Robot Localization with Particle Filtering

February 11, 2026

1 M5 Project

1.1 Justin Stutler

In this project you will solve a robot localization problem using the particle filtering technique. The problem is illustrated in the figures below. You will have two types of robots: the pacman (triangle) and the ghost (pentagon). The pacman can measure its distance to the landmarks (blue circles), but the ghost can only measure its distance to the pacman. The environment is cyclic, so when the pacman or the ghost cross one of the sides, it appears in the opposite side. You will have to code the downweigh and resample steps of the particle filtering code provided as starting point.

An initial version of the code with the problem specification (below) and a report template (at the bottom) are available in this notebook. Deliverables are the final code (non-functioning code is worth 0 points) and the report.

You must implement: - downweigh and resample steps for the pacman (20pts) - downweigh and resample steps for the ghost (20pts)

For your solution, describe the: - how you updated the ghost particles without having the real location of the pacman (20pts) - which other alternatives have you considered, and why you decided not to use them (10pts)

Run your code at least 10 times and: - compare the uncertainty in the location of the pacman and the ghost over time (15pts) - explain what causes the ghost particles to concentrate in a well-defined cluster (15pts) - see the rightmost picture above

2 Implementation

In this project, you can only modify the last cell of code. The area that can be modified is marked between the comments "YOUR CODE STARTS HERE" and "YOUR CODE ENDS HERE".

2.0.1 Install Packages

[ ]:
```
!pip install numpy
!pip install matplotlib
```

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

[2]:
```
# world configuration
landmarks  = [[20.0, 20.0], [80.0, 80.0], [20.0, 80.0], [80.0, 20.0]]
world_size = 100.0

# generic robot implementation
class robot:
    def __init__(self):
        # initialise robot with random location and orientation
        self.x = random.random() * world_size
        self.y = random.random() * world_size
        self.orientation = random.random() * 2.0 * math.pi

        self.forward_noise = 0.0
        self.turn_noise    = 0.0
        self.sense_noise   = 0.0

    def set(self, new_x, new_y, new_orientation):
        if new_x < 0 or new_x >= world_size:
            raise ValueError('X coordinate out of bound')
        if new_y < 0 or new_y >= world_size:
            raise ValueError('Y coordinate out of bound')
        if new_orientation < 0 or new_orientation >= 2 * math.pi:
            raise ValueError('Orientation must be in [0..2pi]')
        self.x = float(new_x)
        self.y = float(new_y)
        self.orientation = float(new_orientation)

    def set_noise(self, new_f_noise, new_t_noise, new_s_noise):
        self.forward_noise = float(new_f_noise);
        self.turn_noise    = float(new_t_noise);
        self.sense_noise   = float(new_s_noise);

    # apply noisy movement to robot
    def new_position(self, turn, forward):
        if forward < 0:
            raise ValueError('Robot cant move backwards')

        # turn, and add randomness to the turning command
        orientation = self.orientation + float(turn) + random.gauss(0.0, self.
 ↪turn_noise)
        orientation %= 2 * math.pi

        # move, and add randomness to the motion command
        dist = float(forward) + random.gauss(0.0, self.forward_noise)
        x = self.x + (math.cos(orientation) * dist)
        y = self.y + (math.sin(orientation) * dist)
        x %= world_size    # cyclic truncate
```

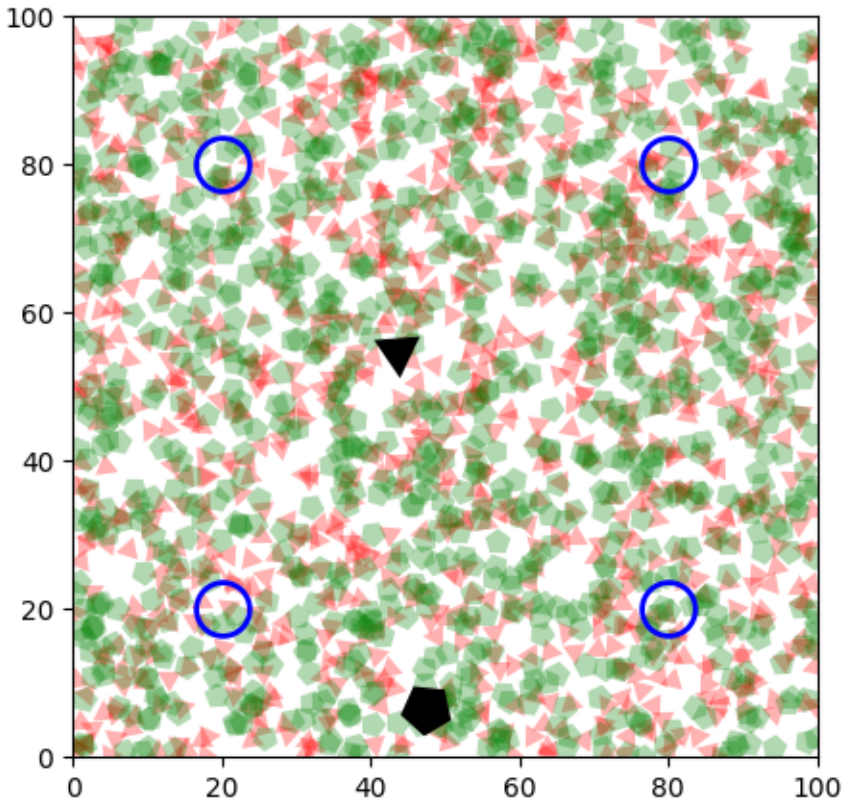
```
        y %= world_size

        return x, y, orientation

    # model noise using Gaussians
    def Gaussian(self, mu, sigma, x):
        # calculates the probability of x for 1-dim Gaussian with mean mu and␣
 ↪var. sigma
        return math.exp(- ((mu - x) ** 2) / (sigma ** 2) / 2.0) / math.sqrt(2.0␣
 ↪* math.pi * (sigma ** 2))

# PacMan class
class pacman(robot):
    # the PacMan can sense its location using the landmarks
    def sense(self):
        Z = []
        for i in range(len(landmarks)):
            dist = math.sqrt((self.x - landmarks[i][0]) ** 2 + (self.y -␣
 ↪landmarks[i][1]) ** 2)
            dist += random.gauss(0.0, self.sense_noise)
            Z.append(dist)
        return Z

    # calculates how likely a measurement should be
    def measurement_prob(self, measurement):
        prob = 1.0
        for i in range(len(landmarks)):
            dist = math.sqrt((self.x - landmarks[i][0]) ** 2 + (self.y -␣
 ↪landmarks[i][1]) ** 2)
            prob *= self.Gaussian(dist, self.sense_noise, measurement[i])
        return prob

    def move(self, turn, forward):
        x, y, orientation = self.new_position(turn, forward)
        res = pacman()
        res.set(x, y, orientation)
        res.set_noise(self.forward_noise, self.turn_noise, self.sense_noise)
        return res

# Ghost class
class ghost(robot):
    # the PacMan ("mypacman") can sense its distance to the ghost
    def sense(self, mypacman):
        Z = math.sqrt((self.x - mypacman.x) ** 2 + (self.y - mypacman.y) ** 2)␣
 ↪+ random.gauss(0.0, self.sense_noise)
        return Z
```

```
    # calculates how likely a measurement should be
    def measurement_prob(self, measurement, mypacman):
        dist = math.sqrt((self.x - mypacman.x) ** 2 + (self.y - mypacman.y) **␣
↪2)
        prob = self.Gaussian(dist, self.sense_noise, measurement)
        return prob

    def move(self, turn, forward):
        x, y, orientation = self.new_position(turn, forward)
        res = ghost()
        res.set(x, y, orientation)
        res.set_noise(self.forward_noise, self.turn_noise, self.sense_noise)
        return res
```

[3]:
```
# world visualization with the distribution of particles
def show_belief(mypacman, pacman_particles, myghost, ghost_particles):
    plt.rcParams["figure.figsize"] = (5,5)

    for p in pacman_particles:
        plt.plot(p.x, p.y, marker=(3, 0, 180.0*p.orientation/math.pi),␣
↪markerfacecolor='red', markersize=10, markeredgewidth=0.0, alpha=.3,␣
↪linestyle='None')

    for p in ghost_particles:
        plt.plot(p.x, p.y, marker=(5, 0, 180.0*p.orientation/math.pi),␣
↪markerfacecolor='green', markersize=10, markeredgewidth=0.0, alpha=.3,␣
↪linestyle='None')

    plt.plot(mypacman.x, mypacman.y, marker=(3, 0, 180.0*mypacman.orientation/
↪math.pi), markerfacecolor='black', markersize=20, markeredgewidth=0.0,␣
↪linestyle='None')
    ### CHANGE: print pacman and ghost after particles so they are not covered␣
↪up
    plt.plot(myghost.x, myghost.y, marker=(5, 0, 180.0*myghost.orientation/math.
↪pi), markerfacecolor='black', markersize=20, markeredgewidth=0.0,␣
↪linestyle='None')

    for x, y in landmarks:
          plt.plot(x, y, marker='o', markersize=20, markeredgewidth=2.0,␣
↪markerfacecolor='None', markeredgecolor='blue')

    plt.xlim([0,100])
    plt.ylim([0,100])

    plt.show()
```

```
# measure the proximity between the real location and the distribution of␣
 ↪particles
def eval(r, p):
    s = 0.0;
    for i in range(len(p)):
        dx = (p[i].x - r.x + (world_size/2.0)) % world_size - (world_size/2.0)
        dy = (p[i].y - r.y + (world_size/2.0)) % world_size - (world_size/2.0)
        err = math.sqrt(dx * dx + dy * dy)
        s += err
    return s / float(len(p))
```

[4]:
```
# pacman/ghost initialization
forward_noise = 3.0
turn_noise = 0.05
sense_noise = 3.0

mypacman = pacman()
mypacman.set_noise(forward_noise, turn_noise, sense_noise)

myghost = ghost()
myghost.set_noise(forward_noise, turn_noise, sense_noise)

# particle distribution
N = 1000 # number of particles
T = 10   # number of moves

# initialise randomly guessed particles for pacman and ghost
p = []
g = []
for i in range(N):
    x = pacman()
    x.set_noise(forward_noise, turn_noise, sense_noise)
    p.append(x)
    x = ghost()
    x.set_noise(forward_noise, turn_noise, sense_noise)
    g.append(x)

show_belief(mypacman, p, myghost, g)

print("Average pacman distance:", eval(mypacman, p))
print("Average ghost distance:", eval(myghost, g))
```




```
Average pacman distance: 38.04963943109306
Average ghost distance: 38.18960676562416
```

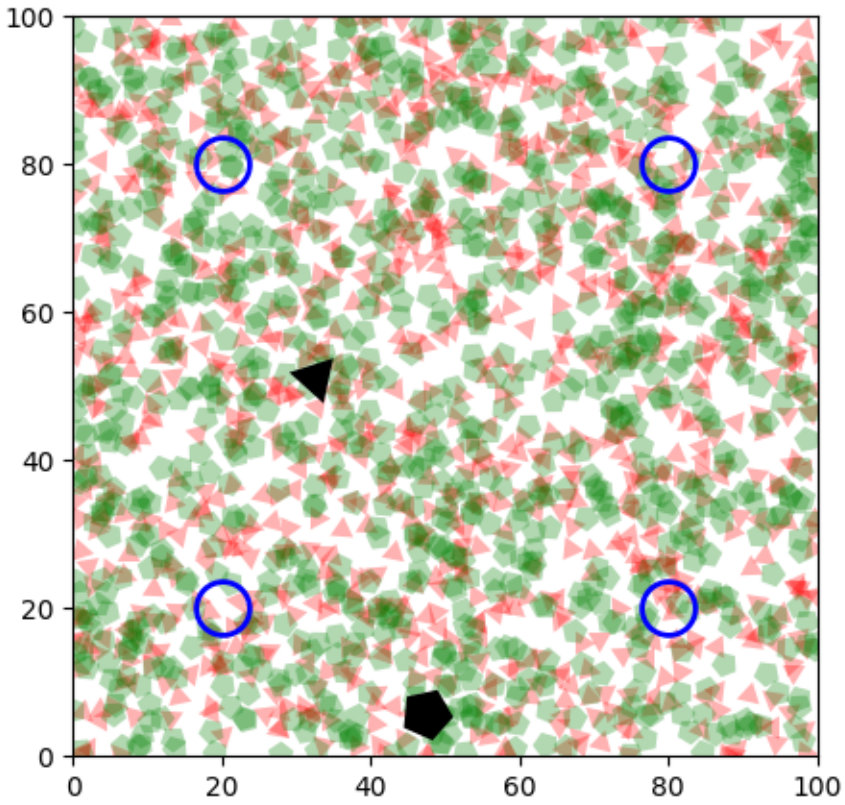
[5]:
```
# list to store distances
p_distances = []
g_distances = []
for turn in range(T):
    print('\nTurn #{}'.format(turn+1))

    # real pacman movement
    # turn 0.1 and move 10 meters
    mypacman = mypacman.move(0.2, 10.0)

    # real move for ghost (random)
    gturn = (random.random()-0.5)*(math.pi/2.0) # random angle in [-45,45]
    gdist = random.random()*20.0           # random distance in [0,20]
    myghost = myghost.move(gturn, gdist)

    # elapse time
    # move particles using the same movement made by robot
    p2 = []
```

```
    g2 = []
    for i in range(N):
        p2.append(p[i].move(0.2, 10.0))
        g2.append(g[i].move(gturn, gdist))
    p = p2
    g = g2

    show_belief(mypacman, p, myghost, g)
    print("Average pacman distance before resample:", eval(mypacman, p))
    print("Average ghost distance before resample:", eval(myghost, g))

    # observe
    ZP = mypacman.sense()        # noisy measurement of the distance between␣
↪the pacman and the landmarks
    ZG = myghost.sense(mypacman) # noisy measurement of the distance between␣
↪the pacman and the ghost

    ####################################
    ####################################
    ####################################
    ###### YOUR CODE STARTS HERE ######
    ####################################
    ####################################
    ####################################

    # downweight and resample pacman and ghost particles here - check the␣
↪Particle Filtering lecture for an example

    # particles do not know the real location of the pacman and the ghost, so␣
↪you cannot access "mypacman" and "myghost" in here
    # remember that you can still use p and g, which can be used to estimate␣
↪the location of the pacman and the ghost

    # you should use the obtained measurements from the sensors to update the␣
↪particles
    # - ZP contains the observed distances between the pacman and the landmarks
    # - ZG is the observed distance between the pacman and the ghost


    # PACMAN
    # weight particles
    pacman_w = []
    for pacman_rob in p:
        pacman_prob = pacman_rob.measurement_prob(ZP)
        pacman_w.append(pacman_prob)
```

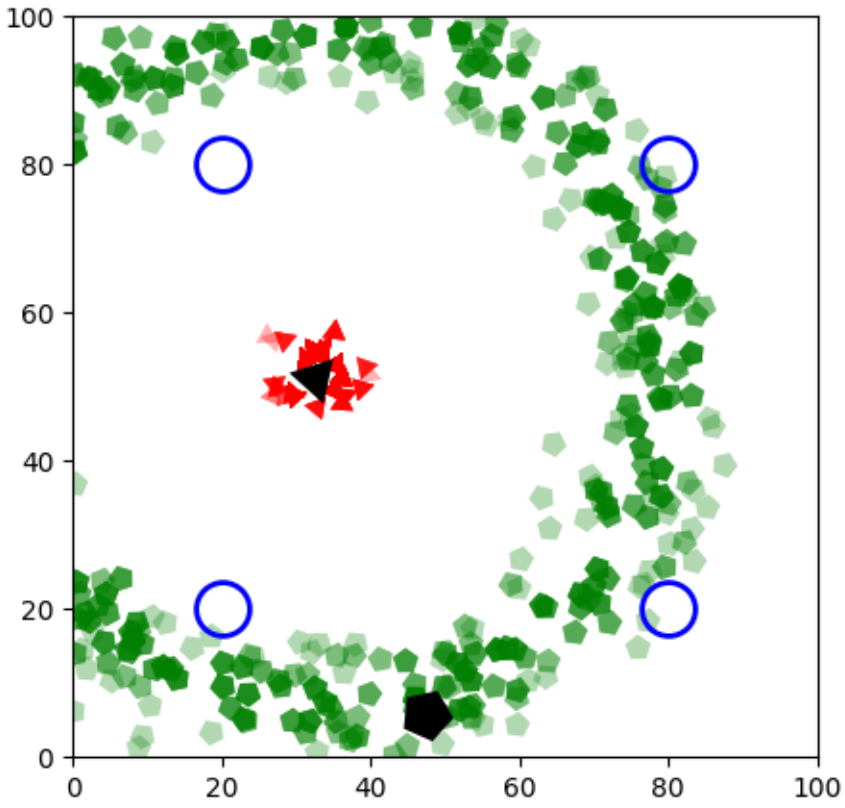
```
    # resample using spinning wheel
    p2 = []
    index = int(random.random()*N)
    beta = 0
    pacman_mw = max(pacman_w)

    for i in range(N):
        beta += random.random() * 2 * pacman_mw
        while beta > pacman_w[index]:
            beta -= pacman_w[index]
            index = (index + 1)%N
        p2.append(p[index])
    p = p2

    # GHOST
    # weight particles
    ghost_w = []
    for g_prob in g:
        prob_sum = 0
        for p_prob in p:
            prob_sum += g_prob.measurement_prob(ZG, p_prob)
        ghost_w.append(prob_sum)

    # resample using spinning wheel
    g2 = []
    index = int(random.random()*N)
    beta = 0
    ghost_mw = max(ghost_w)

    for i in range(N):
        beta += random.random() * 2 * ghost_mw
        while beta > ghost_w[index]:
            beta -= ghost_w[index]
            index = (index + 1)%N
        g2.append(g[index])
    g = g2


    # display in graph later to evaluate change
    # update list
    p_distances.append(eval(mypacman, p))
    g_distances.append(eval(myghost, g))

    ###################################
    ###################################
    ###################################
    ###### YOUR CODE ENDS HERE ######
```

```
    ##################################
    ##################################
    ##################################

    show_belief(mypacman, p, myghost, g)
    print("Average pacman distance after resample:", eval(mypacman, p))
    print("Average ghost distance after resample:", eval(myghost, g))
```

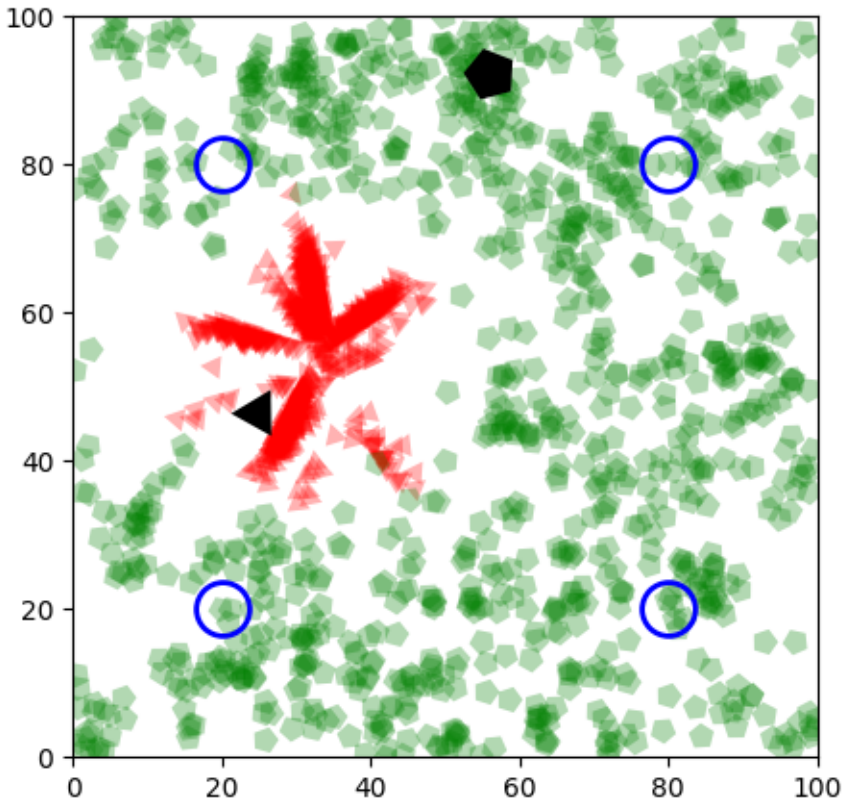
```
Turn #1
```



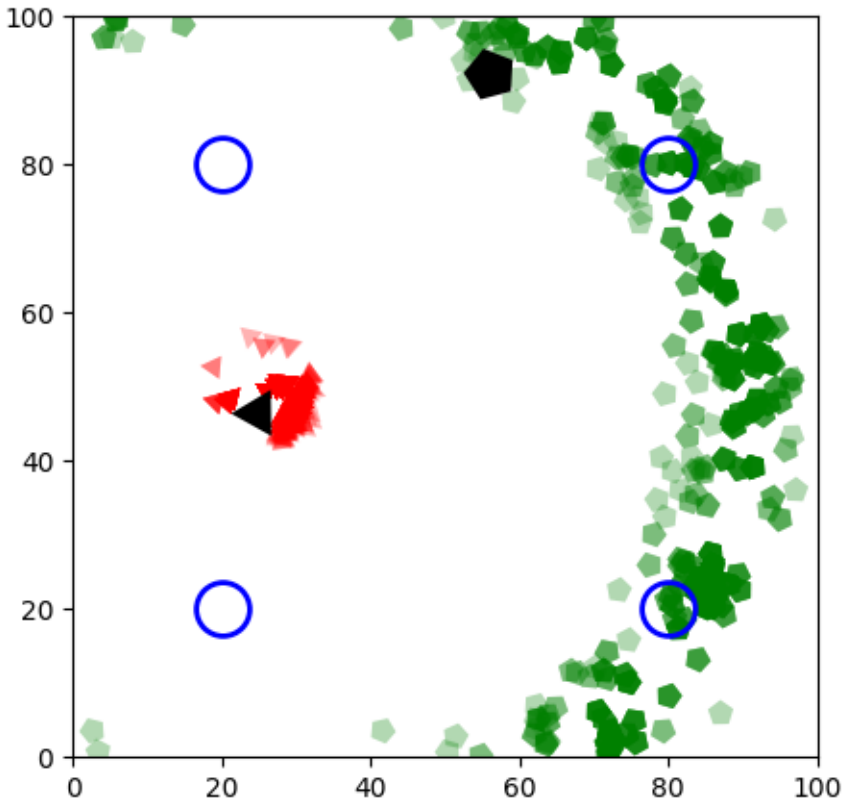

```
Average pacman distance before resample: 38.04391614022457
Average ghost distance before resample: 38.18859447657948
```



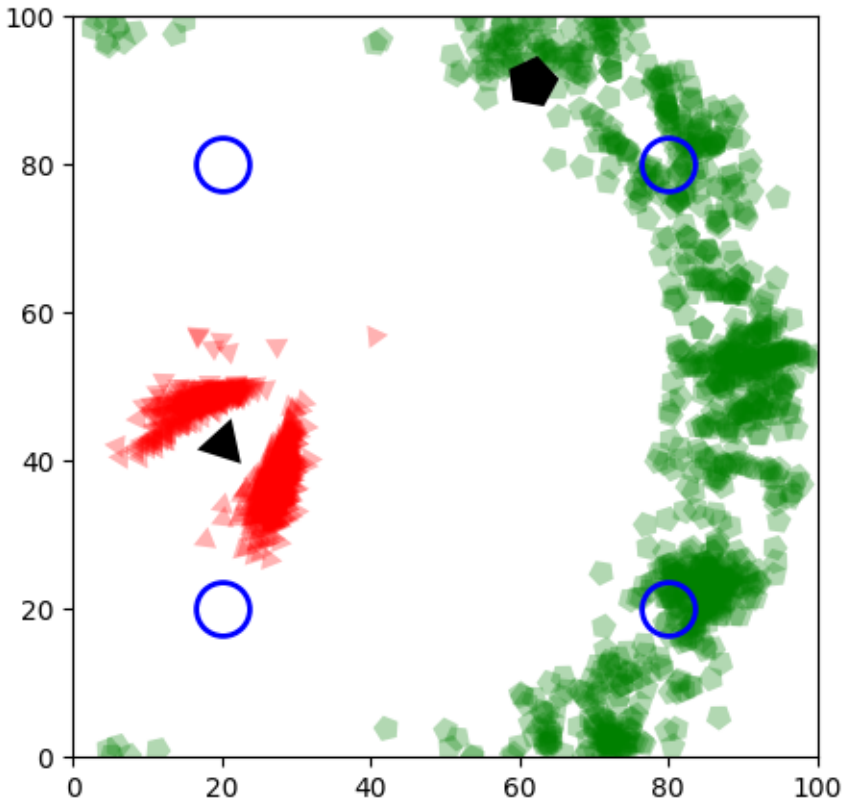

```
Average pacman distance after resample: 3.5626643849837976
Average ghost distance after resample: 31.652337794602595

Turn #2
```



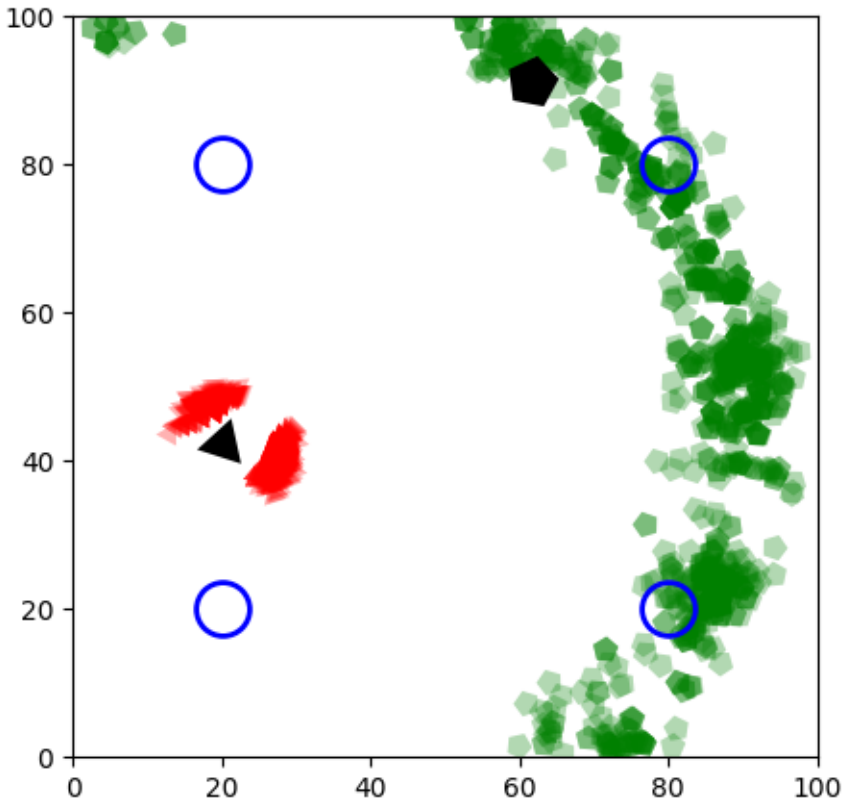

```
Average pacman distance before resample: 14.20313653746869
Average ghost distance before resample: 33.822537370482095
```



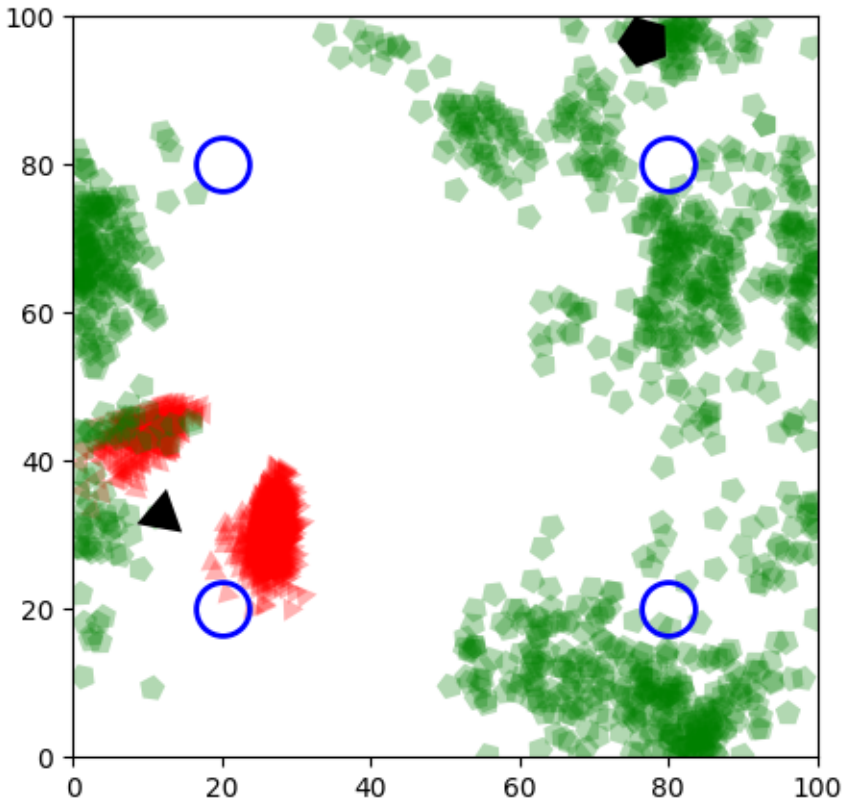

```
Average pacman distance after resample: 5.138157190415509
Average ghost distance after resample: 36.09282373102289

Turn #3
```



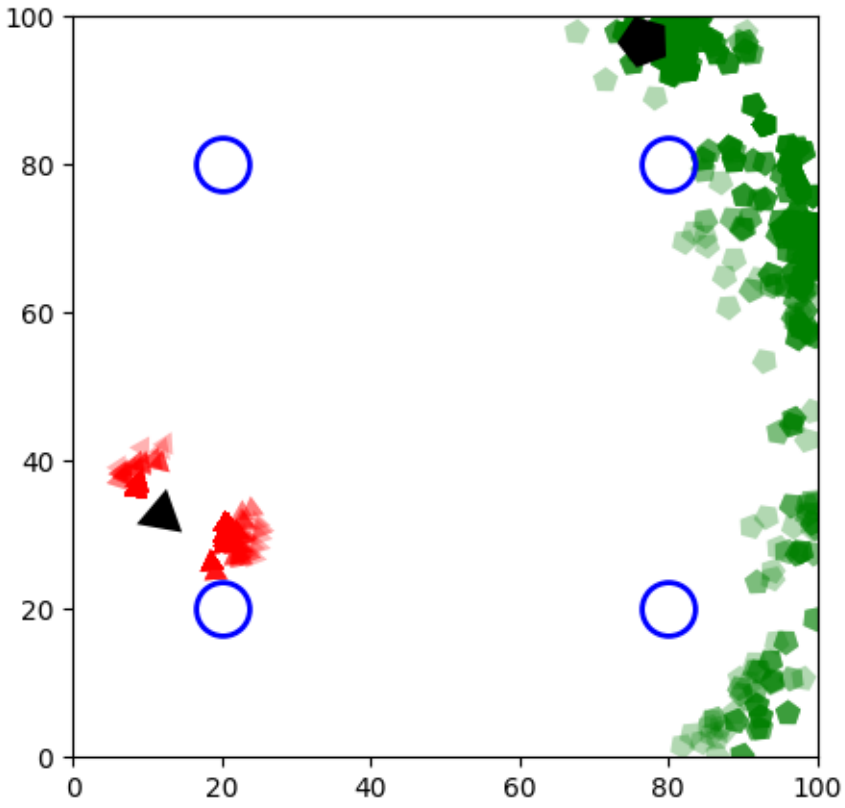

```
Average pacman distance before resample: 8.782781244519049
Average ghost distance before resample: 32.762486439727766
```



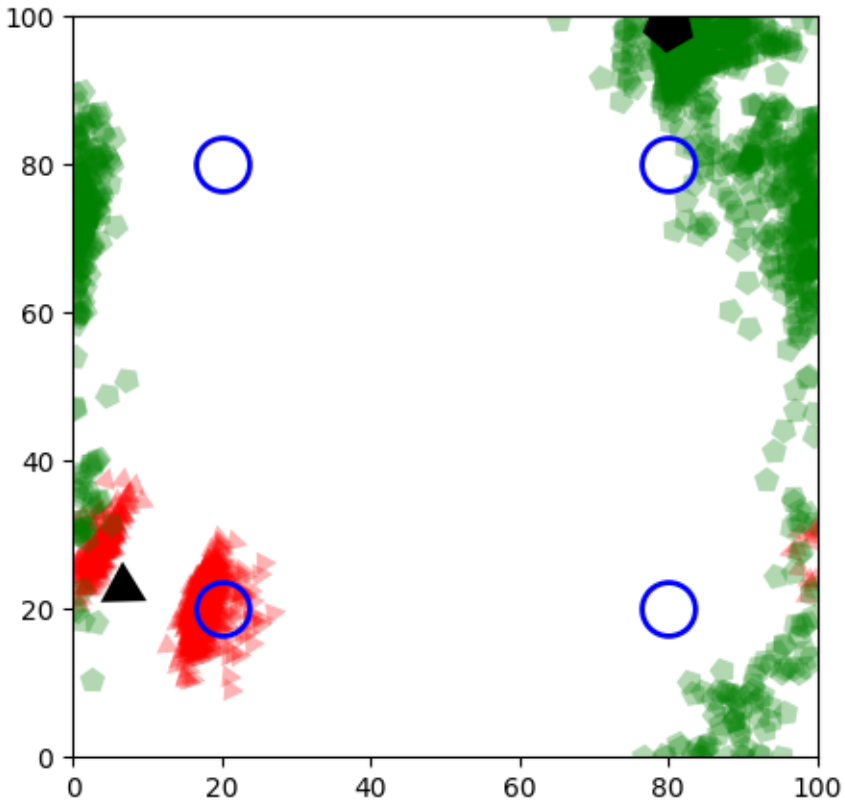

```
Average pacman distance after resample: 7.406850402338689
Average ghost distance after resample: 33.271116888979364

Turn #4
```



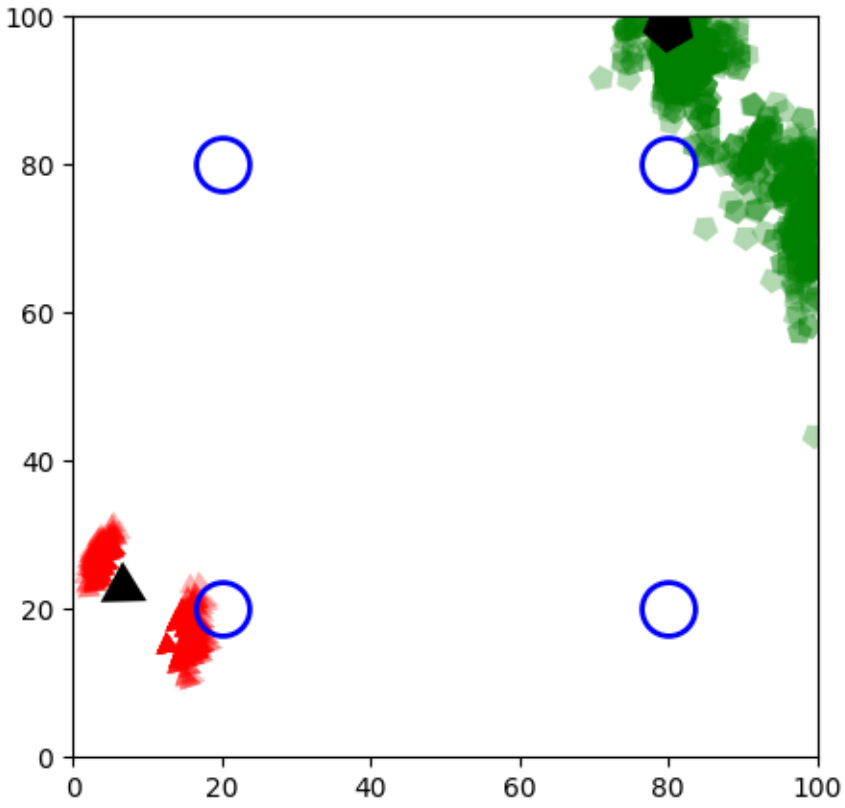

```
Average pacman distance before resample: 14.271308701189911
Average ghost distance before resample: 27.959973249967057
```



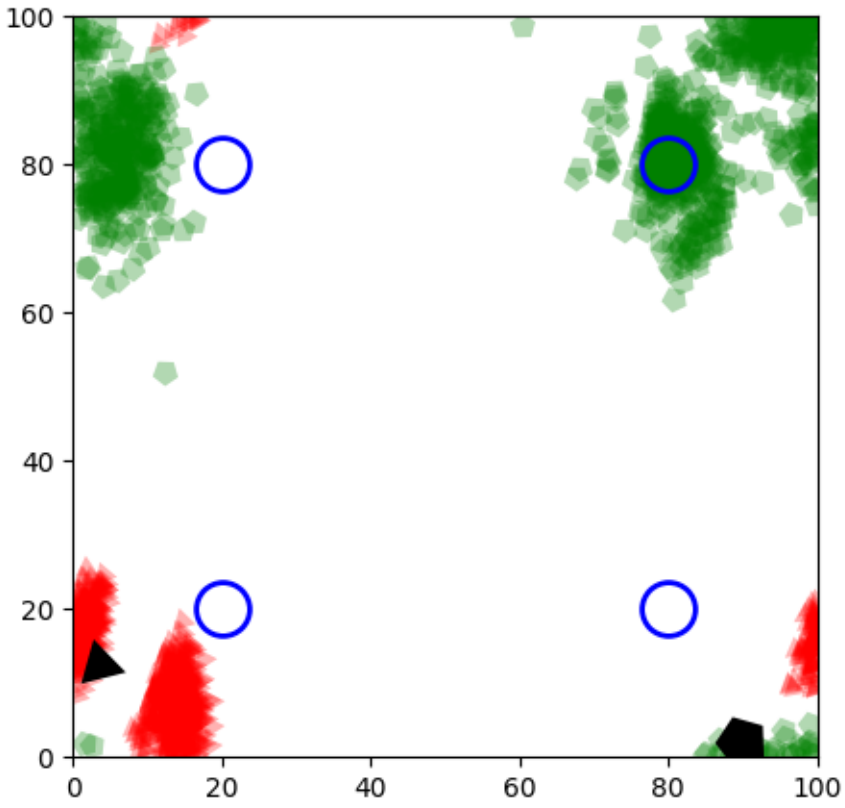

```
Average pacman distance after resample: 8.101508872136627
Average ghost distance after resample: 20.196460901502043

Turn #5
```



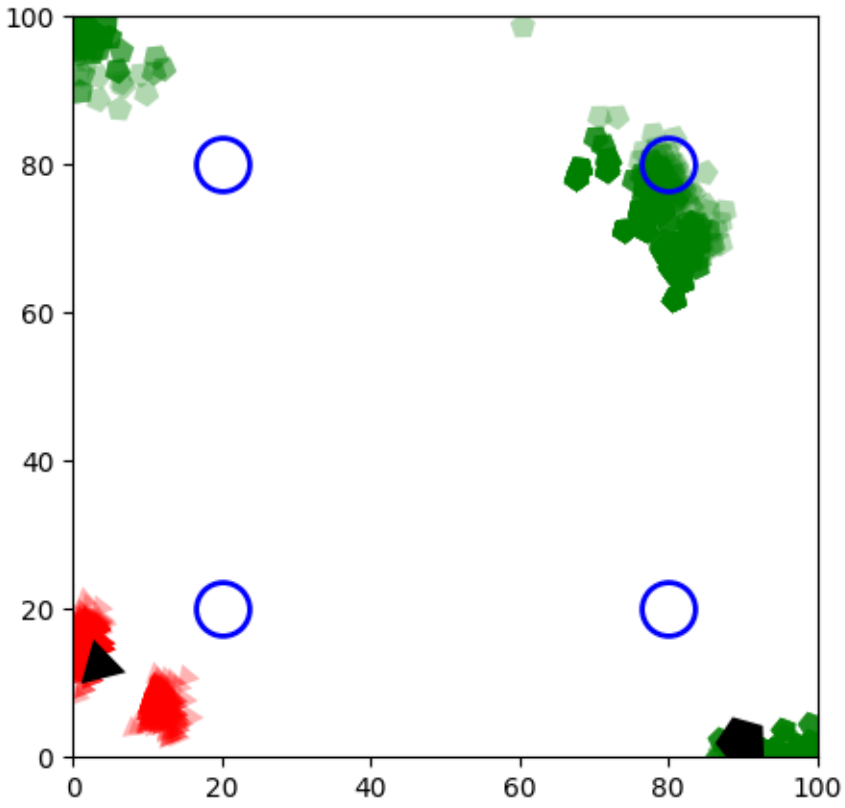

```
Average pacman distance before resample: 10.493443076007452
Average ghost distance before resample: 19.90876376781165
```



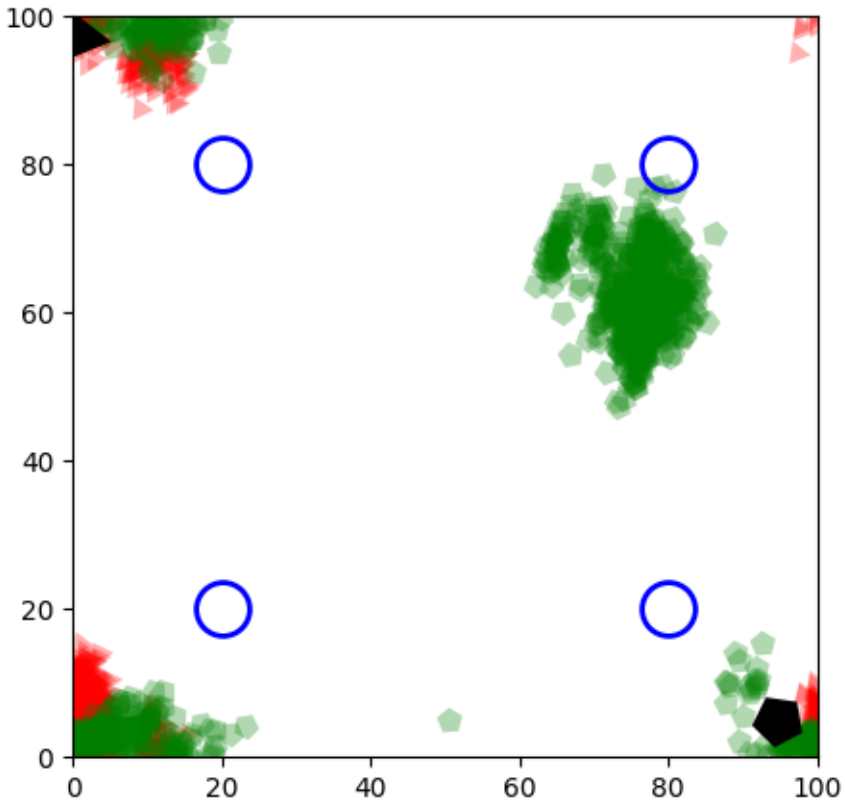

```
Average pacman distance after resample: 8.408364411714128
Average ghost distance after resample: 14.68503480975314

Turn #6
```



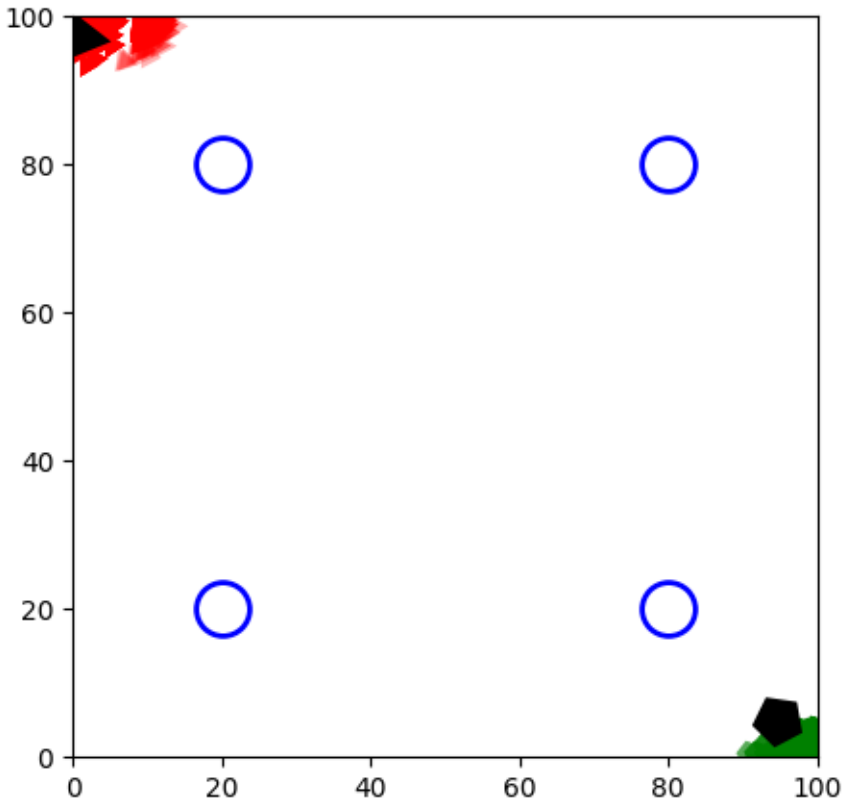

```
Average pacman distance before resample: 9.488697791242435
Average ghost distance before resample: 20.99197831535603
```



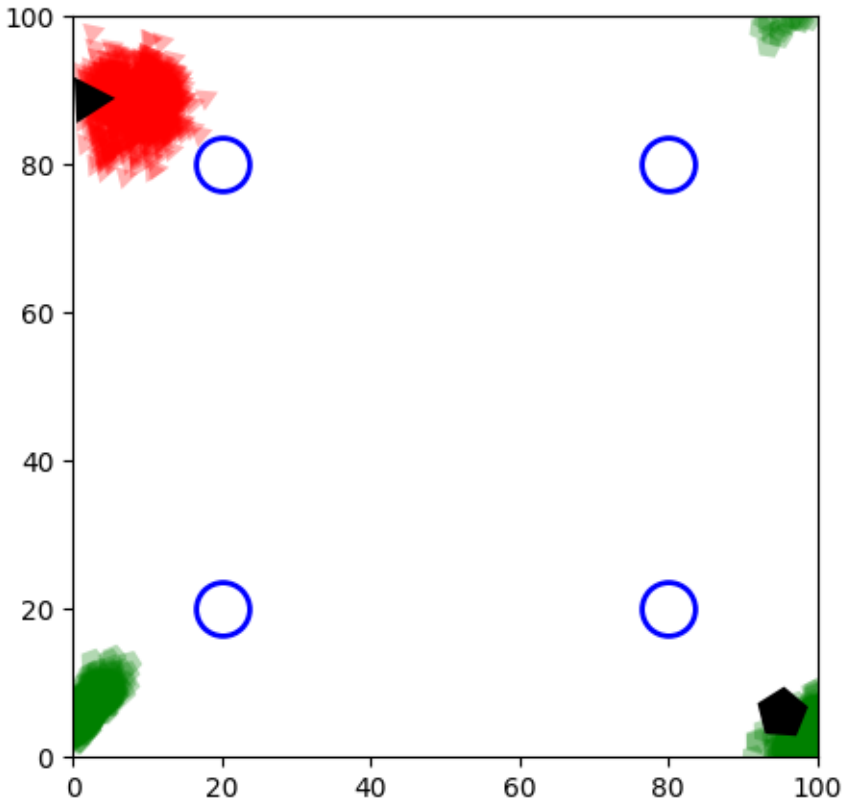

```
Average pacman distance after resample: 5.134345612595122
Average ghost distance after resample: 22.780685086667166

Turn #7
```



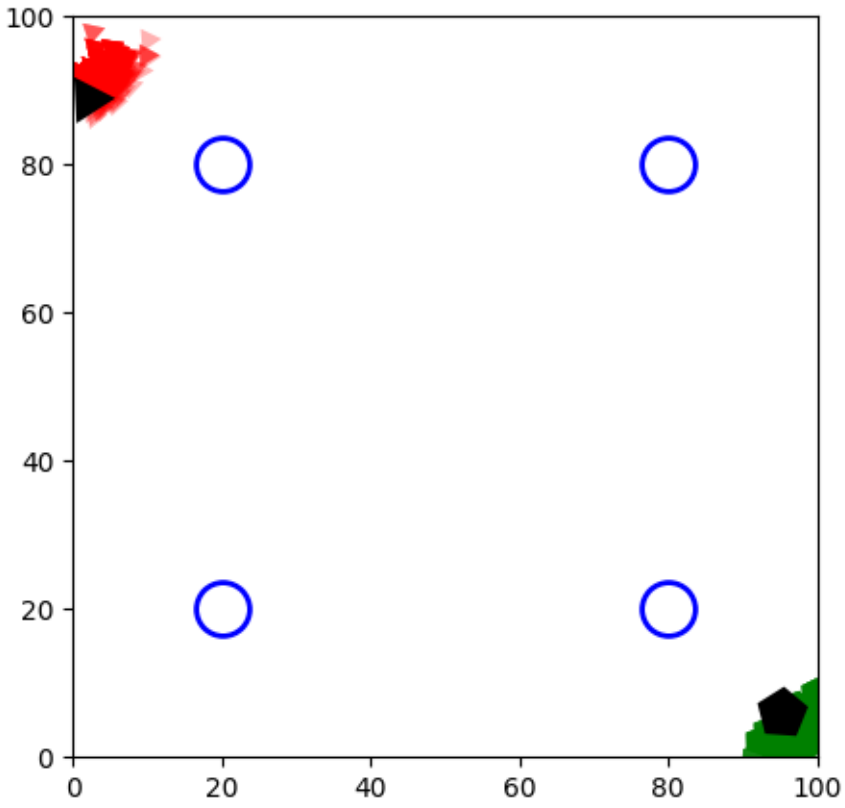

```
Average pacman distance before resample: 8.887836556093847
Average ghost distance before resample: 32.96862334120446
```



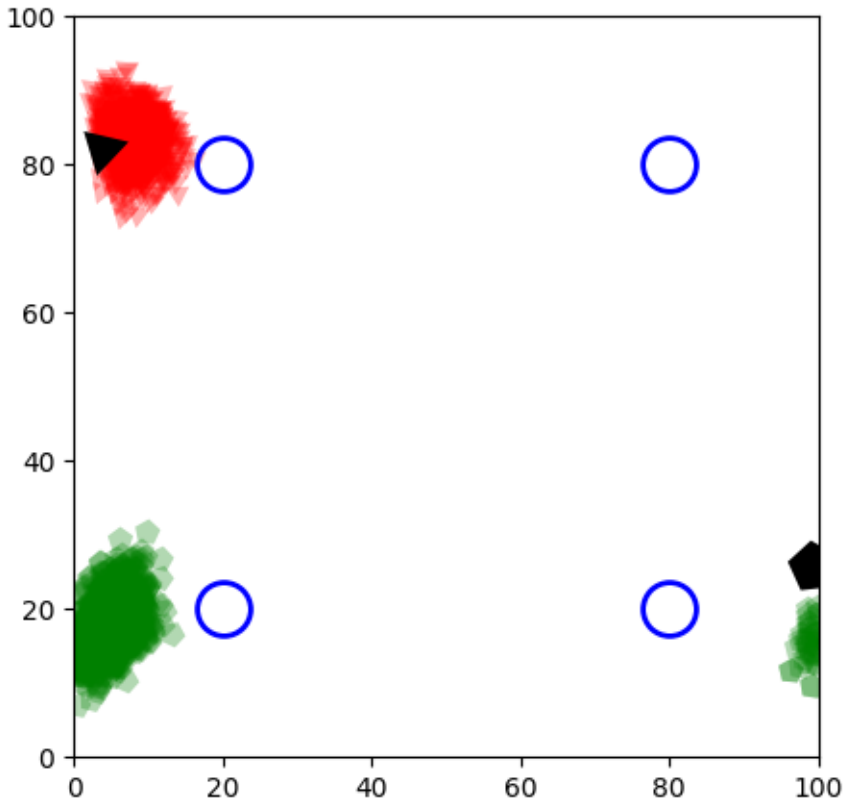

```
Average pacman distance after resample: 5.617434437454068
Average ghost distance after resample: 4.662513522248517

Turn #8
```



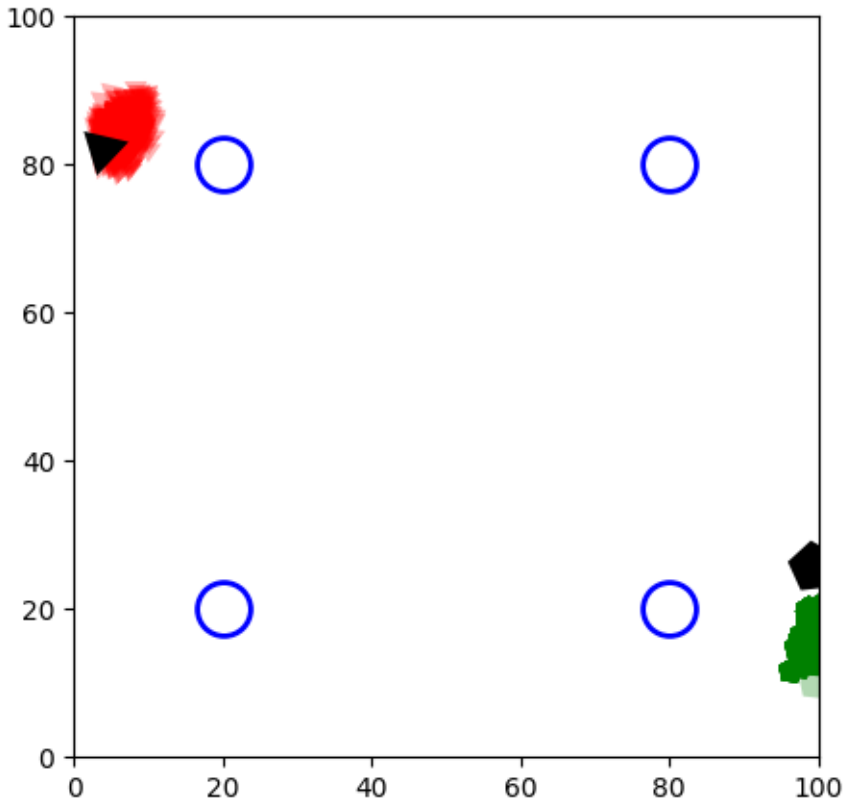

```
Average pacman distance before resample: 7.428266388266711
Average ghost distance before resample: 6.244039842183565
```



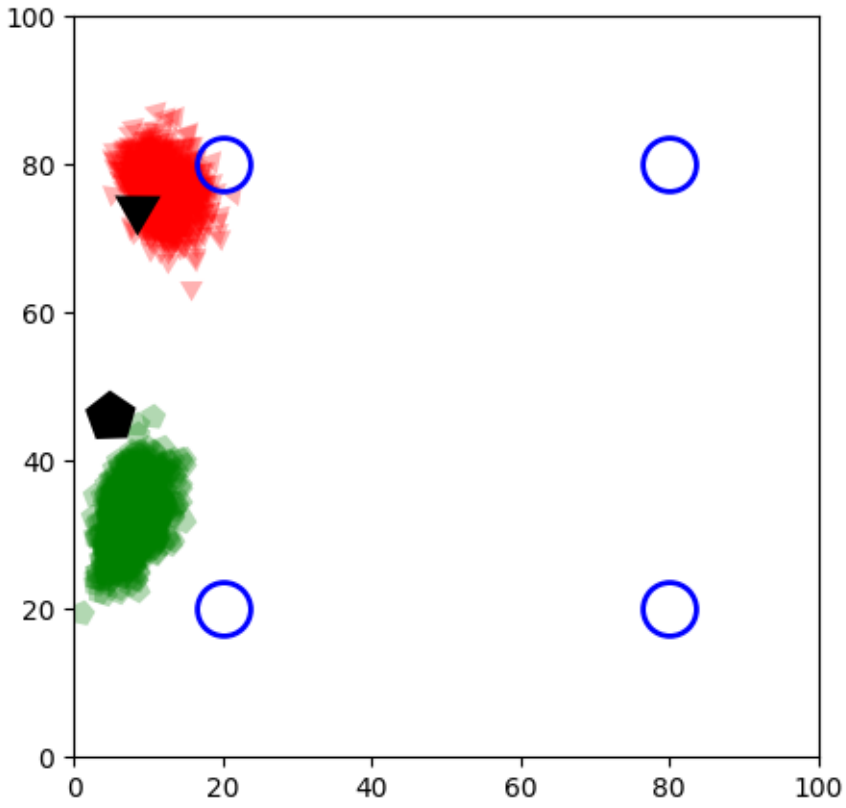

```
Average pacman distance after resample: 4.341248511471954
Average ghost distance after resample: 3.966204313566575

Turn #9
```



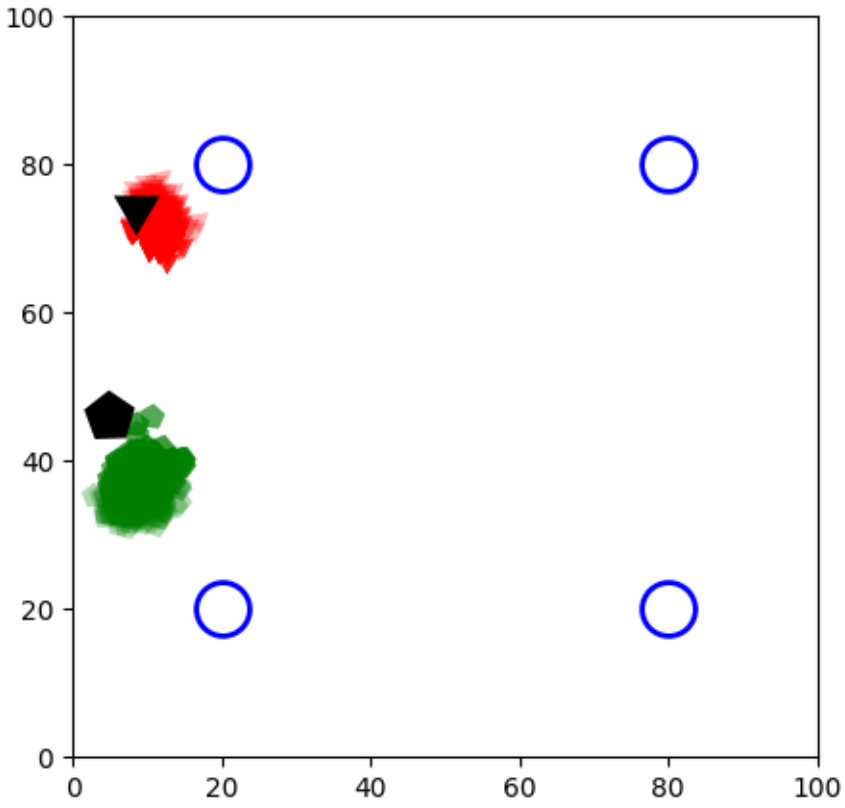

```
Average pacman distance before resample: 5.457317675941225
Average ghost distance before resample: 10.277048484897389
```




```
Average pacman distance after resample: 3.9984402250097038
Average ghost distance after resample: 8.924825764496822

Turn #10
```




```
Average pacman distance before resample: 5.562476186823142
Average ghost distance before resample: 13.60959246354531
```

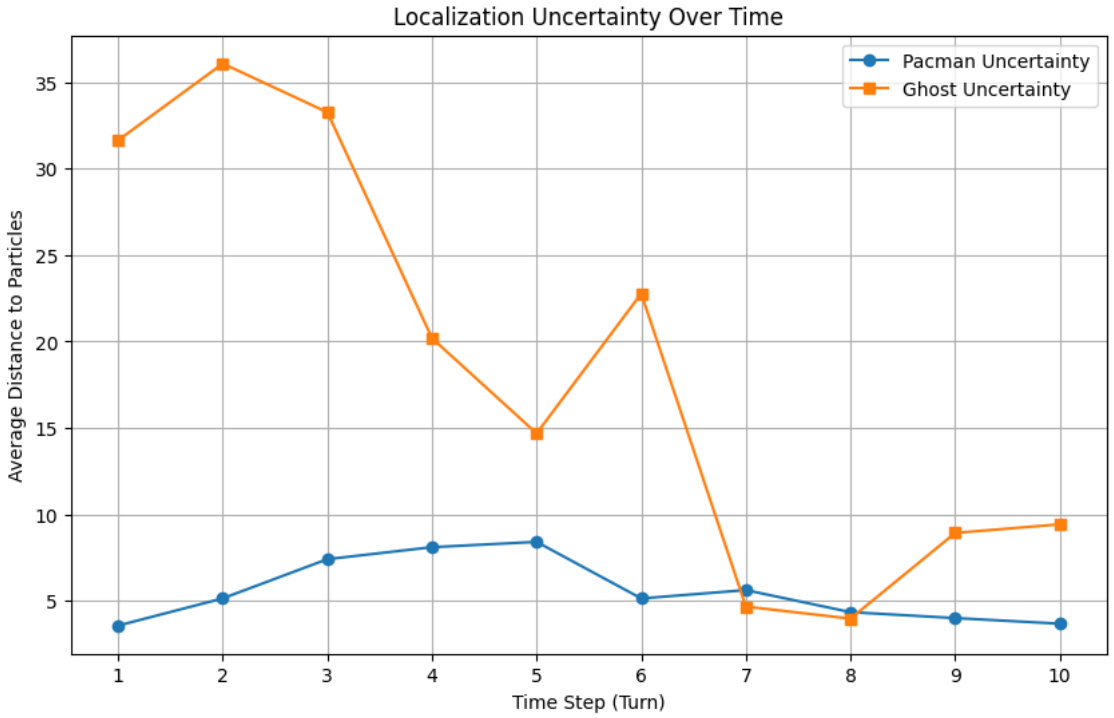



```
Average pacman distance after resample: 3.6735117201170975
Average ghost distance after resample: 9.423897892972706
```

2.0.2 Graph Distances to evaluate change

[6]:
```
# Create a list for the x-axis representing each time step
time_steps = range(1, T + 1)

# Create the plot
plt.figure(figsize=(10, 6)) # Adjust size for better readability
plt.plot(time_steps, p_distances, marker='o', linestyle='-', label='Pacman␣
↪Uncertainty')
plt.plot(time_steps, g_distances, marker='s', linestyle='-', label='Ghost␣
↪Uncertainty')

# Add titles and labels for clarity
plt.title('Localization Uncertainty Over Time')
plt.xlabel('Time Step (Turn)')
plt.ylabel('Average Distance to Particles')
plt.xticks(time_steps) # Ensure x-axis ticks are integers for each turn
plt.grid(True) # Add a grid to make it easier to read values
```

```
plt.legend() # Display the legend to identify the lines

# Show the plot
plt.show()
```



3 Report

3.1 Updating the ghost particles without having the real location of the pacman

The real location of the pacman is unknown, so it cannot be used to update the ghost particles. 1. Instead, an observation is made which gives us the distance from the ghost particle and the pacman at a given time. 2. The distance between the ghost particle and each pacman particle is measured. 3. The ghost particle is given a probability score which is higher if the distances are closer together. 4. All of the probabilities for the ghost particle are then summed and used as a weight for that particle. 5. Once all ghost particles have a weight, they are resampled based on their weights with particles having a higher weight being more likely to be sampled.

3.1.1 Observation:

1. An observation is made (The ghost is 10m away from the pacman: ZG = 10).

3.1.2 Downweighing:

2. For every ghost particle (g), iterate through all pacman particles (p):

a. compare the distance between the particles.
b. if the distance is close to the value of the observation, the particle is assigned a higher probability score (the probability of the particle representing the observation (.90))
c. sum all of the probabilities to get a weight
d. the weight is then assigned to the ghost particle (weight: 10.23, max_weight=15.00) Note: a ghost particle with a high weight is a particle that was close to the observed distance to many pacman particles

3.1.3 Resampling:

3. Resample the distribution (choose with replacement) using the weights (higher weight = more likley to be chosen).

3.2 Alternatives I considered and why I decided not to use them

I considered implementing a method where the average location of all of the pacman particles is used as a single point of reference to measure the distance for each ghost particle. Instead, I chose to implement marginalization(a method where for each ghost particle, the distance between it and every pacman particle is measured and used as a probability rather than just a single averaged point of reference). This method provides many more points of reference which in turn provides a more accurate weight distribution for resampling. The single point method loses important information throughout the process and performs worse than marginilization, as marginilization uses many more points of reference and does not lose the critical information.

4 Experimental results

4.1 Comparing the Uncertainty in the Location of the Pacman and the Ghost over time

The plot above shows the uncertainty in the location of the pacman and ghost over time. As seen, the pacman uncertainty drastically decreases after step 1 and remains consistently low for the remainder of the test. The ghost uncertainty remains high for the first few iterations before decreasingly rapidly. Then the ghost uncertainty increases once again before decreasing and remaining low for the remainder of the test.

4.1.1 Pacman Uncertainty

The pacman uncertainty rapidly decreases at the beggining of the test displaying an immediately high accuracy of predicting the location of the pacman and remains low for the remainder of the test. This is due to its ability to measure the distance from itself and the four pillars found in the test area. The pacman uncertainty slightly increases at times, but this is likely due to the noise in the data as well as the teleporting behavior of the pacman's location as it crosses a test area boundary.

4.1.2 Ghost Uncertainty

The ghost uncertainty is high at the beggining of the test before decreasing, increasing, and decreasing once more. The ghost particles show a tendency to form a circle around the pacman's predicted location. This is likely due to the fact that the only information received is the distance

from the predicted pacman location (ZG). The ghost particles that are the observed distance from the most pacman particles with be weighed the highest and therefore most likely to be chosen during resampling. The ghost particles eventually collapse into a smaller circle closer to the pacman and the ghost uncertainty decreases and remains low. This behavior is most likely due to the combination of noise present, the ghost getting within a close proximity of the pacman, and the teleporting location of the pacman when crossing a boundary.

4.2 The ghost particles tend to concentrate in a well-defined cluster after many iterations

This occurs naturally after many iterations of downweighing and resampling the ditribution. The observation and downweighing steps allow for the accumulation of data. Each step, the ghost location is confined to a cluster of particles. Each step, the cluster of particles is repopulated. Over time, the clusters overlap and the overlap grows smaller. This increases our ability to predict the ghost location, as our best prediction lies within the overlap area.

[ ]: