



#### Unit 6: Rover's Coordinates

#### Application: A Random Walk

In this application, you will see how often Rover can travel on the grid (making random turns north and east only) from the origin (0, 0) to the point (2, 2). You will also investigate the experimental and theoretical probabilities of reaching that point at random.

#### Objectives:

- Use a random number to decide in which direction to go next
- Determine success or failure of reaching the goal point (2,2)
- Plot coordinates on the screen
- Use the color LED to report success or failure
- Repeat the experiment multiple times and determine how often Rover reaches the goal
- Examine the theoretical probability of reaching the goal

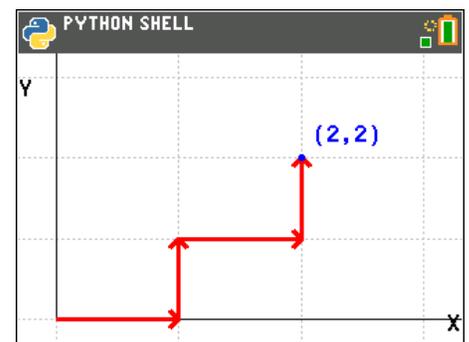
**Teacher Tip:** The plotting statements can be omitted for a shorter program but the `tiplotlib` is imported for just this purpose.

Rover starts at the origin (O). At each grid point, Rover can move only east (to the right) or north (up) at random, one unit at a time. There are several different routes Rover can take to reach the goal (2, 2). One route is highlighted in red/bold in the image to the right. In how many *different* ways can Rover get to the goal? If each move is random it is possible that Rover will miss the goal.

*What is the probability that Rover makes it to the goal?*

Write a *simulation* of this problem to keep track of the number of times Rover makes it to the goal and determine the percentage of the trials that are successful.

*Think about the failures: How do you know that Rover fails to make it to the goal?*



**Teacher Tip:** When  $x=3$  or  $y=3$ , we can stop because Rover lost the chance to get to the point (2,2) since Rover can only move up or right.

If there is not enough room to drive (you need about 0.5m x 0.5m space), set Rover's unit to a smaller value using `rv.grid_m_unit(scale_value)`. The default is 0.1 meters.



1. Begin a new Python Rover Coding project and add

**import ti\_plotlib as plt**

Since this is a 'random' walk also add the random module:

**from random import \***

Set up a **plt** screen to display Rover's positions along the route using the **ti\_plotlib Setup** menu:

```
plt.cls()
plt.window(-1, 4, -1, 4)
plt.grid(1, 1, "dash")
plt.axes("on")
```

```
EDITOR: RVCOORDD
PROGRAM LINE 0007
# Rover
from time import *
from ti_system import *
import ti_rover as rv
import ti_plotlib as plt
from random import *

plt.cls()
plt.grid(1,1,"dash")
plt.window(-1,4,-1,4)
plt.axes("on")
```

2. Use a variable for the number of trials: **tr = 10**

Use a variable to count the successes: **su = 0**

Use *two* variables for the goal point: **px, py = 2, 2** (Yes, this is valid!)

Use a **for** loop to perform the trials: **for i in range(tr):**

Now begin a trial:

*In the loop block, use two different variables for Rover's position:*

◆ ◆ **rx, ry = 0, 0**

```
EDITOR: RVCOORDD
PROGRAM LINE 0008
plt.cls()
plt.grid(1,1,"dash")
plt.window(-1,4,-1,4)
plt.axes("on")

tr=10
su=0
px,py=2,2
for i in range(tr):
  rx,ry=0,0
  ..
```

**Teacher Tip:** In Python **a, b = c, d** means: **a = c** and **b = d**.

Another valid Python expression is **a=b=0** which sets both **a** and **b** to 0 (or something else).

3. Plot this initial point on the graphing screen:

◆ ◆ **plt.plot(rx, ry, "o")**

```
EDITOR: RVCOORDD
PROGRAM LINE 0009
plt.grid(1,1,"dash")
plt.window(-1,4,-1,4)
plt.axes("on")

tr=10
su=0
px,py=2,2
for i in range(tr):
  rx,ry=0,0
  plt.plot(rx,ry,"o")
  ..
```

**Teacher Tip:** Rover knows where it is on the grid. **rv.waypoint\_x()** and **rv.waypoint\_y()** tell the current position but it's better for students to maintain control of the variables.



# 10 Minutes of Code – Python

## TI-84 PLUS CE PYTHON WITH THE TI-INNOVATOR™ ROVER

### UNIT 6: APPLICATION

### TEACHER NOTES

- Make a **while** loop that continues as long as Rover is *not* at the goal point **and** Rover has not failed. Think about what condition determines a failure.

This additional condition is left as an exercise.

Find **!=** ('does not equal') and **and** on the [test] key ([2<sup>nd</sup>] [math])

```

EDITOR: RVCOORDD
PROGRAM LINE 0019
plt.window(-1,4,-1,4)
plt.axes("on")

tr=10
su=0
px,py=2,2
for i in range(tr):
    rx,ry=0,0
    plt.plot(rx,ry,"o")
    while (rx,ry)!= (px,py) and ?:_
    ****
  
```

**Teacher Tip:** Failure happens when Rover reaches a point where either the x-coordinate or the y-coordinate is 3 since Rover is 'out of the box'. The **while** loop should be:

**while (rx, ry) != (px, py) and rx < 3 and ry < 3:**

- In the **while** block, use **randint(0, 1)** to decide whether to go east (0 degrees) or north (90 degrees). This is accomplished with the statement:

◆◆◆◆ **dir = randint(0, 1) \* 90**

Recall that **randint( , )** is found in the **random...** module.

Notice the extra indentation. This statement is in the **while** block which is in the **for** block.

```

EDITOR: RVCOORDD
PROGRAM LINE 0021
plt.axes("on")

tr=10
su=0
px,py=2,2
for i in range(tr):
    rx,ry=0,0
    plt.plot(rx,ry,"o")
    while (rx,ry)!= (px,py) and ?:_
        dir=randint(0,1)*90
    ****
  
```

- Get Rover to turn to the correct *angle* (**dir**) and move *forward* 1 unit. (The statements in the image are incomplete)

Add an **rv.wait\_until\_done()** statement to control the plotting speed.

```

EDITOR: RVCOORDD
PROGRAM LINE 0023
tr=10
su=0
px,py=2,2
for i in range(tr):
    rx,ry=0,0
    plt.plot(rx,ry,"o")
    while (rx,ry)!= (px,py) and ?:_
        dir=randint(0,1)*90
        rv.to_angle(dir)
        rv.forward(1)
        rv.wait_until_done()
    ****
  
```

**Teacher Tip:** **rv.left()** and **rv.right()** can be used as well but it is more complicated. When Rover turns to an *angle*, it always rotates counter-clockwise (even 360 degrees when it's already heading in the correct direction).



- Update Rover's position variables, **rx** and **ry**.

If Rover moved east (**dir == 0**),  
     add 1 to **rx**.

Otherwise,  
     add 1 to **ry**.

Plot Rover's position in the screen using **plt.plot(rx,ry,"o")** using proper indentation!

- After the **while** loop ends (notice the indentation again), determine if Rover was successful:

**if (rx, ry) == (px, py):**

**else:**

and count the success by adding 1 to the variable **su**.

Print "success" or "fail".

If Rover is successful, light the color LED in **GREEN**, otherwise light the color LED in **RED** or use the colors of your choice.

Then enter (not shown):

- two statements for Rover to return to (0,0) and face east (to angle 0 degrees)
- a statement to turn the LED off
- statements to redraw the plotting screen starting with **plt.cls()**, and then use the three **Setup** statements again

- After the **for** loop ends (when all the trials are finished), print the results of the experiments:

- the total number of successes
- the percentage of the successes (successes / trials \*100)

A sample run of 10 trials is shown in the image to the right. The percentage is the **experimental probability** of success.

```

EDITOR: RVCOORDD
PROGRAM LINE 0027
3 and ry<3:
....dir=randint(0,1)*90
....rv.to_angle(dir)
....rv.forward(1)
....rv.wait_until_done()
....if dir==0:
....
....else:
....
....plt.plot(rx,ry,"o")

```

```

EDITOR: RVCOORDD
PROGRAM LINE 0036
....else:
....
....plt.plot(rx,ry,"o")
....if (rx,ry)==(px,py):
....su=su+1
....print("success")
....rv.color_rgb(0,255,0)
....else:
....print("fail")
....rv.color_rgb(255,0,0)

```

```

PYTHON SHELL
fail
fail
fail
success
success
fail
success
success
successes= 4
percentage= 40.0
>>> |

```

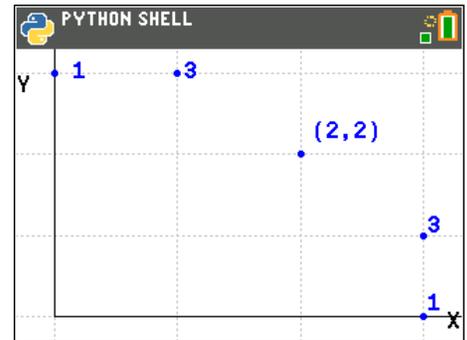


#### 10. Theoretical Probability

This image shows the number of routes to each end point of the trips.

The probability of success is  $3/8$  or  $.375$ .

How close did your experiments come? What should you do to get a more accurate **experimental probability**?



**Teacher Tip:** Use a probability tree diagram to determine the theoretical probability. There are 6 paths to success, each taking four steps. Each step has probability  $\frac{1}{2}$  for success, so a successful route happens with probability  $(1/2)^4$ .  $P(\text{success}) = 6 \cdot (1/2)^4 = 3/8 = .375$

Sample final code:

Tip: Comment all the **rv.** statements and try 100 trials.

```
# Unit 6 Application: random walk
#=====
from time import *
from ti_system import *
import ti_rover as rv
import ti_plotlib as plt
from random import *
#=====
trials = int(input("# of trials? "))
successes = 0
px, py = 2, 2
for i in range(trials):
    plt.cls()
    plt.window(-1,4,-1,4)
    plt.grid(1,1,"dashed")
    plt.axes("on")
    rx, ry = 0, 0
    plt.plot(rx,ry,"o")
    rv.color_rgb(0,0,0)
    while (rx,ry) != (px,py) and rx<3 and ry<3:
        dir=randint(0, 1) * 90
        rv.to_angle(dir)
        rv.forward(1)
        plt.plot(rx,ry,"o")
        rv.wait_until_done()
        if dir==0:
            rx+=1
        else:
            ry+=1
        plt.plot(rx,ry,"o")
    # end of while loop
    if (rx,ry) == (px,py):
        successes+=1
        print("SUCCESS!")
        rv.color_rgb(0,255,0)
    else:
        print("FAIL")
        rv.color_rgb(255,0,0)
    rv.to_xy(0,0)
    rv.to_angle(0)
    rv.wait_until_done()
# end of for loop
print("Successes:", successes)
print("Percentage: ", successes/trials*100)
```



```
PYTHON SHELL
successes= 37
percentage= 37.0
>>> |
Fns... a A # Tools Editor Files
```

Sample run with 100 trials