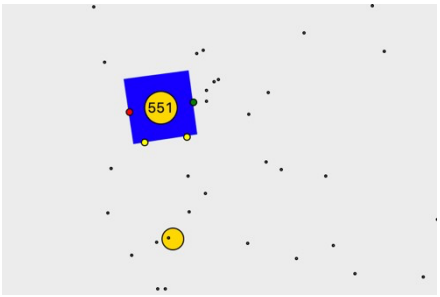
Introduction

The work in this session is designed to explore the idea of reactive + state agents. That is, agents that act in a "reactive" way by taking in percepts and carrying out actions, but which also have an **internal memory**. This memory can be modified by the things that they perceive or the state that they find themselves in, and the memory can act as an additional factor in deciding which action to take at each time-step. We will also start to think about the idea of "planning"—looking beyond the immediate time-step to plan a sequence of steps. Finally, this worksheet also introduces the idea of carrying out an experiment to compare to different approaches to a given task, so that we can understand objectively whether a particular change that we have made makes a difference.

There is a new kind of object in the simulation: Dirt, depicted as small grey circles. The tasks in this worksheet treat the robot as a vacuum cleaner, which picks up that dirt as it moves.

Getting Started

Download the file called *simpleBot2.py* from the module Moodle page, and **make sure that you can run it.** You can run this from the command line, or import the file into an IDE. When you run it, a window should appear, containing one robot, one charger (the gold-coloured circle), and lots of "dirt" spots.



Note that the robot "vacuums up" the dirt as it passed over it.

Tasks

The code that you have been provided with is a slight extension of the code from last week's exercise. The "battery" function has been added, and a simple mechanism to keep robots within the window has been implemented (if a robot goes off one side of the window, it re-enters on the other side). The Lamp function has been removed because it is not relevant to the tasks in this worksheet. The initial behaviour implemented in transferFunction is a wandering behaviour, where the robot travels in a random forward direction for a certain number of timesteps, then chooses another direction and travels in that direction. Note that this behaviour requires a simple memory, i.e. of the number of moves since the last turn or forward move. We have also added a function that enables you to move the bot by clicking on the screen, to help with testing. There are also two "WiFi hubs" which work similar to the lamps in the last worksheet, and which will be needed for the second task.

Monitoring the Dirt Collected

In this worksheet, we are going to start to think about how we can do experiments with different behaviours. An important part of experiment is measurement. So, let's measure the amount of dirt collected.

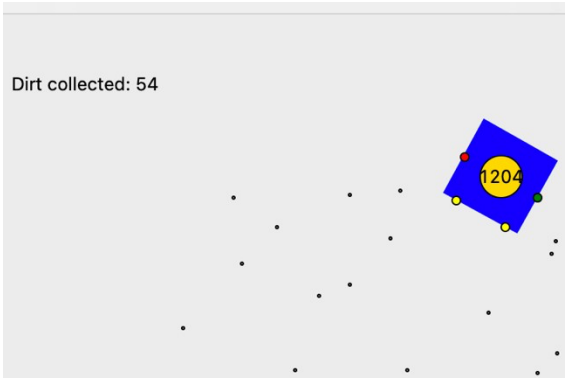
1. **Create a class called *Counter,*** which is initialised with a variable called dirtCollected, which is **initialised to zero** in the *__init__(self)* method. This is going to represent the dirt collected by *all* of the robots in the simulation; this is why we don't just use a variable inside the robot object.
2. Now **create a method in that class called *itemCollected*** which increases dirtCollected by 1. This should take five lines of code in total.

3. **Create an instance** of the *Counter* class in the register function (call this *count*). Now, look at the *collectDirt* method in the *Bot* class. You will need to pass *count* out of the *register* function in the return statement alongside *registryActives* and *registryPassives*
4. **Modify the *main* method** to receive that output. Then, make a change to *moveIt* to also have *count* as a parameter (you will also need to change the *canvas.after* method).
5. Finally, change the *collectDirt* method in *Bot* to have *count* as a parameter. **Modify the *collectDirt* method** so that every time the dirt is collected (this is the condition *if self.distanceTo(rr)<30:*) then the *itemCollected* method is called. Just after this, **add a *print* statement** to print *count.dirtCollected*. **Run the program** and look at the printout to make sure that the *count.dirtCollected* is going up as the robot collects more dirt.

Now, let's display this in the window.

1- **Add *canvas* as a parameter** to the *itemCollected* method in *Counter*.

2- **Use the *canvas.createText* method** to write the current amount of dirt collected in a corner of the window. You can see an example of how to use *createText* in the *draw* method below. There are a number of ways to do the update. You could delete the current value, then redraw, or you could look up how to use the *itemconfigure* method in tk to change the value of the text. This should look something like this:



Finally, an important aspect of experiments is to make sure that all other factors are taken into account equally. So, you should **place a limit on the time that the program will run** (this is sometimes referred to as a "budget of computations"). The simplest way to do this is to have the program count the number of moves made, and stop after a certain number (say, 5000). To make the program stop you can use a command such as sys.exit() Alternatively, to stop the program after a certain period of time, by using the functions in the time library. Once you have implemented this, **compare the amount of dirt** collected by one robot alone, two, three, four etc.

This kind of experimentation will be an important part of the coursework. You will not only be required to implement an agent system to carry out a task, but you will be asked to formulate a question to test by doing experiments on your system. For example, in this case, the question is "how does the amount of dirt collected change when you vary the number of robots?".

We hope that most of you will get to this stage by the end of the class, and that most of you will make a start on the next stage.

Building up a Map

One way for an agent to increase its "intelligence" is for it to memorise an aspect of its environment, and then to use that memory to carry out some kind of planning process. One kind of memory is a map of the environment, which is what we will explore now. You can either start with your solution to the previous exercise, or you can build on *simpleBot2_withCounting.py*, which you can download from the Moodle page and which contains a solution to the tasks so far.

What we will do is to map the regions that the robot has already explored, by dividing the space down into a 10x10 square regions and marking them as explored or not.

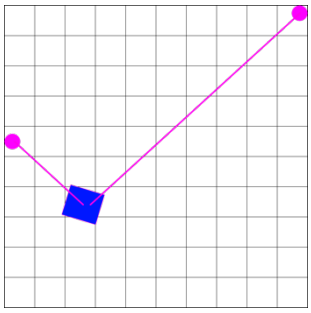
1- **Start by adding an object to the Bot object called *self.map.*** The best structure for this is the *numpy* array, and so we can create this (initially full of zeros) by using the statement *self.map = np.zeros((10,10))* in the *Bot __init__* method.

2- Now, each time the robot passes through one of these regions, we will change the 0 to a 1 to indicate that it has been covered. The simplest way to do this is to assume that the robot has a knowledge of its (x,y) position, e.g. perhaps through some kind of GPS or similar. This is an oversimplification that we will correct later. So, each time the *move* method is executed, let's see if the map needs updating. **Add a method** *updateMap(self)* to the Bot. Now, we are going to have to calculate which square of the map we are currently in. The canvas is 1000x1000 pixels, divided into 10x10 map squares. So, we will need to divide the current position by 100 to get the map position. Let's do this like this: *xMapPosition = int(math.floor(self.x/100))* . **Add this, and a corresponding one for *self.y*, to the *updateMap* method.** Then, **set the relevant cell in the map array to 1**, which will indicate that this map square has been visited. **Call *updateMap*** at the end of each *move* method.

3- It would be good to visualise this, to make sure that it is working. So, **add another method** called drawMap(self) to the *Bot* class. Using two nested for loops, get each value from the map array, and **if it is 1, draw a rectangle**. This is rather complicated, so here is the code for this.
*self.canvas.create_rectangle(100*xx,100*yy,100*xx+100,100*yy+100,fill="pink",width=0,tags="map")* .
Graphically, there is still a problem with this—we have drawn the rectangles on the top of the other things on the canvas. So, include the statement *self.canvas.tag_lower("map")* to put the map at the "back" of the visual scene (this is "z-buffer sorting" for those of you who are familiar with computer graphics ideas).

Now, you can use the map to guide the robot. Find a region that has not been explored, and in transferFunction direct the robot towards that (you can modify the "light following" code from the first worksheet). More advanced: find the *nearest* region that hasn't been explored, and move towards that region.

In practice, the robot doesn't "know" its (x,y) position like this. Instead, we use a localisation method. One method is to use the relative strength of wi-fi signals to work out the location of the bot. The strength of the two signals will vary from region to region in the map. For example, in the illustration below, the robot will be getting a strong signal from the hub on the left, but a weaker one from the one in the top right corner.



Write some code that takes a sample of the wi-fi strength in each square of the map, and then use this to decide which square the bot is in. This is a more realistic way of getting the position of the robot than assuming that we have access to an absolute position. This is a rather advanced task, and we would not anticipate that most of you will be able to get this far in the class.

Extensions

Here are some extensions to the tasks above. We are not anticipating that you will attempt these in the class, but they provide ideas that might form the beginnings of your coursework.

- Carry out some experiments and record the results carefully. For example, you might compare whether there is a difference between the robot wandering randomly, and the robot guided by the

map, or with different numbers of robots. Write a program that runs various experiments and automatically creates a table of the different results, or draws a chart of the progress of the different robots in gathering dirt. Writing code that allows you to carry out a number of different experiments and summarise the results will be really useful when it comes to the coursework.

- Multi-robot coordination. What if multiple robots share their map? What if you program it so that the robots try to keep a certain distance from each other, so that they implicitly cover different areas of the space.
- Robots have a "capacity" for the amount of dirt that they can handle, and they have to visit a "bin" to dump the waste. You will need to think about prioritisation—if a robot has both a full bin and a low battery, which task do we prioritise first, visiting the bin, or charging the battery? What if the bin is really close, and the charger a long way away?