CS1110 Spring 2015

Assignment 4: Due Thursday March 5 at 11pm

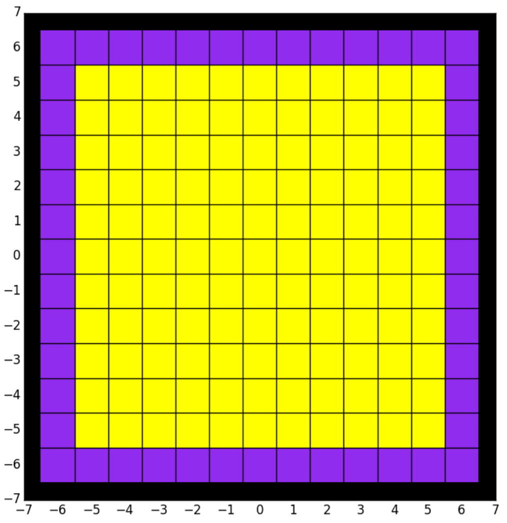
You must work either on your own or with one partner. If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group.

You may discuss background issues and general solution strategies with others, but the programs you submit must be the work of just you (and your partner). We assume that you are thoroughly familiar with the discussion of academic integrity that is on the course website. Any doubts that you have about "crossing the line" should be discussed with a member of the teaching staff before the deadline.

Objectives. Introduce while-loops, more practice with for-loops and functions. The assignment is based on Lectures 8, 9, and 10 and Lab 5.

1 Overview and Set-up

There is one module to submit: `RW2.py`. It is to include a number of functions that can be used to determine certain properties associated with a *2-dimensional random walk*. Many processes can be modeled with random walks—they are incredibly important throughout science and engineering. The random walk in this assignment can be described in terms of a robot that is trying to escape from a tiled "playpen":



A playpen of size n is a $(2n + 1)$-by-$(2n + 1)$ square centered at the origin (0,0). Its tiles are 1-by-1. The playpen displayed above has size 5. In all the figures in this handout, the playpen is yellow and it is surrounded by 1-by-1 purple tiles.

In the simulation, the robot starts at (0,0) and moves about by taking random hops like this:

- With probability .25, it hops east (E). This means that its current x-coordinate is increased by 1.
- With probability .25, it hops north (N). This means that its current y-coordinate is increased by 1.
- With probability .25, it hops west (W). This means that its current x-coordinate is decreased by 1.
- With probability .25, it hops south (S). This means that its current y-coordinate is decreased by 1.

Let's establish some terminology. The robot "escapes" when it is no longer on a yellow tile, i.e., when it hops onto a purple tile. This event terminates the simulation. The robot escapes "happy" if the last yellow tile on its journey is one of the four corner tiles. The robot is "homesick" if (by some measure) it is closer to a purple tile than to (0,0).

The module `RW2.py` that you are to develop will include functions designed to help answer these questions:

1. For a playpen of size n how many hops (on average) does it take for the robot to escape?
2. For a playpen of size n, what is the probability that the robot is happy when it escapes?
3. For a playpen of size n, how many hops (on average) does it take before the robot feels homesick for the first time?

Start by setting up a folder called `A4`. Into this folder download `simpleGraphicsE.py` and the template `RW2.py` from the assignments page on the course website. LET A4 BE THE CURRENT WORKING DIRECTORY WHENEVER YOU ARE WORKING ON THIS ASSIGNMENT.

2 Escape!

We can encode the robot's journey in what we'll call a *travel string*. A travel string is made up of the characters "`E`", "`N`", "`W`", and "`S`" and defines the sequence of hop directions. For example, `s = 'NESWSSES'` is a travel string that encodes the journey of a robot that escapes from a playpen of size 2:

k	`s[k]`	"Current" Location	"Next" Location
0	`N`	$(0,0)$	$(0,1)$
1	`E`	$(0,1)$	$(1,1)$
2	`S`	$(1,1)$	$(1,0)$
3	`W`	$(1,0)$	$(0,0)$
4	`S`	$(0,0)$	$(0,-1)$
5	`S`	$(0,-1)$	$(0,-2)$
6	`E`	$(0,-2)$	$(1,-2)$
7	`S`	$(1,-2)$	$(1,-3)$

Implement a function `MakePath(n)` that returns a random travel string for a robot that escapes from a playpen of size n. The travel string should be built up through repeated concatenation, e.g.,

`''`, `'N'`, `'NE'`, `'NES'`, `'NESW'`, `'NESWS'`, `'NESWSS'`, `'NESWSSE'` , `'NESWSSES'`

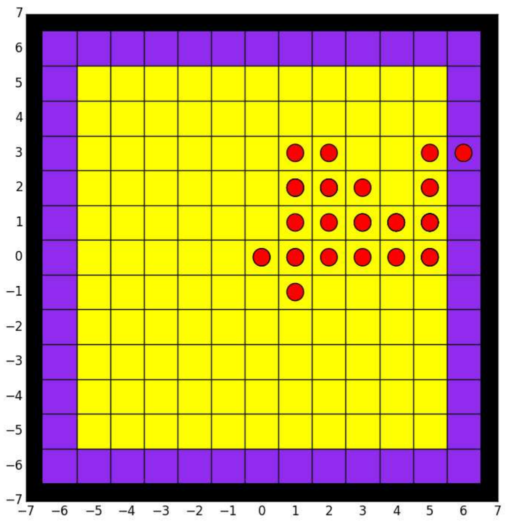
Use `randi` to generate the hop directions. This iteration must be under the control of a `while`-loop because the hopping simulation terminates as soon as the robot leaves the playpen, i.e., as soon as it hops onto a purple tile. Figure out how test for this event and design the while-loop accordingly. Take a hint from the above example where the simulation stops because the condition `abs(y) > 2` is `True`.

It turns out that the robot will always escape so strictly speaking, there is no danger of an infinite loop. Nevertheless, it is prudent to guard against excessively long escape routes and programming errors that do turn your while loop into an infinite loop. So during the debugging phase you may want to design `MakePath(n)` so that it terminates the simulation after 10000 hops and prints a message indicating the fact.

To facilitate the debugging of `MakePath(n)` you should make use of `simpleGraphicsE.py` and the procedure `DrawPlayPen(n)` that is provided in the template `RW2.py`. For example, an Application Script in `RW2.py` of the form

```
n = 5
MakeWindow(n+2,bgcolor=BLACK,labels=True)
DrawPlaypen(n)
s = MakePath(n)
ShowWindow()
```

could produce a figure like



provided you judiciously place statements of the form

```
DrawDisk(x,y,.25,color=RED)
```

in your implementation of `MakePath`. In particular, every time you update the robot's (x, y) location, draw a little red disk that marks the spot using `DrawDisk`. (Note that the display does not actually show the escape route—just the tiles that are visited along the way to the escape point.)

After you get `MakePath` working you can use it to estimate the average number of hops that are required for the robot to escape. Exploit the fact that there is a connection between the length of the string returned by `MakePath(n)` and the number of hops that are required for the robot to reach the purple zone. Encapsulate this computation in a function of the form

```
HopsAve(n,nTrials)
```

It should return a float that is an estimate of the average number of hops that are required for the robot to escape from a size n playpen. The average should be based on the number of trials that are specified by `nTrials`. Thus, `HopsAve(10,1000)` will compute 1000 times the number of hops it takes a robot to hop out of a size 10 playpen and return the average based on those simulations.

Set up the Application Script in `RW2.py` so that it is easy for you to try out different values for `n` and `nTrials`. You should notice that if the value of n is increased, then the average number of hops needed to escape also increases. Can you spot a correlation between the value of `n` and the value returned by `HopsAve(n,nTrials)`? Include a brief comment at the end of `HopsAve` that describes what you have discovered about this correlation. Advice: stay away from playpens that have size greater than 30—the simulations will just take too long. And setting `nTrials = 1000` is plenty big enough to see what's going on. And much smaller values of `nTrials` make sense while you are still debugging.

3 Happy?

If you look at a travel string carefully then you can deduce the coordinates of the robot's final resting location in the purple zone. Denote that final location by (x_f, y_f). For example, in the above example $(x_f, y_f) = (6, 3)$. Recall that a robot is "happy" if it exits the playpen from one of the four corner tiles. Here are some hints that can be used to identify when this is the case:

- By counting the number of `E`'s and `W`'s in the travel string you can figure out x_f
- By counting the number of `N`'s and `S`'s in the travel string you can figure out y_f
- By examining the values of $|x_f|$, $|y_f|$ and n, you can figure out if the last yellow tile on the escape route is a corner tile.

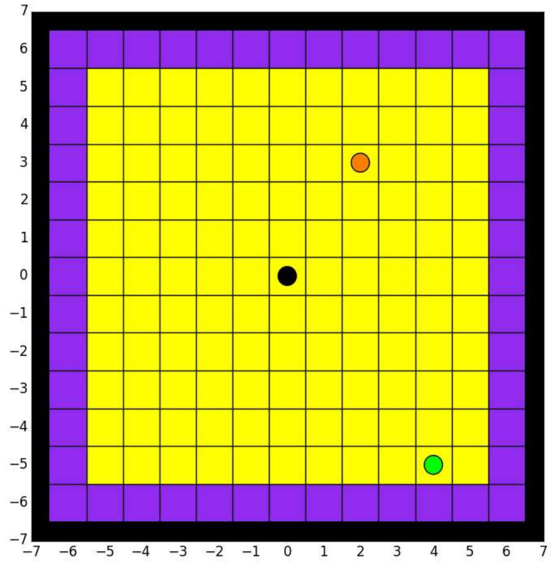
Implement a boolean-valued function `isHappy(s,n)` that takes a travel string `s` and the playpen size `n` and returns `True` if the robot escapes happy and `False` otherwise.

Making effective use of `isHappy`, implement a function `ProbHappy(n,nTrials)` that returns a float that is an estimate of the probability that a robot exits happy from a size n playpen. The estimate should be based on the number of trials specified by `nTrials`. Thus, `ProbHappy(10,1000)` will estimate the probability that a robot escapes happy from a size 10 playpen. The estimate would be based on 1000 experiments where an experiment involves turning a single robot loose and seeing if exits happy.

Note that a size n playpen has $8n$ tiles around its perimeter. If exiting is equally likely at each of these tiles, then the probability of exiting from a corner tile would be $1/(8n)$. That would mean that the probability of a happy exit would equal $4/(8n) = 1/(2n)$. Set up the Application Script in `RW2.py` so that it facilitates comparison between the value returned by `ProbHappy(n,nTrials)` and $p = 1/(2n)$. Experiment and share your discoveries in a brief comment at the end of `ProbHappy`.

4 Homesick?

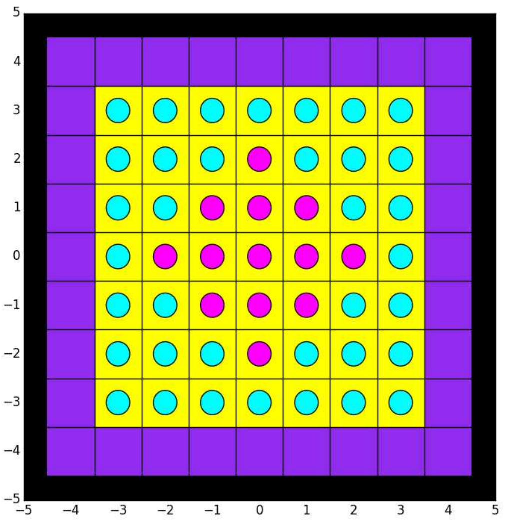
Think of (0,0) as the robot's home. If a robot is located at (x, y), then its *home distance* is $|x| + |y|$. This is the minimum number of hops required to reach (0,0). Its *edge distance* is the minimum number of hops needed to escape. Let's look at three examples:



Dot	Home Distance	Edge Distance
ORANGE	5	3
GREEN	9	1
BLACK	0	6

Figure out how to compute the edge distance given x, y, and n. Hint: it involves $n - |x|$ and $n - |y|$.

We say that a robot at (x, y) is *homesick* if its edge distance is strictly less than its home distance. Below is a size-3 playpen with cyan disks on the "homesick" locations and magenta disks on the "non-homesick" locations.



Implement a boolean-valued function `isHomesick(x,y,n)` that return `True` if a a robot on a size n playpen is homesick at (x, y).

After you get `isHomesick(x,y,n)` working, then proceed to implement a function

`Homesick(s,n)`

that inputs a travel string `s` and a playpen size n and returns an `int` that indicates the first hop that results in the robot becoming homesick. To clarify what this means, suppose a robot on a size 3 playpen generates travel string `s = 'NENEEWWWN'`. The following table indicates that first-time homesickness results after hop 2:

k	s[k]	Location After that Hop	Home Distance	Edge Distance	Homesick?
0	N	(0, 1)	1	3	No
1	E	(1, 1)	2	3	No
2	N	(1, 2)	3	2	Yes
3	E	(2, 2)	4	2	Yes
4	E	(3, 2)	5	1	Yes
5	W	(2, 2)	4	2	Yes
6	W	(1, 2)	3	2	Yes
7	W	(0, 2)	2	2	No
8	N	(0, 3)	3	1	Yes

In this case, the value of `Homesick('NENEEWWWN',3)` would be 2.

The table suggests how to organize your implementation of `Homesick`. A while loop will be in charge. It will generate the (x, y) locations that arise during the journey and use `isHomesick` to to look for the first occurrence of homesickness. The loop will terminate when this happens. Overall, the implementation of `Homesick` will look a lot like your implementation of `MakePath`. However, instead of generating the travel string as in `MakePath`, it will use the travel string to generate the (x, y) "stops."

Implement a function `HomesickAve(n,nTrials)` that estimates the average number of hops that are required for a robot to become homesick for the first time when trying to escape from a size n playpen. The average should be based on the number of trials specified by `nTrials`. How does `HomesickAve(n,nTrials)` compare to `n`? Include a brief comment at the end of `HomesickAve` that summarizes what you have discovered about this through simulation.

5 Submission to CMS

The final version of `RW2.py` that you submit to CMS should have these seven functions:

10pts	`MakePath`	
5pts	`HopsAve`	With comment on correlation between n and the average.
5pts	`isHappy`	
5pts	`ProbHappy`	With comment on the connection between $1/(2n)$ and the probability.
5pts	`isHomesick`	
5pts	`Homesick`	
5pts	`HomesickAve`	With comment on the connection between n and the average.

Remove the Application Script–we'll test your functions our own way.