

In [1]:

```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, Aer, transpile
2 import numpy as np
3 from qiskit.visualization import plot_histogram
4 from qiskit import *
5 import random
6 import matplotlib.pyplot as plt
7 from operator import attrgetter
8 import matplotlib.pyplot as plt
9 import heapq
10 from operator import itemgetter
11 from pydub import AudioSegment # for audio
12 from pydub.playback import play # for audio
13
14 import statistics # added for the mean computation
15 from collections import defaultdict # added to compare elements of the list
16 from itertools import tee # to allow pairwise comparisons
17 from scipy.spatial.distance import cosine # to compute cosine distance

```

In []:

1

In [2]:

```
1 # Target & reward -----
```

In [3]:

```

1 class Target:
2     def __init__(self, name, x, y): # no indetermination in the target's position
3         self.name = name
4         self.x = x
5         self.y = y

```

In [4]:

```

1 T = Target("T", 0.8, 0.9) # PSO comparison of September 14
2 # T = Target("T", 0.9, 0.5) # deep in the ocean
3
4 # for getting back to the beginning
5 T2 = Target("T2", 0.2, 0.5) # back to the ship

```

In [5]:

```

1 def reward(T, betax, betay):
2     return 1 - ((T.x - betax)**2 + (T.y - betay)**2)**0.5
3     # the closer the target, the less the distance, the higher the reward
4

```

In [6]:

```
1 # Obstacles -----
```

In [7]:

```

1 class Obstacle: # Just a point for now
2     def __init__(self,name,x,y):
3         self.name = name
4         self.x = x
5         self.y = y

```

In [8]:

```

1 O = Obstacle("Oo", 0.8, 0.2) # deep in the ocean

```

In [9]:

```

1 # Robots -----

```

In [10]:

```

1 class Robotx(object):
2     _registry = []
3
4     def __init__(self, name, alphax, betax, alphay, betay, gamma, delta, position):
5         self._registry.append(self)
6         self.name = name
7         self.alphax = alphax
8         self.betax = betax
9         self.alphay = alphay
10        self.betay = betay
11        delta = reward(T, betax, betay)
12        gamma = 1 - delta
13        self.gamma = gamma
14        self.delta = delta
15        self.position = position # new -- I need it for sound

```

In []:

```

1

```

In [11]:

```

1 # arbitrary number of robots that, at the start, are uniformly distributed in the
2 # centered in starting_cluster_coord
3 #
4 num_of_robots = 10
5 radius = 0.1
6 # starting_cluster_coord = (0.6, 0.6)
7 starting_cluster_coord = (0.2, 0.5)
8
9 a_x, a_y = 1-starting_cluster_coord[0]-radius, 1-starting_cluster_coord[0]+radius
10 b_x, b_y = 1-starting_cluster_coord[1]-radius, 1-starting_cluster_coord[1]+radius
11
12 for i in range(num_of_robots):
13     x = random.uniform(a_x,a_y)
14     y = random.uniform(b_x,b_y)
15     Robotx('R'+str(i), x, 1-x, y, 1-y, 1 - reward(T, 1-x, 1-y), reward(T, 1-x, 1-y))

```

In [12]:

```
1 # note: values are stored with full precision, rounding is done only on visualiz
2
3 for k in Robotx._registry:
4     print(f"{k.name} {k.betax:.2f} {k.betay:.2f} {k.gamma:.2f} {k.delta:.2f} {k.

R0 0.11 0.44 0.83 0.17 1
R1 0.11 0.53 0.78 0.22 2
R2 0.11 0.45 0.83 0.17 3
R3 0.26 0.59 0.62 0.38 4
R4 0.28 0.54 0.63 0.37 5
R5 0.14 0.57 0.74 0.26 6
R6 0.21 0.58 0.67 0.33 7
R7 0.28 0.41 0.72 0.28 8
R8 0.23 0.41 0.75 0.25 9
R9 0.11 0.49 0.80 0.20 10
```

In [13]:

```
1 for k in Robotx._registry:
2     print(f"{k.name} {k.delta:.2f}")

R0 0.17
R1 0.22
R2 0.17
R3 0.38
R4 0.37
R5 0.26
R6 0.33
R7 0.28
R8 0.25
R9 0.20
```

In [14]:

```

1  #for k in Robotx._registry:
2      #print(statistics.mean(k.betax))
3      #k.betax + (k+1).betax
4
5  # explanation here:
6  # https://stackoverflow.com/questions/10879867/sum-average-an-attribute-of-a-list
7
8  # September 13, 2022
9
10
11 # sum_x = sum(k.betax for k in Robotx._registry)
12 # sum_y = sum(k.betay for k in Robotx._registry)
13
14 listX = list(k.betax for k in Robotx._registry)
15 listY = list(k.betay for k in Robotx._registry)
16
17 def distance_A(T, listX, listY):
18     sum_x = sum(listX)
19     sum_y = sum(listY)
20     center_x = sum_x/num_of_robots
21     center_y = sum_y/num_of_robots
22     return ((T.x - center_x)**2 + (T.y - center_y)**2)**0.5
23
24 print("distance_A", distance_A(T, listX, listY))
25
26 def Euclidean_distance(T, listX, listY): # the same as distance_A
27     sum_x = sum(listX)
28     sum_y = sum(listY)
29     center_x = sum_x/num_of_robots
30     center_y = sum_y/num_of_robots
31     return ((T.x - center_x)**2 + (T.y - center_y)**2)**0.5
32
33 print("Euclidean", Euclidean_distance(T, listX, listY))
34
35 def Manhattan_distance(T, listX, listY):
36     sum_x = sum(listX)
37     sum_y = sum(listY)
38     center_x = sum_x/num_of_robots
39     center_y = sum_y/num_of_robots
40     return (abs(T.x - center_x) + abs(T.y - center_y))
41
42 print("Manhattan", Manhattan_distance(T, listX, listY))
43
44 def Cosine_distance(T, listX, listY):
45     sum_x = sum(listX)
46     sum_y = sum(listY)
47     center_x = sum_x/num_of_robots
48     center_y = sum_y/num_of_robots
49     array_1 = np.array([center_x, T.x])
50     array_2 = np.array([center_y, T.y])
51     return cosine(array_1, array_2)
52
53 print("Cosine", Cosine_distance(T, listX, listY))

```

```

distance_A 0.7351252716423883
Euclidean 0.7351252716423883
Manhattan 1.016631328153391
Cosine 0.03966653152082844

```

In [15]:

```

1 # September 13
2
3 # function taken from https://stackoverflow.com/questions/31044711/method-to-get
4
5 def pairwise(iterable):
6     "s -> (s0,s1), (s1,s2), (s2, s3), ..."
7     a, b = tee(iterable)
8     next(b, None)
9     return zip(a, b) # not izip

```

In [16]:

```

1 # September 13
2
3 # method adapted from https://stackoverflow.com/questions/31044711/method-to-get
4
5 # with the Euclidean distance rather than the simple difference
6
7 def distance_B(listX, listY):
8     return (max((b - a)**2 for (a,b) in pairwise(listX)) + max((c - d)**2 for (c,d) in pairwise(listY)))
9

```

In [17]:

```

1 # September 13
2
3 # classic one! "within-cluster distance"
4 # distance between the swarm barycenter (as a centroid) and each element
5
6 # distance_A can be seen as a particular case of between-cluster distance (distance between two clusters)
7 # where the second cluster is actually only a point (the target)
8
9 # "within cluster sum of squares"
10
11 def distance_C(listX, listY):
12     sum_x = sum(listX)
13     sum_y = sum(listY)
14     center_x = sum_x/num_of_robots
15     center_y = sum_y/num_of_robots
16     return (max((center_x - a)**2 for a in listX) + max((center_y - b)**2 for b in listY))
17
18 print(distance_C(listX, listY))

```

0.13576264249015121

In [18]:

```

1 # September 13
2
3 print(distance_A(T, listX, listY), distance_B(listX, listY), distance_C(listX, listY))

```

0.7351252716423883 0.2367617792807916 0.13576264249015121

In []:

1

In []:

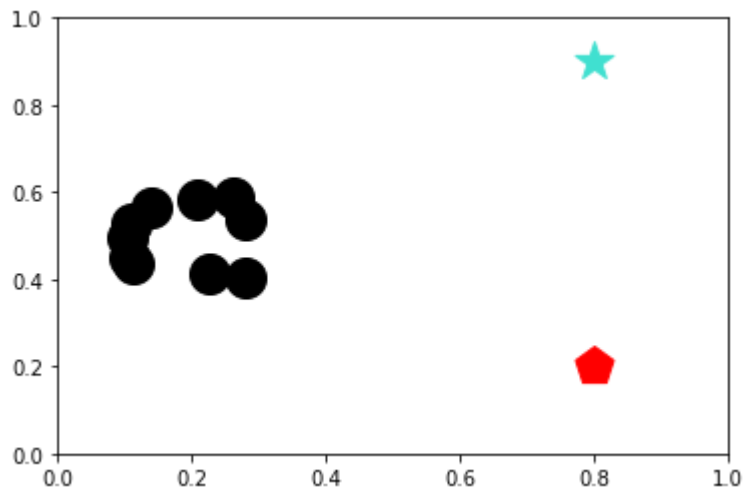
1

In []:

1

In [19]:

```
1 def plot_scatterplot():
2     for i in Robotx._registry:
3         plt.scatter(i.betax, i.betay, s = 400, marker = 'o', color = 'black')
4
5     plt.scatter(T.x, T.y, s = 400, marker = '*', color = 'turquoise')
6     plt.scatter(O.x, O.y, s = 400, marker = 'p', color = 'red')
7
8     plt.axis([0, 1, 0, 1])
9
10    plt.show()
11
12 plot_scatterplot()
```



In [20]:

```

1  # initialization of sound parameters
2
3
4  # we need 'append' to create such a list!
5
6  l = []
7  for x in range(11):
8      value = AudioSegment.from_file("notes_/tC.mp3")
9      l.append(value)
10  for i in range(11):
11      print(l[i])
12
13  for k in Robotx._registry:
14      print(k.position)
15
16  for k in Robotx._registry:
17      print(l[k.position])

```

```

<pydub.audio_segment.AudioSegment object at 0x7facb07e9250>
<pydub.audio_segment.AudioSegment object at 0x7facb0884d90>
<pydub.audio_segment.AudioSegment object at 0x7facb0884ca0>
<pydub.audio_segment.AudioSegment object at 0x7fac937486a0>
<pydub.audio_segment.AudioSegment object at 0x7fac93748a60>
<pydub.audio_segment.AudioSegment object at 0x7fac93748c40>
<pydub.audio_segment.AudioSegment object at 0x7fac93748250>
<pydub.audio_segment.AudioSegment object at 0x7fac93748520>
<pydub.audio_segment.AudioSegment object at 0x7fac93748160>
<pydub.audio_segment.AudioSegment object at 0x7fac93748700>
<pydub.audio_segment.AudioSegment object at 0x7fac93769400>
1
2
3
4
5
6
7
8
9
10
<pydub.audio_segment.AudioSegment object at 0x7facb0884d90>
<pydub.audio_segment.AudioSegment object at 0x7facb0884ca0>
<pydub.audio_segment.AudioSegment object at 0x7fac937486a0>
<pydub.audio_segment.AudioSegment object at 0x7fac93748a60>
<pydub.audio_segment.AudioSegment object at 0x7fac93748c40>
<pydub.audio_segment.AudioSegment object at 0x7fac93748250>
<pydub.audio_segment.AudioSegment object at 0x7fac93748520>
<pydub.audio_segment.AudioSegment object at 0x7fac93748160>
<pydub.audio_segment.AudioSegment object at 0x7fac93748700>
<pydub.audio_segment.AudioSegment object at 0x7fac93769400>

```

In [21]:

```

1  # audio block #1
2
3  # audio 1
4
5  # we can define "audio" as an attribute... no, better not.
6
7  audio = []
8
9  for x in range(11): # it should be between 1 and 11
10     valuex = AudioSegment.from_file("notes_/tC.mp3")
11     audio.append(valuex)
12  for i in range(11):
13     print(audio[i]) # at this stage, they are supposed to all give tC.mp3
14
15  for i in Robotx._registry:
16     if (i.betax == 0):
17         if (i.betay == 0.5):
18             valuex = AudioSegment.from_file("notes_/tC.mp3") # i.audio
19             audio.append(valuex)
20             print("tC")
21     if (i.betax > 0 and i.betax <= 0.17):
22         if (i.betay < 0.5):
23             valuex = AudioSegment.from_file("notes_/tB.mp3")
24             audio.append(valuex)
25             print("tB")
26         if (i.betay >= 0.5):
27             valuex = AudioSegment.from_file("notes_/tC#.mp3")
28             audio.append(valuex)
29             print("tC#")
30     if (i.betax > 0.17 and i.betax <= 0.3):
31         if (i.betay < 0.5): # if (R1.betay >= 0.17 and R1.betay < 0.3):
32             valuex = AudioSegment.from_file("notes_/tA#.mp3")
33             audio.append(valuex)
34             print("tA#")
35         if (i.betay >= 0.5):
36             valuex = AudioSegment.from_file("notes_/tD.mp3")
37             audio.append(valuex)
38             print("tD")
39     if (i.betax > 0.3 and i.betax <= 0.5):
40         if (i.betay < 0.5): # (R1.betay == 1):
41             valuex = AudioSegment.from_file("notes_/tD#.mp3")
42             audio.append(valuex)
43             print("tD#")
44         if (i.betay >= 0.5):
45             valuex = AudioSegment.from_file("notes_/tA.mp3")
46             audio.append(valuex)
47             print("tA")
48     if (i.betax > 0.5 and i.betax <= 0.64):
49         if (i.betay < 0.5):
50             valuex = AudioSegment.from_file("notes_/tE.mp3")
51             audio.append(valuex)
52             print("tE")
53         if (i.betay >= 0.5):
54             valuex = AudioSegment.from_file("notes_/tG#.mp3")
55             audio.append(valuex)
56             print("tG#")
57     if (i.betax > 0.64 and i.betax <= 0.84):
58         if (i.betay < 0.5):
59             valuex = AudioSegment.from_file("notes_/tF.mp3")

```



```

60         audio.append(valuex)
61         print("tF")
62     if (i.betay >= 0.5):
63         valuex = AudioSegment.from_file("notes_/tG.mp3")
64         audio.append(valuex)
65         print("tG")
66     if (i.betax > 0.84 and i.betax <= 1):
67         #if (R1.betay == 0.5):
68         valuex = AudioSegment.from_file("notes_/tF#.mp3")
69         audio.append(valuex)
70         print("tF#")
71
72
73
74 for i in Robotx._registry:
75     print(audio[i.position]) # at this stage, they are supposed to all give tC.
76
77
78
79
80 mix = []
81
82 for s in range(11): # it should be between 1 and 11
83     #values = (audio[s].overlay(audio[s+1])).overlay(audio[s+3])
84
85     # is there a more synthetic way to write this??
86     values = audio[s].overlay(audio[s+1])
87     values2 = values.overlay(audio[s+2])
88     values3 = values2.overlay(audio[s+3])
89     values4 = values3.overlay(audio[s+4])
90     values5 = values4.overlay(audio[s+5])
91     values6 = values5.overlay(audio[s+6])
92     values7 = values6.overlay(audio[s+7])
93     values8 = values7.overlay(audio[s+8])
94     values9 = values8.overlay(audio[s+9])
95     mix.append(values9)
96     print(mix[s])
97
98 mix[10].export("notes_/10_robot_sound/mixed_time_1.mp3", format='mp3') # export
99 play(mix[10])
100

```

```

<pydub.audio_segment.AudioSegment object at 0x7fac937bcf70>
<pydub.audio_segment.AudioSegment object at 0x7facc0663820>
<pydub.audio_segment.AudioSegment object at 0x7facc06630d0>
<pydub.audio_segment.AudioSegment object at 0x7facc0663040>
<pydub.audio_segment.AudioSegment object at 0x7facc06630a0>
<pydub.audio_segment.AudioSegment object at 0x7facc0663790>
<pydub.audio_segment.AudioSegment object at 0x7facc066e460>
<pydub.audio_segment.AudioSegment object at 0x7facc066e220>
<pydub.audio_segment.AudioSegment object at 0x7facc066e160>
<pydub.audio_segment.AudioSegment object at 0x7facc066e250>
<pydub.audio_segment.AudioSegment object at 0x7facc066e070>
tB
tC#
tB
tD
tD
tC#
tD
tA#

```

```

tA#
tB
<pydub.audio_segment.AudioSegment object at 0x7facc0663820>
<pydub.audio_segment.AudioSegment object at 0x7facc06630d0>
<pydub.audio_segment.AudioSegment object at 0x7facc0663040>
<pydub.audio_segment.AudioSegment object at 0x7facc06630a0>
<pydub.audio_segment.AudioSegment object at 0x7facc0663790>
<pydub.audio_segment.AudioSegment object at 0x7facc066e460>
<pydub.audio_segment.AudioSegment object at 0x7facc066e220>
<pydub.audio_segment.AudioSegment object at 0x7facc066e160>
<pydub.audio_segment.AudioSegment object at 0x7facc066e250>
<pydub.audio_segment.AudioSegment object at 0x7facc066e070>
<pydub.audio_segment.AudioSegment object at 0x7facc0663b80>
<pydub.audio_segment.AudioSegment object at 0x7facc0663be0>
<pydub.audio_segment.AudioSegment object at 0x7facc06637f0>
<pydub.audio_segment.AudioSegment object at 0x7facc066e0d0>
<pydub.audio_segment.AudioSegment object at 0x7facc0663100>
<pydub.audio_segment.AudioSegment object at 0x7facc066e190>
<pydub.audio_segment.AudioSegment object at 0x7facc066e3d0>
<pydub.audio_segment.AudioSegment object at 0x7facc066e130>
<pydub.audio_segment.AudioSegment object at 0x7facd013b190>
<pydub.audio_segment.AudioSegment object at 0x7facd013b460>
<pydub.audio_segment.AudioSegment object at 0x7facc066e400>
Could not import the PyAudio C module '_portaudio'.

```

```

avplay version 12.3, Copyright (c) 2003-2018 the Libav developers
  built on Nov  2 2021 03:53:01 with Apple clang version 13.0.0 (clang
-1300.0.29.3)
Failed to set value '-hide_banner' for option 'autoexit'

```

In [22]:

```

1 for r in Robotx._registry:
2     if (r.delta < 0.5):
3         print(f"{r.name} {r.delta:.2f} achtung!") # and start from this point to

```

```

R0 0.17 achtung!
R1 0.22 achtung!
R2 0.17 achtung!
R3 0.38 achtung!
R4 0.37 achtung!
R5 0.26 achtung!
R6 0.33 achtung!
R7 0.28 achtung!
R8 0.25 achtung!
R9 0.20 achtung!

```

In []:

```

1
2

```

In [23]:

```

1 # Reshuffling -----

```

In []:

```

1

```

In [24]:

```

1  # I'm adding this one as the only non-quantum thing:
2
3  result = all(i.delta < 0.8 for i in Robotx._registry)
4
5  # Printing result
6  print("Do all the robots have a reward lower than 0.8? : " + str(result))
7
8  # if True: reshuffle positions
9  # if False: do nothing
10
11 if result == True:
12     flag = True
13     while flag:
14         flag = False
15         for i in Robotx._registry:
16             i.alphax = np.random.uniform(0,0.9)
17             i.betax = 1 - i.alphax
18             i.alphay = np.random.uniform(0,0.9)
19             i.betay = 1 - i.alphay
20             if (i.betax - 0.x <= 0.2 and i.betay - 0.y <= 0.2 <= 0.2):
21                 flag = True

```

Do all the robots have a reward lower than 0.8? : True

In [25]:

```

1  for k in Robotx._registry:
2      print(f"{k.name} {k.betax:.2f} {k.betay:.2f} {k.gamma:.2f} {k.position}")

```

```

R0 0.39 0.66 0.83 1
R1 0.50 0.96 0.78 2
R2 0.16 0.85 0.83 3
R3 0.74 0.94 0.62 4
R4 0.45 0.98 0.63 5
R5 0.38 0.74 0.74 6
R6 0.56 0.98 0.67 7
R7 0.73 0.85 0.72 8
R8 0.28 0.79 0.75 9
R9 0.58 0.87 0.80 10

```

In [26]:

```

1  for i in Robotx._registry: # recalculate the rewards
2      i.delta = reward(T, i.betax, i.betay)
3      i.gamma = 1 - i.delta
4      print(f"{i.name} {i.delta:.2f}")

```

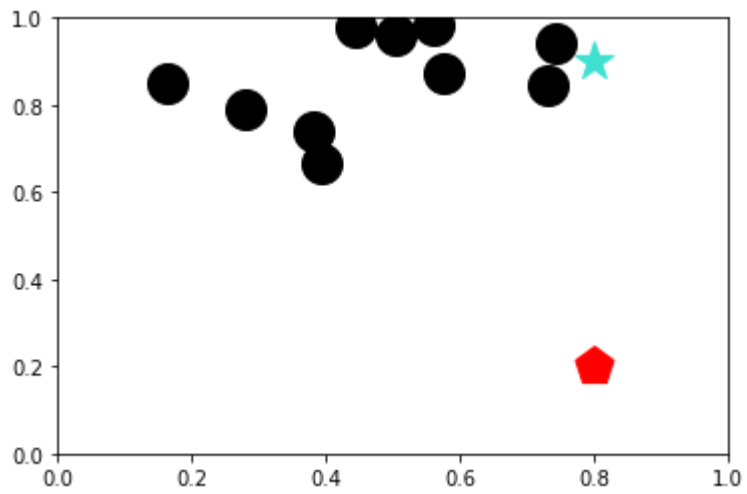
```

R0 0.53
R1 0.70
R2 0.36
R3 0.93
R4 0.64
R5 0.55
R6 0.75
R7 0.91
R8 0.47
R9 0.78

```

In [27]:

```
1 plot_scatterplot()
```



In [28]:

```

1  # audio block #2
2
3
4  # audio 2
5
6  audio = []
7
8  for x in range(11): # it should be between 1 and 11
9      valuex = AudioSegment.from_file("notes_/tC.mp3")
10     audio.append(valuex)
11  for i in range(11):
12     print(audio[i]) # at this stage, they are supposed to all give tC.mp3
13
14  for i in Robotx._registry:
15     if (i.betax == 0):
16         if (i.betay == 0.5):
17             valuex = AudioSegment.from_file("notes_/tC.mp3") # i.audio
18             audio.append(valuex)
19             print("tC")
20     if (i.betax > 0 and i.betax <= 0.17):
21         if (i.betay < 0.5):
22             valuex = AudioSegment.from_file("notes_/tB.mp3")
23             audio.append(valuex)
24             print("tB")
25         if (i.betay >= 0.5):
26             valuex = AudioSegment.from_file("notes_/tC#.mp3")
27             audio.append(valuex)
28             print("tC#")
29     if (i.betax > 0.17 and i.betax <= 0.3):
30         if (i.betay < 0.5): # if (R1.betay >= 0.17 and R1.betay < 0.3):
31             valuex = AudioSegment.from_file("notes_/tA#.mp3")
32             audio.append(valuex)
33             print("tA#")
34         if (i.betay >= 0.5):
35             valuex = AudioSegment.from_file("notes_/tD.mp3")
36             audio.append(valuex)
37             print("tD")
38     if (i.betax > 0.3 and i.betax <= 0.5):
39         if (i.betay < 0.5): # (R1.betay == 1):
40             valuex = AudioSegment.from_file("notes_/tD#.mp3")
41             audio.append(valuex)
42             print("tD#")
43         if (i.betay >= 0.5):
44             valuex = AudioSegment.from_file("notes_/tA.mp3")
45             audio.append(valuex)
46             print("tA")
47     if (i.betax > 0.5 and i.betax <= 0.64):
48         if (i.betay < 0.5):
49             valuex = AudioSegment.from_file("notes_/tE.mp3")
50             audio.append(valuex)
51             print("tE")
52         if (i.betay >= 0.5):
53             valuex = AudioSegment.from_file("notes_/tG#.mp3")
54             audio.append(valuex)
55             print("tG#")
56     if (i.betax > 0.64 and i.betax <= 0.84):
57         if (i.betay < 0.5):
58             valuex = AudioSegment.from_file("notes_/tF.mp3")
59             audio.append(valuex)

```

```

60         print("tF")
61     if (i.betay >= 0.5):
62         valuex = AudioSegment.from_file("notes_/tG.mp3")
63         audio.append(valuex)
64         print("tG")
65     if (i.betax > 0.84 and i.betax <= 1):
66         #if (R1.betay == 0.5):
67         valuex = AudioSegment.from_file("notes_/tF#.mp3")
68         audio.append(valuex)
69         print("tF#")
70
71
72
73 for i in Robotx._registry:
74     print(audio[i.position]) # at this stage, they are supposed to all give tC.
75
76
77
78
79 mix = []
80
81 for s in range(11): # it should be between 1 and 11
82     #values = (audio[s].overlay(audio[s+1])).overlay(audio[s+3])
83
84     # is there a more synthetic way to write this??
85     values = audio[s].overlay(audio[s+1])
86     values2 = values.overlay(audio[s+2])
87     values3 = values2.overlay(audio[s+3])
88     values4 = values3.overlay(audio[s+4])
89     values5 = values4.overlay(audio[s+5])
90     values6 = values5.overlay(audio[s+6])
91     values7 = values6.overlay(audio[s+7])
92     values8 = values7.overlay(audio[s+8])
93     values9 = values8.overlay(audio[s+9])
94     mix.append(values9)
95     print(mix[s])
96
97 mix[10].export("notes_/10_robot_sound/mixed_time_2.mp3", format='mp3') # export
98 play(mix[10])
99
100 # I'm trying to use the same code, but saving the file as another one.

```

```

<pydub.audio_segment.AudioSegment object at 0x7facd0131fa0>
<pydub.audio_segment.AudioSegment object at 0x7facd0131eb0>
<pydub.audio_segment.AudioSegment object at 0x7facd012aa60>
<pydub.audio_segment.AudioSegment object at 0x7facd012a5b0>
<pydub.audio_segment.AudioSegment object at 0x7facd012a1c0>
<pydub.audio_segment.AudioSegment object at 0x7facd012a640>
<pydub.audio_segment.AudioSegment object at 0x7facd012aa00>
<pydub.audio_segment.AudioSegment object at 0x7facd012ad30>
<pydub.audio_segment.AudioSegment object at 0x7facd012a670>
<pydub.audio_segment.AudioSegment object at 0x7facd012a1f0>
<pydub.audio_segment.AudioSegment object at 0x7facd012a310>
tA
tG#
tC#
tG
tA
tA
tG#
tG

```

```

tD
tG#
<pydub.audio_segment.AudioSegment object at 0x7facd0131eb0>
<pydub.audio_segment.AudioSegment object at 0x7facd012aa60>
<pydub.audio_segment.AudioSegment object at 0x7facd012a5b0>
<pydub.audio_segment.AudioSegment object at 0x7facd012a1c0>
<pydub.audio_segment.AudioSegment object at 0x7facd012a640>
<pydub.audio_segment.AudioSegment object at 0x7facd012aa00>
<pydub.audio_segment.AudioSegment object at 0x7facd012ad30>
<pydub.audio_segment.AudioSegment object at 0x7facd012a670>
<pydub.audio_segment.AudioSegment object at 0x7facd012a1f0>
<pydub.audio_segment.AudioSegment object at 0x7facd012a310>
<pydub.audio_segment.AudioSegment object at 0x7facd0131f10>
<pydub.audio_segment.AudioSegment object at 0x7facd012ab80>
<pydub.audio_segment.AudioSegment object at 0x7facd012a9d0>
<pydub.audio_segment.AudioSegment object at 0x7facd012ab20>
<pydub.audio_segment.AudioSegment object at 0x7facd012a820>
<pydub.audio_segment.AudioSegment object at 0x7facd012a2b0>
<pydub.audio_segment.AudioSegment object at 0x7facd012aca0>
<pydub.audio_segment.AudioSegment object at 0x7facd012af10>
<pydub.audio_segment.AudioSegment object at 0x7facd012ae20>
<pydub.audio_segment.AudioSegment object at 0x7facd012a7f0>
<pydub.audio_segment.AudioSegment object at 0x7facd012ad00>
Could not import the PyAudio C module '_portaudio'.

```

avplay version 12.3, Copyright (c) 2003-2018 the Libav developers
 built on Nov 2 2021 03:53:01 with Apple clang version 13.0.0 (clang
 -1300.0.29.3)
 Failed to set value '-hide_banner' for option 'autoexit'

In [29]:

```

1  # September 13
2
3  # Trying to understand how to update distances
4  # through the update of listX, listY
5
6  listX = list(k.betax for k in Robotx._registry)
7  listY = list(k.betay for k in Robotx._registry)
8
9  #print(listX)
10 #print(listY)
11
12 print(distance_A(T, listX, listY), distance_B(listX, listY))

```

0.3236987412319191 0.6516596152810638

In [30]:

```

1  # Quantum circuit construction -----

```

In [31]:

```

1  q = QuantumRegister(5, 'q') # qubits # changed to 9, formerly 15
2  m2 = ClassicalRegister(1, 'c1') # classical bits (separated is better)
3  m3 = ClassicalRegister(1, 'c2')
4  m4 = ClassicalRegister(1, 'c3')
5
6  qc3 = QuantumCircuit(q, m2, m3, m4) # to reach the target
7  qc4 = QuantumCircuit(q, m2, m3, m4) # to get back to the nest

```

In [32]:

```
1 # Which robot should enter the gate? -----
```

In [33]:

```
1 def print_formatted_vector(*args):
2     for vector in args:
3         print "[" + "".join(f"{val:.2f} " for val in vector).strip() + "]"
```

In [34]:

```
1 # in case of ties on delta score, the max() function outputs the first maximum
2 closest_robot = max(Robotx._registry, key=attrgetter('delta'))
3 print(f"Closest robot to the target: {closest_robot.name} {closest_robot.betax:.2f} {closest_robot.betay:.2f} {closest_robot.gamma:.2f} {closest_robot.delta:.2f}")
4
5 # and then it enters the gate
6 vector0 = [closest_robot.alphax, closest_robot.betax]
7 vector1 = [closest_robot.alphay, closest_robot.betay]
8 vector3 = [closest_robot.gamma, closest_robot.delta]
9
10 normalized_v0 = vector0/np.linalg.norm(vector0)
11 normalized_v1 = vector1/np.linalg.norm(vector1)
12 normalized_v3 = vector3/np.linalg.norm(vector3)
13
14 print_formatted_vector(vector0, vector1, vector3)
15 print_formatted_vector(normalized_v0, normalized_v1, normalized_v3)
```

Closest robot to the target: R3 0.74 0.94 0.93

```
[0.26 0.74]
[0.06 0.94]
[0.07 0.93]
[0.33 0.95]
[0.06 1.00]
[0.08 1.00]
```

In [35]:

```
1 # Setting up |q_0> -----
```

In [36]:

```
1 # direct initialization with amplitudes vector
2 qc3.initialize(normalized_v0, q[0])
3 qc3.initialize(normalized_v1, q[1])
4 qc3.initialize(normalized_v3, q[2])
```

Out[36]:

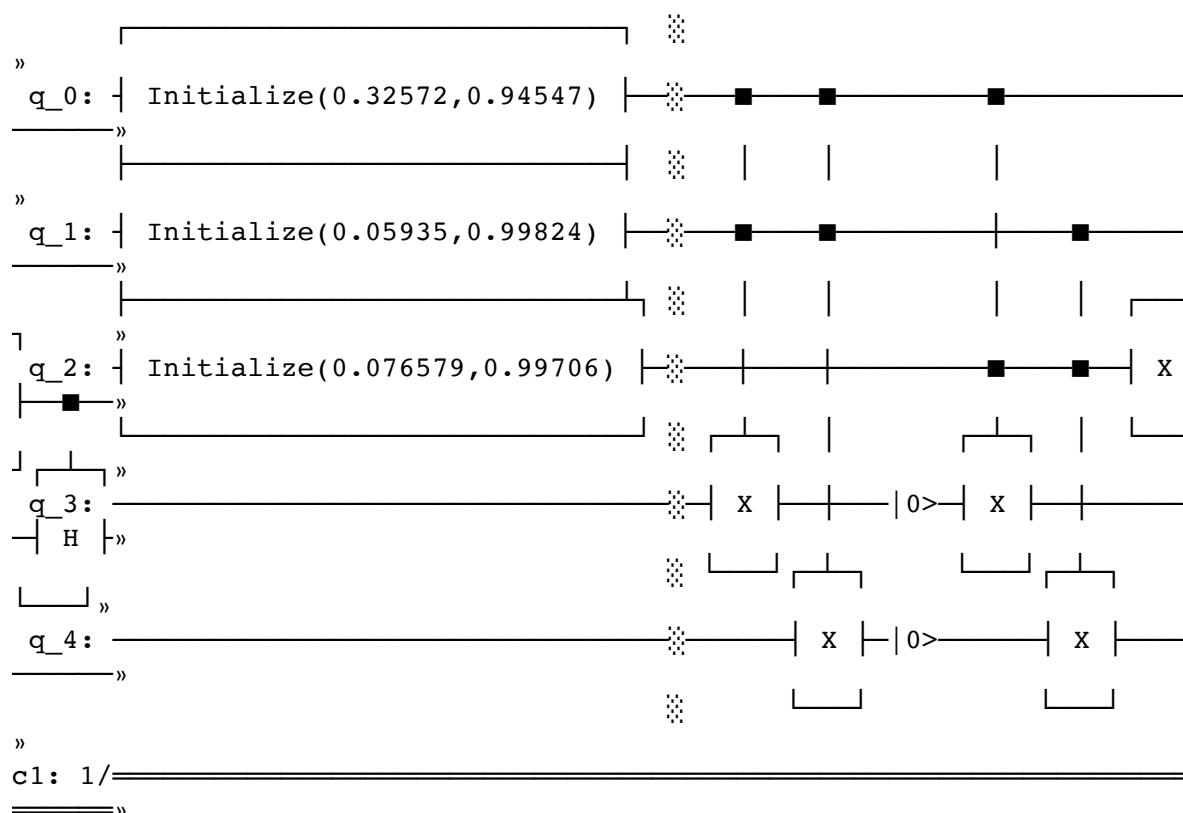
```
<qiskit.circuit.instructionset.InstructionSet at 0x7facb1b7a840>
```


In [37]:

```

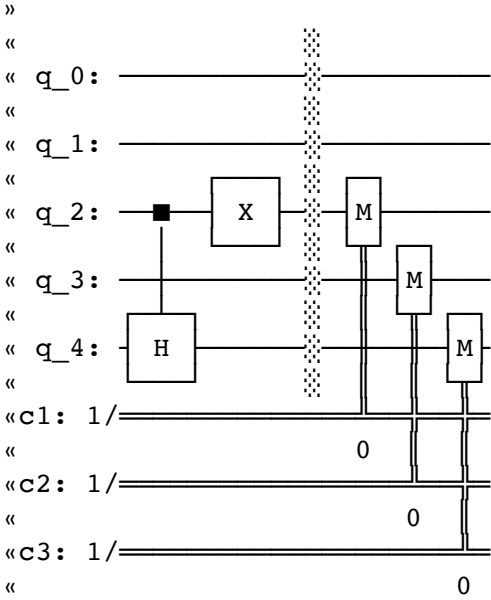
1  # this is the core code, and it is unchanged across time
2
3  qc3.barrier(q)
4  qc3.ccx(q[0],q[1],q[3])
5  qc3.ccx(q[0],q[1],q[4])
6
7  qc3.reset(q[3])
8  qc3.reset(q[4])
9
10 qc3.ccx(q[0],q[2],q[3])
11 qc3.ccx(q[1],q[2],q[4])
12
13 qc3.x(q[2])
14
15 qc3.ch(q[2],q[3])
16 qc3.ch(q[2],q[4])
17
18 qc3.x(q[2])
19
20 qc3.barrier(q)
21
22 # perform measurements and store them in classical bits
23
24 qc3.measure(q[2],m2[0])
25 qc3.measure(q[3],m3[0])
26 qc3.measure(q[4],m4[0])
27
28 # visualization of the circuit
29
30 # qc3.draw(fold=-1, output="mpl")
31 # plt.show();
32
33 print(qc3)

```



»
c2: 1/=====

»
c3: 1/=====



In [38]:

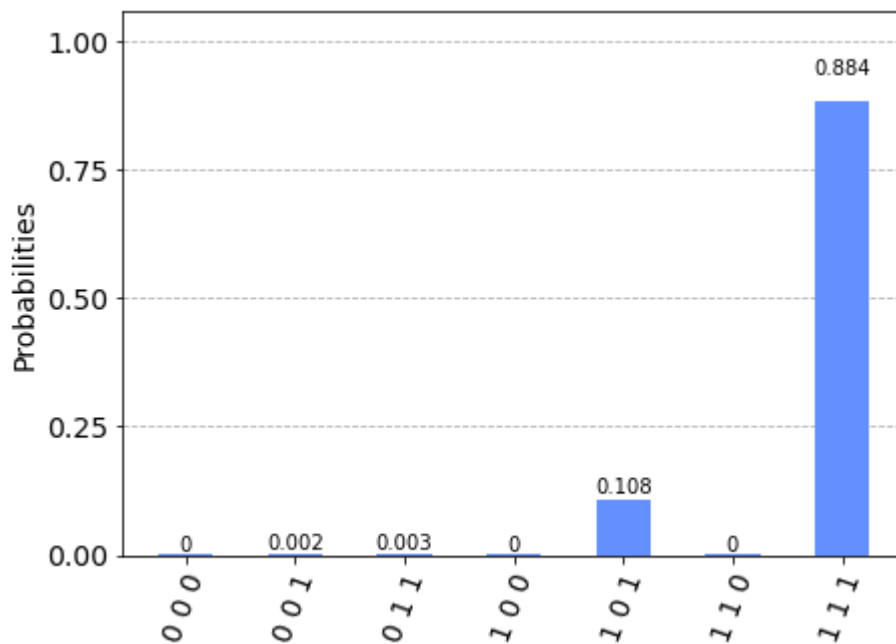
```

1 aer_sim = Aer.get_backend("aer_simulator")
2 transpiled_qc = transpile(qc3, aer_sim)
3 result = aer_sim.run(transpiled_qc).result()
4
5 counts = result.get_counts()
6 print("counts: ", counts)
7
8 plot_histogram(result.get_counts())

```

counts: {'0 0 0': 1, '0 0 1': 2, '1 0 0': 1, '1 1 0': 1, '1 1 1': 905, '0 1 1': 3, '1 0 1': 111}

Out[38]:



In [39]:

```

1 def eval_outcome(most_prob_dict, n_outcome):
2     mapped_weights0 = list(map(lambda res: int(res[n_outcome*2])*most_prob_dict[
3     return sum(mapped_weights0)/sum(most_prob_dict.values())

```

In [40]:

```

1 num_most_prob_states = 4
2
3 # https://docs.python.org/3/library/heapq.html:
4 #
5 # heapq.nlargest(n, iterable, key=None) returns a list with the n largest elements
6 #
7 most_prob_dict = dict(heapq.nlargest(num_most_prob_states, counts.items(), key=lambda x: x[1]))
8 print(f"{num_most_prob_states} most probable states: {most_prob_dict}")
9
10 outcome0, outcome1 = eval_outcome(most_prob_dict, 0), eval_outcome(most_prob_dict, 1)
11
12 print(f"outcome0: {outcome0:.2f}\noutcome1: {outcome1:.2f}")
13

```

4 most probable states: {'1 1 1': 905, '1 0 1': 111, '0 1 1': 3, '0 0 1': 2}
 outcome0: 1.00
 outcome1: 0.89

In [41]:

```

1 # Setting new positions after the gate -----

```

In [42]:

```

1 for i in Robotx._registry:
2     print(f"{i.name} {i.betax:.2f} {i.betay:.2f}")
3     if (i.delta != closest_robot.delta or all(i.delta == j.delta for j in Robotx._registry)):
4         # CHANGE: but taking into account the case where all robots have the same delta
5         # for z
6         #i.betaz = outcome0
7         # the lower this value, the closer the robot to the 0, the higher alpha
8         #i.alphaz = round(1 - i.betaz, 3)
9         # for y
10        i.betay = outcome0 # changed this
11        i.alphay = 1 - i.betay
12        # for x
13        i.betax = outcome1 # changed this
14        i.alphax = 1 - i.betax

```

R0 0.39 0.66
 R1 0.50 0.96
 R2 0.16 0.85
 R3 0.74 0.94
 R4 0.45 0.98
 R5 0.38 0.74
 R6 0.56 0.98
 R7 0.73 0.85
 R8 0.28 0.79
 R9 0.58 0.87

In [43]:

```
1 for k in Robotx._registry:
2     print(f"{k.name} {k.betax:.2f} {k.betay:.2f} {k.gamma:.2f} {k.position}")
```

```
R0 0.89 1.00 0.47 1
R1 0.89 1.00 0.30 2
R2 0.89 1.00 0.64 3
R3 0.74 0.94 0.07 4
R4 0.89 1.00 0.36 5
R5 0.89 1.00 0.45 6
R6 0.89 1.00 0.25 7
R7 0.89 1.00 0.09 8
R8 0.89 1.00 0.53 9
R9 0.89 1.00 0.22 10
```

In [44]:

```
1 # former rewards
2 for i in Robotx._registry:
3     print(f"before the gate: {i.name} {i.delta:.2f}")
```

```
before the gate: R0 0.53
before the gate: R1 0.70
before the gate: R2 0.36
before the gate: R3 0.93
before the gate: R4 0.64
before the gate: R5 0.55
before the gate: R6 0.75
before the gate: R7 0.91
before the gate: R8 0.47
before the gate: R9 0.78
```

In [45]:

```
1 # new rewards
2 for i in Robotx._registry: # recalculate the rewards
3     i.delta = reward(T, i.betax, i.betay)
4     i.gamma = 1 - i.delta
5     print(f"after the gate: {i.name} {i.delta:.2f}")
```

```
after the gate: R0 0.87
after the gate: R1 0.87
after the gate: R2 0.87
after the gate: R3 0.93
after the gate: R4 0.87
after the gate: R5 0.87
after the gate: R6 0.87
after the gate: R7 0.87
after the gate: R8 0.87
after the gate: R9 0.87
```

In [46]:

```

1  # audio block #3
2
3  # audio 3
4
5  # we can define "audio" as an attribute... no, better not.
6
7  audio = []
8
9  for x in range(11): # it should be between 1 and 11
10     valuex = AudioSegment.from_file("notes_/tC.mp3")
11     audio.append(valuex)
12  for i in range(11):
13     print(audio[i]) # at this stage, they are supposed to all give tC.mp3
14
15  for i in Robotx._registry:
16     if (i.betax == 0):
17         if (i.betay == 0.5):
18             valuex = AudioSegment.from_file("notes_/tC.mp3") # i.audio
19             audio.append(valuex)
20             print("tC")
21     if (i.betax > 0 and i.betax <= 0.17):
22         if (i.betay < 0.5):
23             valuex = AudioSegment.from_file("notes_/tB.mp3")
24             audio.append(valuex)
25             print("tB")
26         if (i.betay >= 0.5):
27             valuex = AudioSegment.from_file("notes_/tC#.mp3")
28             audio.append(valuex)
29             print("tC#")
30     if (i.betax > 0.17 and i.betax <= 0.3):
31         if (i.betay < 0.5): # if (R1.betay >= 0.17 and R1.betay < 0.3):
32             valuex = AudioSegment.from_file("notes_/tA#.mp3")
33             audio.append(valuex)
34             print("tA#")
35         if (i.betay >= 0.5):
36             valuex = AudioSegment.from_file("notes_/tD.mp3")
37             audio.append(valuex)
38             print("tD")
39     if (i.betax > 0.3 and i.betax <= 0.5):
40         if (i.betay < 0.5): # (R1.betay == 1):
41             valuex = AudioSegment.from_file("notes_/tD#.mp3")
42             audio.append(valuex)
43             print("tD#")
44         if (i.betay >= 0.5):
45             valuex = AudioSegment.from_file("notes_/tA.mp3")
46             audio.append(valuex)
47             print("tA")
48     if (i.betax > 0.5 and i.betax <= 0.64):
49         if (i.betay < 0.5):
50             valuex = AudioSegment.from_file("notes_/tE.mp3")
51             audio.append(valuex)
52             print("tE")
53         if (i.betay >= 0.5):
54             valuex = AudioSegment.from_file("notes_/tG#.mp3")
55             audio.append(valuex)
56             print("tG#")
57     if (i.betax > 0.64 and i.betax <= 0.84):
58         if (i.betay < 0.5):
59             valuex = AudioSegment.from_file("notes_/tF.mp3")

```

```

60         audio.append(valuex)
61         print("tF")
62     if (i.betay >= 0.5):
63         valuex = AudioSegment.from_file("notes_/tG.mp3")
64         audio.append(valuex)
65         print("tG")
66     if (i.betax > 0.84 and i.betax <= 1):
67         #if (R1.betay == 0.5):
68         valuex = AudioSegment.from_file("notes_/tF#.mp3")
69         audio.append(valuex)
70         print("tF#")
71
72
73
74 for i in Robotx._registry:
75     print(audio[i.position]) # at this stage, they are supposed to all give tC.
76
77
78
79
80 mix = []
81
82 for s in range(11): # it should be between 1 and 11
83     #values = (audio[s].overlay(audio[s+1])).overlay(audio[s+3])
84
85     # is there a more synthetic way to write this??
86     values = audio[s].overlay(audio[s+1])
87     values2 = values.overlay(audio[s+2])
88     values3 = values2.overlay(audio[s+3])
89     values4 = values3.overlay(audio[s+4])
90     values5 = values4.overlay(audio[s+5])
91     values6 = values5.overlay(audio[s+6])
92     values7 = values6.overlay(audio[s+7])
93     values8 = values7.overlay(audio[s+8])
94     values9 = values8.overlay(audio[s+9])
95     mix.append(values9)
96     print(mix[s])
97
98 mix[10].export("notes_/10_robot_sound/mixed_time_3.mp3", format='mp3') # export
99 play(mix[10])
100

```

```

<pydub.audio_segment.AudioSegment object at 0x7faca0336b50>
<pydub.audio_segment.AudioSegment object at 0x7facb0834d60>
<pydub.audio_segment.AudioSegment object at 0x7facb083fc70>
<pydub.audio_segment.AudioSegment object at 0x7facb083f580>
<pydub.audio_segment.AudioSegment object at 0x7facb083f850>
<pydub.audio_segment.AudioSegment object at 0x7facb083fbb0>
<pydub.audio_segment.AudioSegment object at 0x7facb083fa00>
<pydub.audio_segment.AudioSegment object at 0x7facb083f430>
<pydub.audio_segment.AudioSegment object at 0x7facb083fb50>
<pydub.audio_segment.AudioSegment object at 0x7facb083f8e0>
<pydub.audio_segment.AudioSegment object at 0x7facd012a580>
tF#
tF#
tF#
tG
tF#
tF#
tF#
tF#

```

```
tF#
tF#
<pydub.audio_segment.AudioSegment object at 0x7facb0834d60>
<pydub.audio_segment.AudioSegment object at 0x7facb083fc70>
<pydub.audio_segment.AudioSegment object at 0x7facb083f580>
<pydub.audio_segment.AudioSegment object at 0x7facb083f850>
<pydub.audio_segment.AudioSegment object at 0x7facb083fbb0>
<pydub.audio_segment.AudioSegment object at 0x7facb083fa00>
<pydub.audio_segment.AudioSegment object at 0x7facb083f430>
<pydub.audio_segment.AudioSegment object at 0x7facb083fb50>
<pydub.audio_segment.AudioSegment object at 0x7facb083f8e0>
<pydub.audio_segment.AudioSegment object at 0x7facd012a580>
<pydub.audio_segment.AudioSegment object at 0x7fac949f4f40>
<pydub.audio_segment.AudioSegment object at 0x7fac949f4370>
<pydub.audio_segment.AudioSegment object at 0x7fac949f4e20>
<pydub.audio_segment.AudioSegment object at 0x7fac949f4f70>
<pydub.audio_segment.AudioSegment object at 0x7facd0131eb0>
<pydub.audio_segment.AudioSegment object at 0x7facd012ad30>
<pydub.audio_segment.AudioSegment object at 0x7fac949f49d0>
<pydub.audio_segment.AudioSegment object at 0x7fac949f4a30>
<pydub.audio_segment.AudioSegment object at 0x7facb0847970>
<pydub.audio_segment.AudioSegment object at 0x7fac949f4d90>
<pydub.audio_segment.AudioSegment object at 0x7fac93781610>
Could not import the PyAudio C module '_portaudio'.
```

```
avplay version 12.3, Copyright (c) 2003-2018 the Libav developers
  built on Nov  2 2021 03:53:01 with Apple clang version 13.0.0 (clang
-1300.0.29.3)
Failed to set value '-hide_banner' for option 'autoexit'
```

In [47]:

```
1 # Reach the most successful robot -----
```


In [48]:

```

1  # not for now
2
3  for i in Robotx._registry: # recalculate the rewards
4      i.delta = reward(T, i.betax, i.betay)
5      i.gamma = (1 - i.delta, 3)
6
7  max_attr_ = max(Robotx._registry, key=attrgetter('delta'))
8  print(f"max_attr_.delta: {max_attr_.delta:.2f}")
9
10 for i in Robotx._registry:
11     if (i.delta == max_attr_.delta):
12         print(f"Most successful robot: {i.name} {i.betax:.2f} {i.betay:.2f} {i.c
13
14 for j in Robotx._registry:
15     # to get other robots following it:
16     if (j != max_attr_): # changed here
17         flag = True
18         while flag:
19             flag = False
20             j.alphax = max_attr_.alphax + np.random.uniform(0,0.01)
21             j.betax = 1 - j.alphax
22             j.alphay = max_attr_.alphay + np.random.uniform(0,0.01)
23             j.betay = 1 - j.alphay
24             if (j.betax - 0.x <= 0.2 and j.betay - 0.y <= 0.2):
25                 flag = True
26
27 # recalculate the rewards here:
28
29 for k in Robotx._registry: # recalculate the rewards
30     k.delta = reward(T, k.betax, k.betay)
31     k.gamma = 1 - k.delta
32     print(f"{k.name} {k.delta:.2f}")

```

max_attr_.delta: 0.93

Most successful robot: R3 0.74 0.94 0.93

R0 0.93

R1 0.93

R2 0.93

R3 0.93

R4 0.93

R5 0.93

R6 0.93

R7 0.92

R8 0.93

R9 0.93

In [49]:

```
1 for i in Robotx._registry: # recalculate the rewards
2     i.delta = reward(T, i.betax, i.betay)
3     i.gamma = 1 - i.delta
4     print(f"{i.name} {i.delta:.2f}")
```

R0 0.93
R1 0.93
R2 0.93
R3 0.93
R4 0.93
R5 0.93
R6 0.93
R7 0.92
R8 0.93
R9 0.93

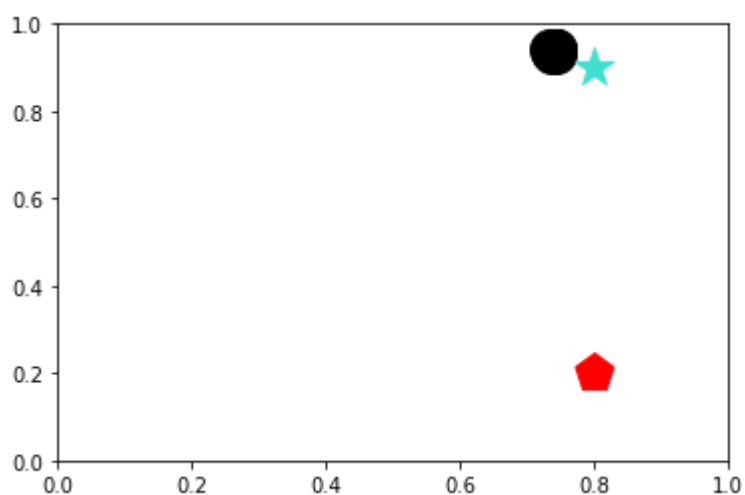
In [50]:

```
1 for k in Robotx._registry:
2     print(f"{k.name} {k.betax:.2f} {k.betay:.2f} {k.gamma:.2f} {k.delta:.2f} {k.}
```

R0 0.74 0.94 0.07 0.93 1
R1 0.74 0.94 0.07 0.93 2
R2 0.74 0.93 0.07 0.93 3
R3 0.74 0.94 0.07 0.93 4
R4 0.74 0.94 0.07 0.93 5
R5 0.74 0.93 0.07 0.93 6
R6 0.74 0.94 0.07 0.93 7
R7 0.74 0.94 0.08 0.92 8
R8 0.74 0.94 0.07 0.93 9
R9 0.74 0.94 0.07 0.93 10

In [51]:

```
1 plot_scatterplot()
```



In [52]:

```
1  # September 13
2
3  # Trying to understand how to update distances
4  # through the update of listX, listY
5
6  listX = list(k.betax for k in Robotx._registry)
7  listY = list(k.betay for k in Robotx._registry)
8
9  #print(listX)
10 #print(listY)
11
12 print(distance_A(T, listX, listY), distance_B(listX, listY))
```

0.07138862667781866 0.01221562201155434

In [53]:

```

1  # audio block #4
2
3  # audio 4
4
5  # we can define "audio" as an attribute... no, better not.
6
7  audio = []
8
9  for x in range(11): # it should be between 1 and 11
10     valuex = AudioSegment.from_file("notes_/tC.mp3")
11     audio.append(valuex)
12  for i in range(11):
13     print(audio[i]) # at this stage, they are supposed to all give tC.mp3
14
15  for i in Robotx._registry:
16     if (i.betax == 0):
17         if (i.betay == 0.5):
18             valuex = AudioSegment.from_file("notes_/tC.mp3") # i.audio
19             audio.append(valuex)
20             print("tC")
21     if (i.betax > 0 and i.betax <= 0.17):
22         if (i.betay < 0.5):
23             valuex = AudioSegment.from_file("notes_/tB.mp3")
24             audio.append(valuex)
25             print("tB")
26         if (i.betay >= 0.5):
27             valuex = AudioSegment.from_file("notes_/tC#.mp3")
28             audio.append(valuex)
29             print("tC#")
30     if (i.betax > 0.17 and i.betax <= 0.3):
31         if (i.betay < 0.5): # if (R1.betay >= 0.17 and R1.betay < 0.3):
32             valuex = AudioSegment.from_file("notes_/tA#.mp3")
33             audio.append(valuex)
34             print("tA#")
35         if (i.betay >= 0.5):
36             valuex = AudioSegment.from_file("notes_/tD.mp3")
37             audio.append(valuex)
38             print("tD")
39     if (i.betax > 0.3 and i.betax <= 0.5):
40         if (i.betay < 0.5): # (R1.betay == 1):
41             valuex = AudioSegment.from_file("notes_/tD#.mp3")
42             audio.append(valuex)
43             print("tD#")
44         if (i.betay >= 0.5):
45             valuex = AudioSegment.from_file("notes_/tA.mp3")
46             audio.append(valuex)
47             print("tA")
48     if (i.betax > 0.5 and i.betax <= 0.64):
49         if (i.betay < 0.5):
50             valuex = AudioSegment.from_file("notes_/tE.mp3")
51             audio.append(valuex)
52             print("tE")
53         if (i.betay >= 0.5):
54             valuex = AudioSegment.from_file("notes_/tG#.mp3")
55             audio.append(valuex)
56             print("tG#")
57     if (i.betax > 0.64 and i.betax <= 0.84):
58         if (i.betay < 0.5):
59             valuex = AudioSegment.from_file("notes_/tF.mp3")

```

```

60         audio.append(valuex)
61         print("tF")
62     if (i.betay >= 0.5):
63         valuex = AudioSegment.from_file("notes_/tG.mp3")
64         audio.append(valuex)
65         print("tG")
66     if (i.betax > 0.84 and i.betax <= 1):
67         #if (R1.betay == 0.5):
68         valuex = AudioSegment.from_file("notes_/tF#.mp3")
69         audio.append(valuex)
70         print("tF#")
71
72
73
74 for i in Robotx._registry:
75     print(audio[i.position]) # at this stage, they are supposed to all give tC.
76
77
78
79
80 mix = []
81
82 for s in range(11): # it should be between 1 and 11
83     #values = (audio[s].overlay(audio[s+1])).overlay(audio[s+3])
84
85     # is there a more synthetic way to write this??
86     values = audio[s].overlay(audio[s+1])
87     values2 = values.overlay(audio[s+2])
88     values3 = values2.overlay(audio[s+3])
89     values4 = values3.overlay(audio[s+4])
90     values5 = values4.overlay(audio[s+5])
91     values6 = values5.overlay(audio[s+6])
92     values7 = values6.overlay(audio[s+7])
93     values8 = values7.overlay(audio[s+8])
94     values9 = values8.overlay(audio[s+9])
95     mix.append(values9)
96     print(mix[s])
97
98 mix[10].export("notes_/10_robot_sound/mixed_time_4.mp3", format='mp3') # export
99 play(mix[10])
100

```

```

<pydub.audio_segment.AudioSegment object at 0x7fac94a12f10>
<pydub.audio_segment.AudioSegment object at 0x7facc1576f10>
<pydub.audio_segment.AudioSegment object at 0x7facc1576e80>
<pydub.audio_segment.AudioSegment object at 0x7facc1576e50>
<pydub.audio_segment.AudioSegment object at 0x7facc1576bb0>
<pydub.audio_segment.AudioSegment object at 0x7fac8003b850>
<pydub.audio_segment.AudioSegment object at 0x7facc1576b50>
<pydub.audio_segment.AudioSegment object at 0x7facd0131fa0>
<pydub.audio_segment.AudioSegment object at 0x7fac8003b9a0>
<pydub.audio_segment.AudioSegment object at 0x7fac8003b940>
<pydub.audio_segment.AudioSegment object at 0x7facb083fc70>
tG
tG
tG
tG
tG
tG
tG
tG

```

tG

tG

```
<pydub.audio_segment.AudioSegment object at 0x7facc1576f10>
<pydub.audio_segment.AudioSegment object at 0x7facc1576e80>
<pydub.audio_segment.AudioSegment object at 0x7facc1576e50>
<pydub.audio_segment.AudioSegment object at 0x7facc1576bb0>
<pydub.audio_segment.AudioSegment object at 0x7fac8003b850>
<pydub.audio_segment.AudioSegment object at 0x7facc1576b50>
<pydub.audio_segment.AudioSegment object at 0x7facd0131fa0>
<pydub.audio_segment.AudioSegment object at 0x7fac8003b9a0>
<pydub.audio_segment.AudioSegment object at 0x7fac8003b940>
<pydub.audio_segment.AudioSegment object at 0x7facb083fc70>
<pydub.audio_segment.AudioSegment object at 0x7facc1589e20>
<pydub.audio_segment.AudioSegment object at 0x7facc1589bb0>
<pydub.audio_segment.AudioSegment object at 0x7facc1589ee0>
<pydub.audio_segment.AudioSegment object at 0x7facc1589df0>
<pydub.audio_segment.AudioSegment object at 0x7facb083f580>
<pydub.audio_segment.AudioSegment object at 0x7fac93781310>
<pydub.audio_segment.AudioSegment object at 0x7fac93781f10>
<pydub.audio_segment.AudioSegment object at 0x7fac93781b20>
<pydub.audio_segment.AudioSegment object at 0x7facb083fa00>
<pydub.audio_segment.AudioSegment object at 0x7facb083f8e0>
<pydub.audio_segment.AudioSegment object at 0x7facb083fb50>
Could not import the PyAudio C module '_portaudio'.
```

avplay version 12.3, Copyright (c) 2003-2018 the Libav developers
built on Nov 2 2021 03:53:01 with Apple clang version 13.0.0 (clang
-1300.0.29.3)
Failed to set value '-hide_banner' for option 'autoexit'

In []:

1

In []:

1

In []:

1

In []:

1

In [54]:

```
1 # September 13
2
3 # Trying to understand how to update distances
4 # through the update of listX, listY
5
6 listX = list(k.betax for k in Robotx._registry)
7 listY = list(k.betay for k in Robotx._registry)
8
9 #print(listX)
10 #print(listY)
11
12 print(distance_A(T, listX, listY), distance_B(listX, listY))
13
14 print("Euclidean", Euclidean_distance(T, listX, listY))
15 print("Manhattan", Manhattan_distance(T, listX, listY))
16 print("Cosine", Cosine_distance(T, listX, listY))
17
```

0.07138862667781866 0.01221562201155434

Euclidean 0.07138862667781866

Manhattan 0.09871936665470737

Cosine 0.0017986511543843786

In []:

1