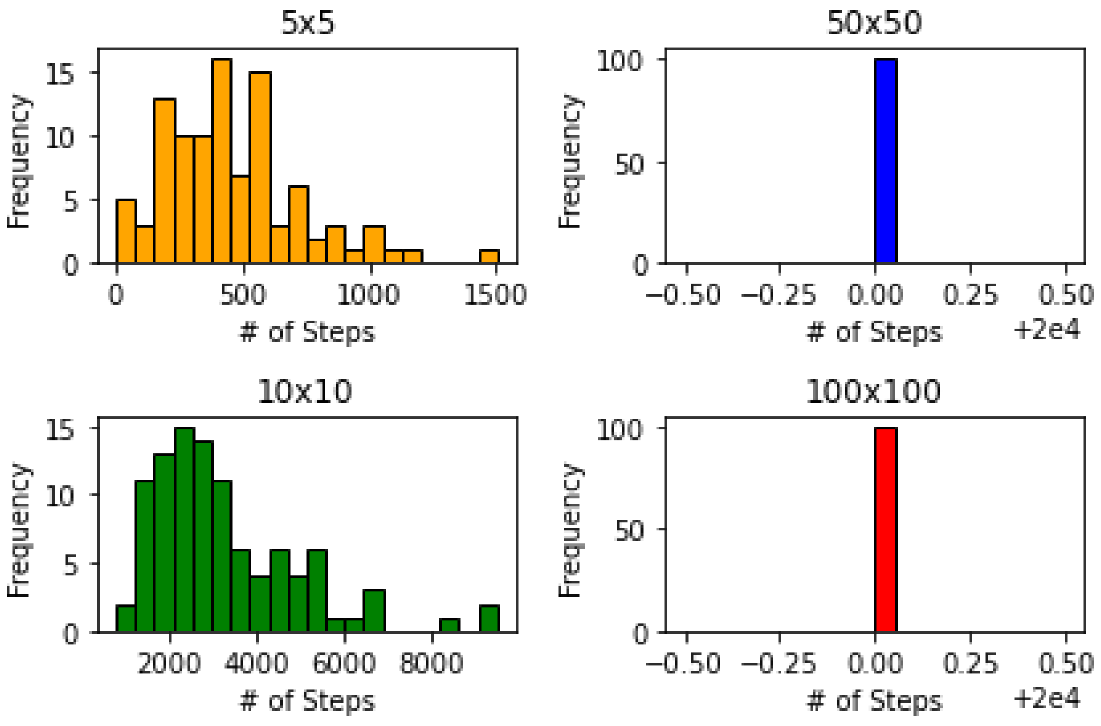


As seen in the data portrayed above, the model based reflex agent performs the absolute best, followed by the simple reflex agent, and then followed by the simple random agent. In each simulation, a given square had a 20% chance of being dirty on creation of an environment and there were no obstacles palced. It should be noted that for each run of an environment, there could be more dirty squares than in other runs. This could explain why the minimum number of steps taken on the model-reflex based agent is so high. Also, when running the agents on a floor sized 50x50 and up we see that really the only agent that could really even be used in larger sizes is the model-based reflex agent. The simple reflex agent can barely function and reaches it's cap in the number of steps required.

```
[61]: env5 = all_data[5][0]
env10 = all_data[10][0]
env50 = all_data[50][0]
env100 = all_data[100][0]
print("Simple Random Agent")
fig, axs = plt.subplots(2, 2)
axs[0, 0].hist(env5,bins = 20, label = ["Simple Random Agent"], color =␣
 ↪"orange", ec = "black")
axs[0, 0].set_title("5x5")
axs[0, 0].set_xlabel("# of Steps")
axs[0, 0].set_ylabel("Frequency")
axs[1, 0].hist(env10,bins = 20, label = ["Simple Random Agent"], color =␣
 ↪"green", ec = "black")
axs[1, 0].set_title("10x10")
axs[1, 0].set_xlabel("# of Steps")
axs[1, 0].set_ylabel("Frequency")
axs[0, 1].hist(env50,bins = 20, label = ["Simple Random Agent"], color =␣
 ↪"blue", ec = "black")
axs[0, 1].set_title("50x50")
axs[0, 1].set_xlabel("# of Steps")
axs[0, 1].set_ylabel("Frequency")
```

```
axs[1, 1].hist(env100,bins = 20, label = ["Simple Random Agent"], color =␣
 ↪"red", ec = "black")
axs[1, 1].set_title("100x100")
axs[1, 1].set_xlabel("# of Steps")
axs[1, 1].set_ylabel("Frequency")
fig.tight_layout()
```

```
Simple Random Agent
```



As seen in the graphs depicted above, the performance of the simple random agent just performs terribly. Therefore, it is not even really worth it to compare it to that of the performance of the simple reflex agent or the model-based agent. I would maybe consider using this type of agent in an extremely small space but even there it has no decision process for its actions, making it a poor robot to choose.

For the simple reflex agent, it performed somewhat ok but really struggled in the larger room situations. It may be a perfectly fine choice to use if you have a very small space since it at least acts based off its percepts but it is still a poor choice

For the model based reflex agent, we saw the best results and it would be the optimal choice of the three for really any room type. The agent is set to be able to handle obstacles (pretty terribly but is still programmed to do so) and actually remembers where it has been to better assist it in making a decision on where to go. It will usually not go over locations it has been to and does need memory in order to function. However, as we can see in the graphs above, it actually had a fairly decent average for the number of steps taken. In the 100x100 case, the average was 11,993.5

which is really not bad considering the total squares in that environment was 10,000.

2.5 Bonus tasks

- **Obstacles:** Add random obstacle squares that also trigger the bumper sensor. The agent does not know where the obstacles are. How does this change the performance?
- **Unknown environment with obstacles:** The agent does not know how large the environment is, where it starts or where the obstacles are.
- **Utility-based agent:** Change the environment, so each square has a fixed probability of getting dirty again. Give this information to the agent (as a 2-dimensional array of probabilities). Cleaning one dirty square produces the utility of 1. Implement a utility-based agent that maximizes the expected utility over a time horizon of 10000 time steps. This is very tricky!

[ ]:
```
# Was unable to really do any of these, sorry!
```