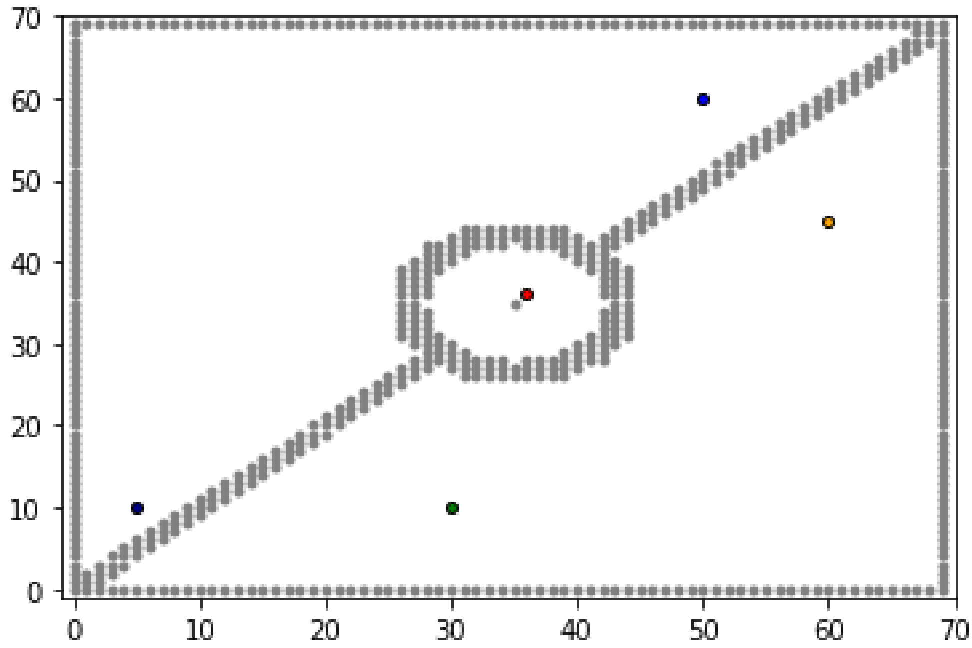
exercise-iii

March 20, 2018

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

n = 70

xc = n/2
yc = n/2
```

In [2]:
```
def is_exit(position): # test if position is an absorption site (set A union B)
    i, j = position
    return (i == 0 or j == 0 or i == n-1 or j == n-1
            or (j == i-1 or j == i or j == i+1) and np.linalg.norm([i-xc, j-yc]) >= 10
            or 7 < np.linalg.norm([i-xc, j-yc]) < 10
            or is_good_exit(position))

def is_good_exit(position): # test if position is a good absorption site (the set A)
    i, j = position
    return i == xc and j == yc

# The various special points in the maze.
red_point = (xc+1, yc+1)
blue_point = (50, 60)
orange_point = (60, 45)
green_point = (30, 10)
darkblue_point = (5, 10)

plt.plot(red_point[0], red_point[1], color='red', marker='o', ms=4, mec='black')
plt.plot(blue_point[0], blue_point[1], color='blue', marker='o', ms=4, mec='black')
plt.plot(orange_point[0], orange_point[1], color='orange', marker='o', ms=4, mec='blac
plt.plot(green_point[0], green_point[1], color='green', marker='o', ms=4, mec='black')
plt.plot(darkblue_point[0], darkblue_point[1], color='darkblue', marker='o', ms=4, mec=

plt.xlim(-1,n)
plt.ylim(-1,n)
plt.scatter([i for i in range(n) for j in range(n) if is_exit( (i,j) )],
            [j for i in range(n) for j in range(n) if is_exit( (i,j) )],
            color='grey', marker='.'
```

```
            )
plt.show()
```




```
In [3]: P = np.zeros( (n,n,n,n) )  # transition probabilities of the unconditional chain
        for i in range(n):
            for j in range(n):
                if not(is_exit( (i,j) )):
                    for possible_move in [(i+1,j), (i+1, j+1), (i, j+1),
                                          (i-1,j+1), (i-1,j), (i-1,j-1),
                                          (i,j-1), (i+1,j-1)]:
                        xnew, ynew = possible_move
                        P[i,j,xnew,ynew] = 1
                    s = np.sum(P[i,j,:,:])
                    P[i,j,:,:] = (1.0 / s ) * P[i,j,:,:]
                else:
                    P[i,j,i,j] = 1.0

        print np.sum(P[blue_point[0], blue_point[1], :, :])

        P[blue_point[0], blue_point[1], :, :] = np.zeros_like(P[blue_point[0], blue_point[1],
        P[blue_point[0], blue_point[1], green_point[0], green_point[1]] = 1

        P[orange_point[0], orange_point[1], :, :] = np.zeros_like(P[orange_point[0], orange_po
        P[orange_point[0], orange_point[1], red_point[0], red_point[1]] = 1
```

```
1.0
```

```
In [4]: a = np.zeros( (n,n,n,n) ) # linear system, see the np.linalg.tensorsolve documentation
        for i in range(n):
            for j in range(n):
                if not(is_exit( (i,j) )):
                    a[i,j,:,:] = - P[i,j,:,:]

                a[i,j,i,j] = 1

        b = np.zeros( (n,n) ) # boundary conditions: 1 for good exists and 0 for others.
        for i in range(n):
            for j in range(n):
                if is_good_exit( (i,j) ):
                    b[i,j] = 1.0


        good_exit_probabilities = np.linalg.tensorsolve(a, b)
```

```
In [5]: P_transformed = np.zeros_like(P)
        for i in range(n):
            for j in range(n):
                if not(is_exit( (i,j) )):
                    for x in range(n):
                        for y in range(n):
                            P_transformed[i,j,x,y] = P[i,j,x,y] * good_exit_probabilities[x,y]
                else: # absorption once it reaches an exit
                    P_transformed[i,j,i,j] = 1.0
```

```
In [6]: class Point:
            def __init__(self, x, y):
                self.x = x
                self.y = y

        all_points = [Point(i,j) for i in range(n) for j in range(n)]

        # Run the walk and return the positions visited.
        def run_walk():
            x0 = darkblue_point[0]
            y0 = darkblue_point[1]
            current = Point(x0, y0)
            x_visited = []
            y_visited = []
            while(True):
                current = np.random.choice(
                    all_points,
                    1, # return one random element
                    p=np.array([P_transformed[current.x, current.y, p.x, p.y] for p in all_poi
                          )[0]
                x_visited.append(current.x)
```

```
        y_visited.append(current.y)
        if is_exit( (current.x, current.y) ):
            break

    return x_visited, y_visited
```

In [7]:
```
def run_and_plot_walk(color):
    x_visited, y_visited = run_walk()
    plt.plot(x_visited, y_visited, color=color)

run_and_plot_walk('green')
run_and_plot_walk('red')
run_and_plot_walk('blue')
run_and_plot_walk('cyan')
run_and_plot_walk('yellow')
plt.xlim(-1,n)
plt.ylim(-1,n)
plt.scatter([i for i in range(n) for j in range(n) if is_exit( (i,j) )],
            [j for i in range(n) for j in range(n) if is_exit( (i,j) )],
            color='grey', marker='.'
           )
plt.show()
```