<u>**Final Project Option C: Target-Finding in 2D Cells**</u>

Individual Questions - Maya:
[1.] Programming: In this cluster, you have seen how DNA is used to create proteins which run all the processes in cells. Some would say that DNA is the software of life. Based on your knowledge of programming, evaluate that statement. In what sense is DNA like a software program? In what sense is it very different?

The main function of DNA itself is to provide instructions to help organisms function by carrying out functions or tasks. This makes DNA similar to a software program because software programs also execute or carry out functions or tasks to often accomplish one goal. Additionally, DNA needs to be converted into messages that help produce proteins. This also makes DNA similar to a software program because processing systems convert our code/software program into something that could be understood and read by our computers. DNA is different from a software program because the function of DNA is a lot more complex than simply providing instructions with everything that goes on in the body of organisms.

[2.] Biophysics: Explain, in what sense, is the following statement true: 'Everything that animals do, atoms do.'

The statement is true in the sense that atoms are the basic building block of matter and all animals are made up of matter. Therefore, animals are made up of atoms. Since animals and atoms are essentially made up of the same thing, they would carry out the same functions. More specifically, animals would carry out all the functions of atoms because they are made of atoms.

Individual Questions - Serena:
[1.] Programming: In this cluster, you have seen how DNA is used to create proteins which run all the processes in cells. Some would say that DNA is the software of life. Based on your knowledge of programming, evaluate that statement. In what sense is DNA like a software program? In what sense is it very different?

DNA resembles a software program in that it's a code that, when executed, produces one intended result. Programming gives the computer instructions on how to perform a task, DNA gives the cell instructions on how to produce proteins. However, there are definitely areas where DNA is different to software. The only thing DNA does is provide a template for protein production. It doesn't dictate where those proteins go, how they are used, or how they assemble whatever organism the DNA belongs to. Where a program should produce a complete result, DNA is more like the ingredients list to a cake, or header file to a program.

[2.] Biophysics: Explain, in what sense, is the following statement true: 'Everything that animals do, atoms do.'
The atomic theory states that everything is made of atoms. The basic building blocks of all animals are atoms, and are bound by the same laws of physics. Because animals act, and animals are a group of atoms, then the actions of the animal are also the actions of the atoms. If $a = b$ and $b = c$ then $a = c$.

Constants

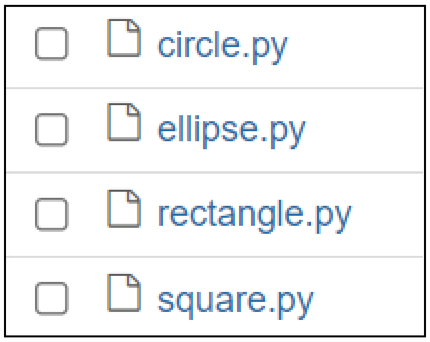
- Area: 1600 $units^2$
- Target:
    - Area: 16 $units^2$
    - Dimensions: 4 $units$ × 4 $units$
    - Position coordinates: (10,10)

```
target       = setup_rectangle((10.0, 10.0), 4, 4)
```

- Random walker starting coordinates: (0,0)

```
start_loc    = ( 0.0, 0.0)
```

Single Changing Parameter: Shape of Cell



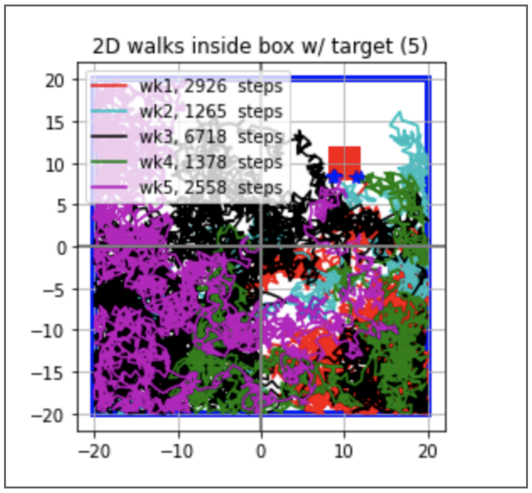
circle.py
ellipse.py
rectangle.py
square.py

Square Cell

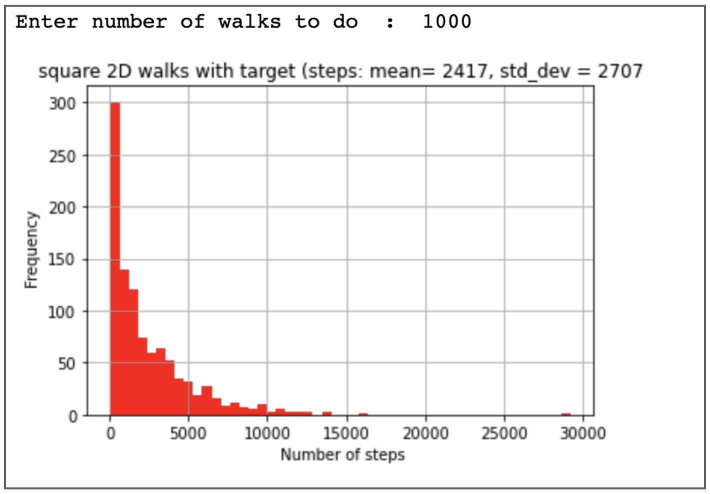
- Dimensions:
  - Length: 40 units
  - Width: 40 units
- Center coordinate: (0,0)

```
boundary     = setup_square((0.0, 0.0), 40)
```

- Testing 1-5 Random Walks on Graph:



- Histogram (shows the frequency distribution of the number of steps for each random walk):



```
Enter number of walks to do  :  1000
```

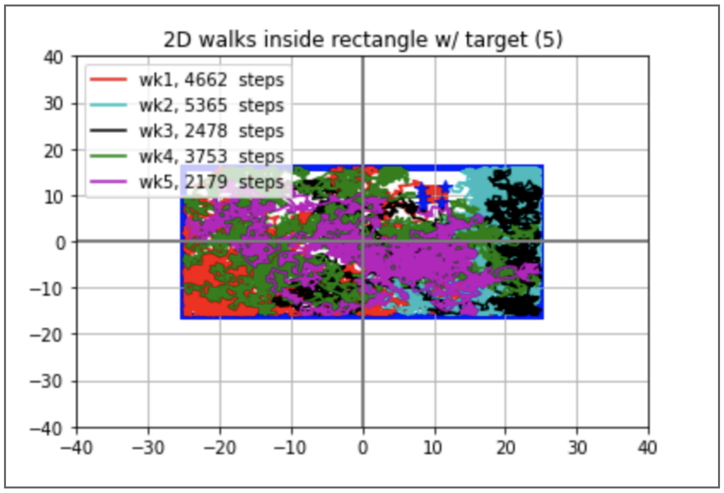
  - Average steps to reach target in square: 2417 steps

Rectangle Cell

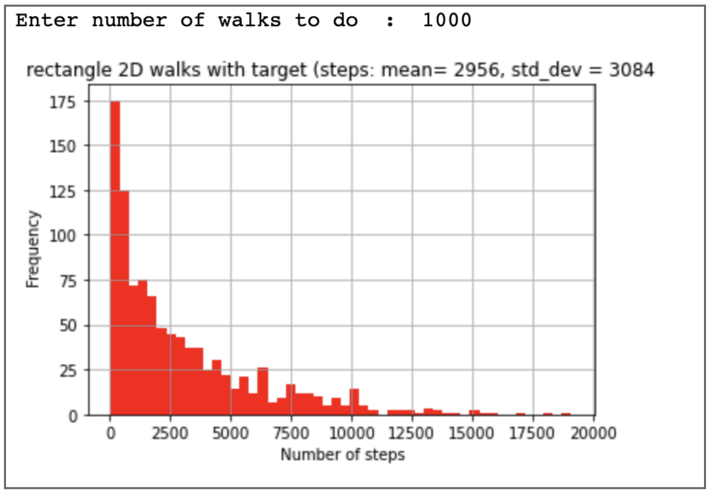
- Dimensions:
    - Length: 50 units (y-axis)
    - Width: 32 units (x-axis)
- Center coordinate: (0,0)

```
boundary     = setup_rectangle((0.0, 0.0), 50, 32)
```

- Testing 1-5 Random Walks on Graph:



- Histogram (shows the frequency distribution of the number of steps for each random walk):



```
Enter number of walks to do  :  1000
```

    - Average steps to reach target in rectangle: 2956 steps

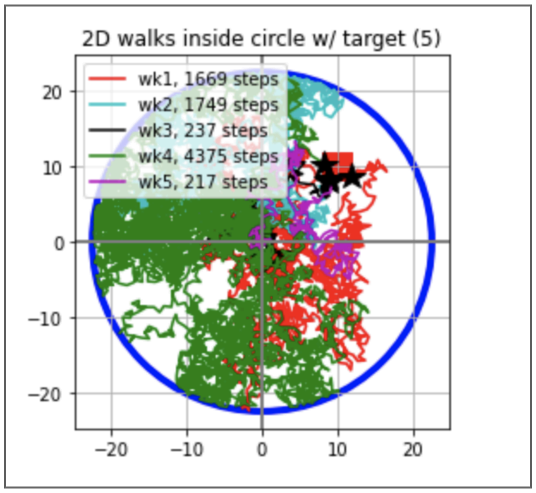
Circle Cell

- Dimensions:
  - Radius: $\sqrt{1600 \div \pi}$ units
    (Area of Circle - 1600 $= \pi r^2$, so $r = \sqrt{1600 \div \pi}$)
- Center coordinate: (0,0)

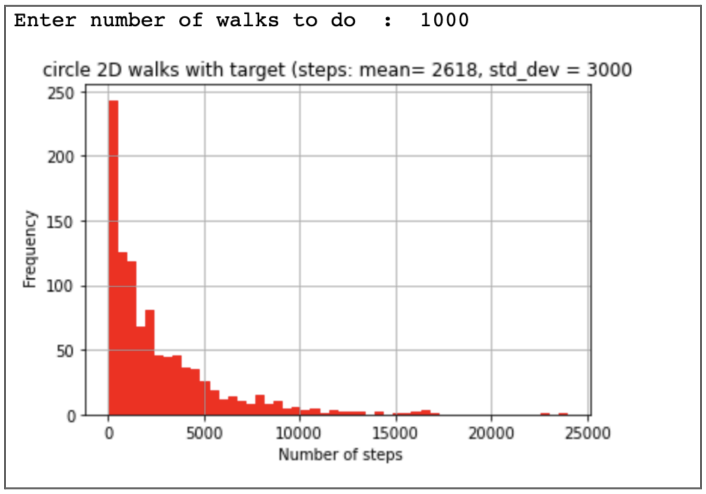
```
circle = setup_circle((0.0, 0.0), np.sqrt(1600/(np.pi)))
```

- Testing 1-5 Random Walks on Graph:

.



- Histogram (shows the frequency distribution of the number of steps for each random walk):



```
Enter number of walks to do  :  1000
```

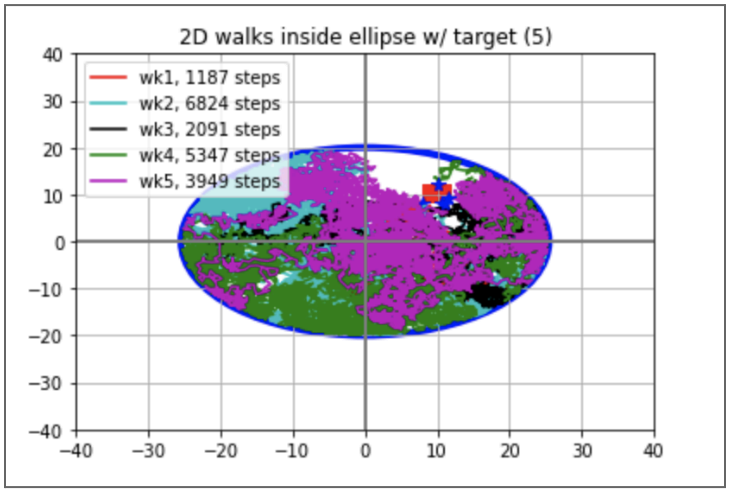
  - Average steps to reach target in circle: 2618 steps

Ellipse Cell

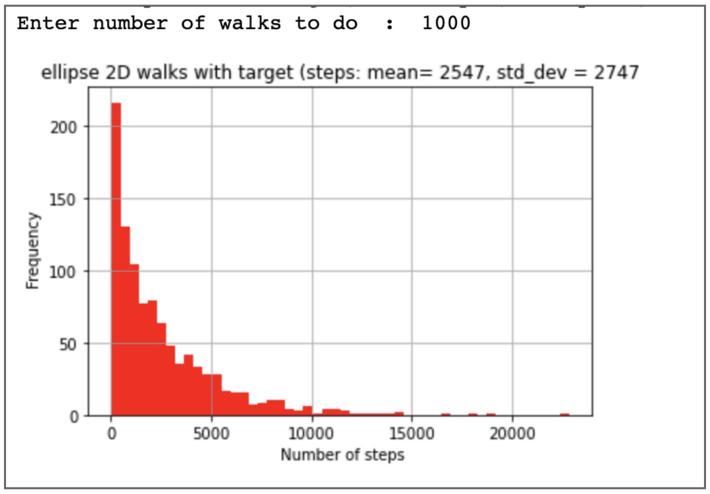
- Dimensions:
  - Major Axis: $2(\frac{1600}{20\pi})$
  - Minor Axis: 40
  - Area of ellipse = 1600 ($A = \pi ab$ where $a = \frac{major\ axis}{2}$ and $b = \frac{minor\ axis}{2}$)
- Center coordinate: (0,0)

```
boundary    = setup_ellipse((0.0, 0.0), (2*(1600/(20*np.pi))), 40)
```

- Testing 1-5 Random Walks on Graph:



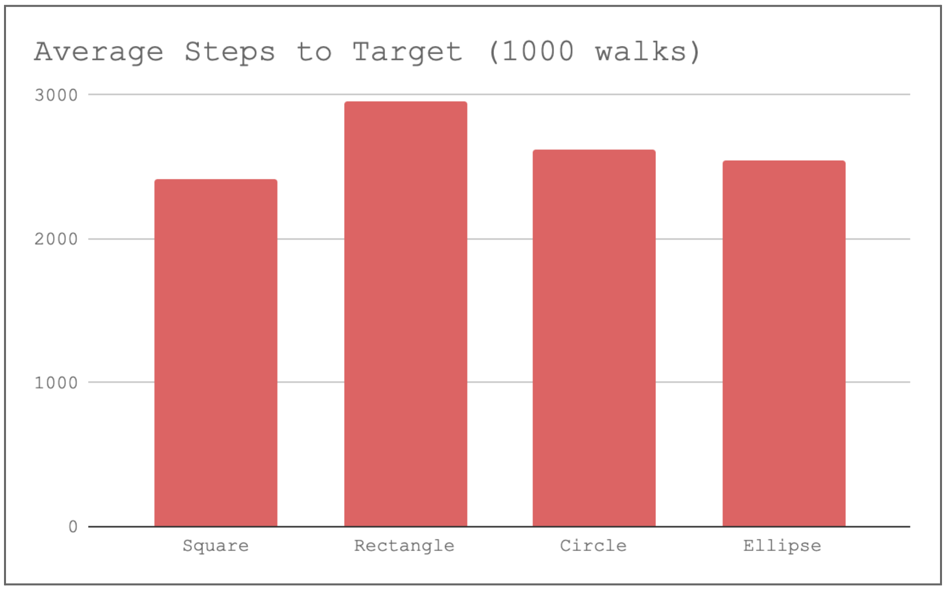
- Histogram (shows the frequency distribution of the number of steps for each random walk):



```
Enter number of walks to do  :  1000
```

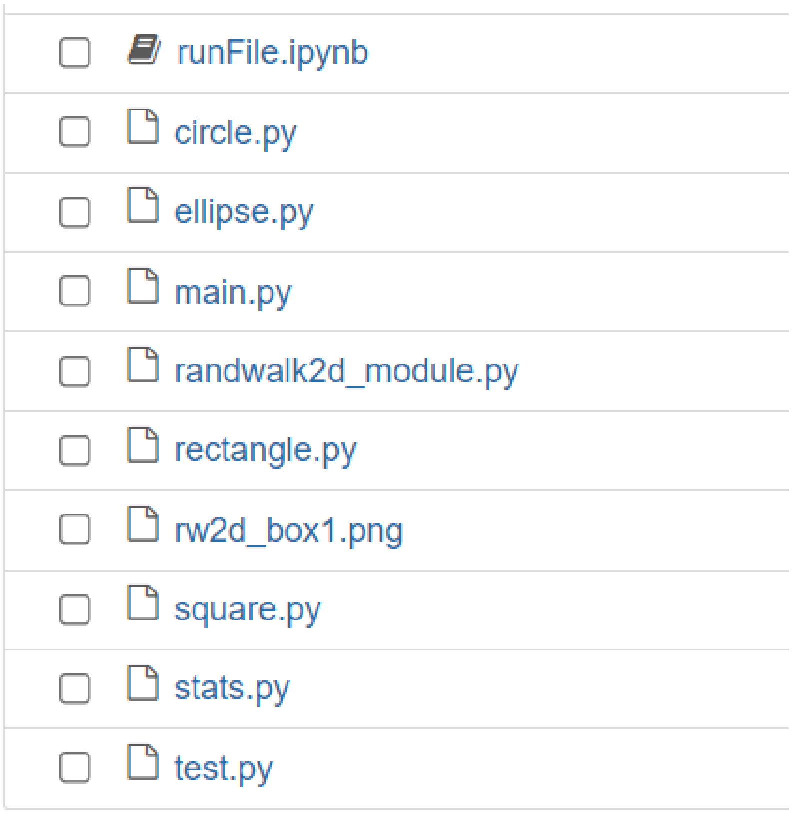
  - Average steps to reach target in ellipse: 2547 steps

Conclusion:



The average number of steps to the target for the square cell was 2417 steps, for the rectangle cell was 2956 steps, for the circle cell was 2618 steps, and for the ellipse cell was 2547 steps. The average number of steps for all of the cells was between 2000 and 3000. Although the square cell had the least number of average steps (2417) and the rectangle cell had the largest number of average steps (2956), there was no clear pattern between the four shapes we tested and the average steps. Overall, we can conclude that changing the cell shape with a constant volume and constant target location doesn't cause a significant difference in the average number of steps taken to the target.

Code from Jupyter Notebook:


runFile.ipynb
circle.py
ellipse.py
main.py
randwalk2d_module.py
rectangle.py
rw2d_box1.png
square.py
stats.py
test.py

Stats.py

```
import matplotlib.pyplot as plt
import numpy as np
from rectangle import random_walk_target_rectangle
from ellipse import random_walk_target_ellipse
from square import random_walk_target_square
from circle import random_walk_target_circle
from randwalk2d_module import setup_rectangle, setup_ellipse,
setup_square, setup_circle, draw_boundary_rectangle,
draw_boundary_ellipse, draw_boundary_circle, draw_target

def rectangle_stats(num_walks):
  max_steps = 50000
  rectangle = setup_rectangle((0.0, 0.0), 50, 32)
  target = setup_rectangle((10.0, 10.0), 4, 4)
  start_loc = ( 0.0, 0.0)
  plt.clf
  walk_list = []
  for i in range(num_walks):
      xwalk1,ywalk1,steps_taken,target_reached =
random_walk_target_rectangle(max_steps, start_loc, rectangle, target
)
      if target_reached :
          walk_list.append(steps_taken)
  walk_list = np.array(walk_list)
  ave_steps = int(np.mean(walk_list))
  std_dev_steps = int(np.std(walk_list))
  plt.hist(walk_list, bins=50, color='red')
  plt.title(f'rectangle 2D walks with target (steps: mean=
{ave_steps}, std_dev = {std_dev_steps}')
  plt.xlabel(' Number of steps')
  plt.ylabel(' Frequency')
  # plt.savefig("rw2d_target_stats.png")
  plt.grid(True)
  plt.show()

def ellipse_stats(num_walks):
  max_steps = 50000
  ellipse = setup_ellipse((0.0, 0.0), (2*(1600/(20*np.pi))), 40)
  target = setup_rectangle((10.0, 10.0), 4, 4)
  start_loc = ( 0.0, 0.0)
  plt.clf
  walk_list = []
  for i in range(num_walks):
      xwalk1,ywalk1,steps_taken,target_reached =
random_walk_target_ellipse(max_steps, start_loc, ellipse, target )
      if target_reached :
          walk_list.append(steps_taken)
  walk_list = np.array(walk_list)
  ave_steps = int(np.mean(walk_list))
  std_dev_steps = int(np.std(walk_list))
  plt.hist(walk_list, bins=50, color='red')
  plt.title(f'ellipse 2D walks with target (steps: mean= {ave_steps},
std_dev = {std_dev_steps}')
  plt.xlabel(' Number of steps')
  plt.ylabel(' Frequency')
  # plt.savefig("rw2d_target_stats.png")
  plt.grid(True)
  plt.show()

def square_stats(num_walks):
  max_steps = 50000
  square = setup_square((0.0, 0.0), 40)
  target = setup_rectangle((10.0, 10.0), 4, 4)
  start_loc = ( 0.0, 0.0)
  plt.clf
  walk_list = []
  for i in range(num_walks):
      xwalk1,ywalk1,steps_taken,target_reached =
random_walk_target_square(max_steps, start_loc, square, target )
      if target_reached :
          walk_list.append(steps_taken)
  walk_list = np.array(walk_list)
  ave_steps = int(np.mean(walk_list))
  std_dev_steps = int(np.std(walk_list))
  plt.hist(walk_list, bins=50, color='red')
  plt.title(f'square 2D walks with target (steps: mean= {ave_steps},
std_dev = {std_dev_steps}')
  plt.xlabel(' Number of steps')
  plt.ylabel(' Frequency')
  # plt.savefig("rw2d_target_stats.png")
  plt.grid(True)
  plt.show()

def circle_stats(num_walks):
  max_steps = 50000
  circle = setup_circle((0.0, 0.0), np.sqrt(1600/(np.pi)))
  target = setup_rectangle((10.0, 10.0), 4, 4)
  start_loc   = (0.0, 0.0)
  plt.clf
  walk_list = []
  for i in range(num_walks):
      xwalk1,ywalk1,steps_taken,target_reached =
random_walk_target_circle(max_steps, start_loc, circle, target )
      if target_reached :
          walk_list.append(steps_taken)
  walk_list = np.array(walk_list)
  ave_steps = int(np.mean(walk_list))
  std_dev_steps = int(np.std(walk_list))
  plt.hist(walk_list, bins=50, color='red')
  plt.title(f'circle 2D walks with target (steps: mean= {ave_steps},
std_dev = {std_dev_steps}')
  plt.xlabel(' Number of steps')
  plt.ylabel(' Frequency')
  # plt.savefig("rw2d_target_stats.png")
  plt.grid(True)
  plt.show()
```

Test.py

```
import matplotlib.pyplot as plt
import numpy as np
from rectangle import random_walk_target_rectangle
from ellipse import random_walk_target_ellipse
from square import random_walk_target_square
from circle import random_walk_target_circle
from randwalk2d_module import setup_rectangle, setup_ellipse,
setup_square, setup_circle, draw_boundary_rectangle,
draw_boundary_ellipse, draw_boundary_circle, draw_target

max_steps = 30000

area = 1600

plot_size = 40

def rectangle_test(num_walks):
  boundary    = setup_rectangle((0.0, 0.0), 50, 32)
  target      = setup_rectangle((10.0, 10.0), 4, 4)
  start_loc   = ( 0.0, 0.0)
  line_type = ['r-','c-','k-','g-','m-']
  for i in range(num_walks):
    xwalk1,ywalk1,steps_taken,target_reached =
random_walk_target_rectangle(max_steps, \
    start_loc, boundary, target )
    plt.plot(xwalk1, ywalk1, line_type[i],label= f'wk{i+1},
{steps_taken}  steps')
    plt.plot(xwalk1[-1],ywalk1[-1], 'b*', ms=8)
  draw_boundary_rectangle(start_loc, boundary, plot_size)
  draw_target(target)
  plt.title(f'2D walks inside rectangle w/ target ({num_walks})' )
  plt.legend(loc="upper left")
  plt.grid(True)
  plt.savefig("rw2d_target1.png")
  plt.show()

def ellipse_test(num_walks):
  boundary    = setup_ellipse((0.0, 0.0), (2*(1600/(20*np.pi))), 40)
  target      = setup_rectangle((10.0, 10.0), 4, 4)
  start_loc   = (0.0, 0.0)
  line_type = ['r-','c-','k-','g-','m-']
  for i in range(num_walks):
    xwalk1,ywalk1,steps_taken,target_reached =
random_walk_target_ellipse(max_steps, \
            start_loc, boundary, target )
    plt.plot(xwalk1, ywalk1, line_type[i],label= f'wk{i+1},
{steps_taken} steps')
    plt.plot(xwalk1[-1],ywalk1[-1], 'b*', ms=8)
  draw_boundary_ellipse(start_loc, boundary, plot_size)
  draw_target(target)
  plt.title(f'2D walks inside ellipse w/ target ({num_walks})' )
  plt.legend(loc="upper left")
  plt.grid(True)
  plt.savefig("rw2d_target1.png")
  plt.show()

def square_test(num_walks):
  boundary    = setup_square((0.0, 0.0), 40)
  target      = setup_rectangle((10.0, 10.0), 4, 4)
  start_loc   = ( 0.0, 0.0)
  line_type = ['r-','c-','k-','g-','m-']
  for i in range(num_walks):
    xwalk1,ywalk1,steps_taken,target_reached =
random_walk_target_square(max_steps, \
    start_loc, boundary, target )
    plt.plot(xwalk1, ywalk1, line_type[i],label= f'wk{i+1},
{steps_taken}  steps')
    plt.plot(xwalk1[-1],ywalk1[-1], 'b*', ms=8)
  draw_boundary_rectangle(start_loc, boundary, plot_size)
  draw_target(target)
  plt.axis('square')
  plt.title(f'2D walks inside box w/ target ({num_walks})' )
  plt.legend(loc="upper left")
  plt.grid(True)
  plt.savefig("rw2d_target1.png")
  plt.show()

def circle_test(num_walks):
  circle = setup_circle((0.0, 0.0), np.sqrt(1600/(np.pi)))
  target = setup_rectangle((10.0, 10.0), 4, 4)
  start_loc   = (0.0, 0.0)
  line_type = ['r-','c-','k-','g-','m-']
  plt.clf
  for i in range(num_walks):
    xwalk1,ywalk1,steps_taken,target_reached =
random_walk_target_circle(max_steps, start_loc, circle, target )
    plt.plot(xwalk1, ywalk1, line_type[i],label= f'wk{i+1},
{steps_taken} steps')
    plt.plot(xwalk1[-1],ywalk1[-1], 'k*', ms=16)
  draw_boundary_circle(start_loc, circle, plot_size)
  draw_target(target)
  plt.axis('square')
  plt.title(f'2D walks inside circle w/ target ({num_walks})' )
  plt.legend(loc="upper left")
  plt.grid(True)
  plt.savefig("rw2d_box1.png")
  plt.show()
```

Randwalk2d_module

```
import random
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle, Ellipse, Circle
random.seed(None)

def rand_grid():
    xdelta, ydelta = random.choice( [(1,0),(0,1),(-1,0),(0,-1)] )
    return xdelta, ydelta

def rand_angle():
    """
    Performs unit 2D random step at random angle ( 0 to 2*PI)
    Returns (x,y) co-ordinates of unit step at random angle
     """
    rand_angle = random.uniform(0,2*np.pi)
    xdelta = np.sin(rand_angle)
    ydelta = np.cos(rand_angle)
    return xdelta, ydelta

def setup_rectangle( box_loc , width, height):
    """
    Sets shape of rectangle. Returns upper and lower corners
    """
    xloc,yloc = box_loc
    low_left  = (xloc - .5 * width, yloc -.5 * height)
    top_right = (xloc + .5 * width, yloc +.5 * height)
    return (low_left, top_right)

def setup_ellipse( ellipse_loc , axis_one, axis_two):
    """
    Sets shape of rectangle. Returns upper and lower corners
    """
    xloc,yloc = ellipse_loc
    axis = (axis_one, axis_two)
    return (ellipse_loc, axis)

def setup_square( square_loc, side):
    xloc, yloc = square_loc
    low_left  = (xloc - .5 * side, yloc -.5 * side)
    top_right = (xloc + .5 * side, yloc +.5 * side)
    return (low_left, top_right)

def setup_circle( circle_loc, radius):
    xloc,yloc = circle_loc
    circle_radius = radius
    return (circle_loc, circle_radius)

def check_inside_rectangle(xvalue, yvalue, box):
    """
    Checks if point is inside a rectangle. Returns True if inside.
    False if not
    """
    xmin, ymin = box[0]
    xmax, ymax = box[1]
    if xmin < xvalue < xmax  and  ymin < yvalue < ymax:
        return True
    else:
        return False

def check_inside_ellipse(xvalue, yvalue, ellipse):
    """
    Checks if point is inside a rectangle. Returns True if inside.
    False if not
    """
    xloc, yloc = ellipse[0]
    axis_one, axis_two = ellipse[1]
    half_axis_one = axis_one/2
    half_axis_two = axis_two/2
    if ((xvalue - xloc) ** 2/half_axis_one**2 + (yvalue - yloc) **
2/half_axis_two**2) < 1:
        return True
    else:
        return False

def check_inside_circle(xvalue, yvalue, circle):
    xloc, yloc = circle[0]
    radius = circle[1]
    if (np.sqrt(((xvalue-xloc)**2) + ((yvalue-yloc)**2))) < radius:
        return True
    else:
        return False

def draw_boundary_rectangle(start_loc,boundary, graph):
    """
    Draw boundary on graph and set size of graph (xlim,ylim)
    """
    xmin, ymin = boundary[0]
    xmax, ymax = boundary[1]
    plt.gca().add_patch(Rectangle((xmin,ymin), xmax-xmin, ymax-ymin,
fill=False, edgecolor='b',lw=4))
    plt.xlim([-graph,graph])
    plt.ylim([-graph,graph])
    plt.axvline(start_loc[0], color='grey', lw=2)
    plt.axhline(start_loc[1], color='grey', lw=2)
    return

def draw_boundary_ellipse(start_loc,boundary, graph):
    """
    Draw boundary on graph and set size of graph (xlim,ylim)
    """
    xloc, yloc = boundary[0]
    axis_one, axis_two = boundary[1]
    plt.gca().add_patch(Ellipse((xloc, yloc), axis_one, axis_two,
fill=False, edgecolor='b',lw=4))
    plt.xlim([-graph,graph])
    plt.ylim([-graph,graph])
    plt.axvline(start_loc[0], color='grey', lw=2)
    plt.axhline(start_loc[1], color='grey', lw=2)
    return

def draw_boundary_circle(start_loc, circle, graph):
    """
    Draw circle on graph and set size of graph (xlim,ylim)
    """
    xloc, yloc = circle[0]
    radius = circle[1]
    plt.gca().add_patch(Circle((xloc,yloc), radius,  fill=False,
edgecolor='b',lw=4))
    plt.xlim([-graph, graph])
    plt.ylim([-graph, graph])
    plt.axvline(start_loc[0], color='grey', lw=2)
    plt.axhline(start_loc[1], color='grey', lw=2)
    return
```

```
def draw_target(target):
    """
    Draw target on graph
    """
    xmin, ymin = target[0]
    xmax, ymax = target[1]
    plt.gca().add_patch(Rectangle((xmin,ymin), xmax-xmin, ymax-ymin,
fill=True, facecolor='r'))
```