In [1]:

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, Aer, trar
import numpy as np
from qiskit.visualization import plot_histogram
from qiskit import *
import random
import matplotlib.pyplot as plt
from operator import attrgetter
import matplotlib.pyplot as plt
import heapq
from operator import itemgetter
from pydub import AudioSegment # for audio
from pydub.playback import play # for audio

import statistics # added for the mean computation
from collections import defaultdict # added to compare elements of the list
from itertools import tee # to allow pairwise comparisons
from scipy.spatial.distance import cosine # to compute cosine distance
```

In [ ]:

```

```

In [2]:

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

In [3]:

```
class Target:
    def __init__(self,name,x,y): # no indetermination in the target's position
        self.name = name
        self.x = x
        self.y = y
```

In [4]:

```
T = Target("T", 0.9, 0.5) # deep in the ocean

# for getting back to the beginning
T2 = Target("T2", 0.2, 0.5) # back to the ship
```

In [5]:

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

```

In [6]:

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

In [7]:

```
class Obstacle: # Just a point for now
    def __init__(self,name,x,y):
        self.name = name
        self.x = x
        self.y = y
```

In [8]:

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

In [9]:

```
# Robots ------------------------------------------------------------------
```

In [10]:

```
class Robotx(object):
    _registry = []

    def __init__(self, name, alphax, betax, alphay, betay, gamma, delta, positic
        self._registry.append(self)
        self.name = name
        self.alphax = alphax
        self.betax = betax
        self.alphay = alphay
        self.betay = betay
        delta = reward(T, betax, betay)
        gamma = 1 - delta
        self.gamma = gamma
        self.delta = delta
        self.position = position # new -- I need it for sound
```

In [ ]:

```

```

In [11]:

```
# arbitrary number of robots that, at the start, are uniformly distributed in th
# centered in starting_cluster_coord
#
num_of_robots = 10
radius = 0.1
# starting_cluster_coord = (0.6, 0.6)
starting_cluster_coord = (0.2, 0.5)

a_x, a_y = 1-starting_cluster_coord[0]-radius, 1-starting_cluster_coord[0]+radiu
b_x, b_y = 1-starting_cluster_coord[1]-radius, 1-starting_cluster_coord[1]+radiu

for i in range(num_of_robots):
    x = random.uniform(a_x,a_y)
    y = random.uniform(b_x,b_y)
    Robotx('R'+str(i), x, 1-x, y, 1-y, 1 - reward(T, 1-x, 1-y), reward(T, 1-x, 1
```

In [12]:

```
# note: values are stored with full precision, rounding is done only on visualiz

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

```
R0 0.24 0.50 0.66 0.34 1
R1 0.22 0.42 0.68 0.32 2
R2 0.25 0.47 0.65 0.35 3
R3 0.22 0.53 0.68 0.32 4
R4 0.17 0.55 0.73 0.27 5
R5 0.25 0.52 0.65 0.35 6
R6 0.28 0.52 0.62 0.38 7
R7 0.21 0.60 0.70 0.30 8
R8 0.13 0.49 0.77 0.23 9
R9 0.19 0.58 0.72 0.28 10
```

In [13]:

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

```
R0 0.34
R1 0.32
R2 0.35
R3 0.32
R4 0.27
R5 0.35
R6 0.38
R7 0.30
R8 0.23
R9 0.28
```

In [14]:

```
#for k in Robotx._registry:
    #print(statistics.mean(k.betax))
    #k.betax + (k+1).betax

# explanation here:
# https://stackoverflow.com/questions/10879867/sum-average-an-attribute-of-a-lis

# September 13, 2022


# sum_x = sum(k.betax for k in Robotx._registry)
# sum_y = sum(k.betay for k in Robotx._registry)

listX = list(k.betax for k in Robotx._registry)
listY = list(k.betay for k in Robotx._registry)

def distance_A(T, listX, listY):
    sum_x = sum(listX)
    sum_y = sum(listY)
    center_x = sum_x/num_of_robots
    center_y = sum_y/num_of_robots
    return ((T.x - center_x)**2 + (T.y - center_y)**2)**0.5

print("distance_A", distance_A(T, listX, listY))

def Euclidean_distance(T, listX, listY): # the same as distance_A
    sum_x = sum(listX)
    sum_y = sum(listY)
    center_x = sum_x/num_of_robots
    center_y = sum_y/num_of_robots
    return ((T.x - center_x)**2 + (T.y - center_y)**2)**0.5

print("Euclidean", Euclidean_distance(T, listX, listY))

def Manhattan_distance(T, listX, listY):
    sum_x = sum(listX)
    sum_y = sum(listY)
    center_x = sum_x/num_of_robots
    center_y = sum_y/num_of_robots
    return (abs(T.x - center_x) + abs(T.y - center_y))

print("Manhattan", Manhattan_distance(T, listX, listY))

def Cosine_distance(T, listX, listY):
    sum_x = sum(listX)
    sum_y = sum(listY)
    center_x = sum_x/num_of_robots
    center_y = sum_y/num_of_robots
    array_1 = np.array([center_x, T.x])
    array_2 = np.array([center_y, T.y])
    return cosine(array_1, array_2)

print("Cosine", Cosine_distance(T, listX, listY))
```

```
distance_A 0.6841873579013458
Euclidean 0.6841873579013458
Manhattan 0.7024119778828791
Cosine 0.15698106369585008
```

In [15]:

```
# September 13

# function taken from https://stackoverflow.com/questions/31044711/method-to-get

def pairwise(iterable):
    "s -> (s0,s1), (s1,s2), (s2, s3), ..."
    a, b = tee(iterable)
    next(b, None)
    return zip(a, b) # not izip
```

In [16]:

```
# September 13

# method adapted from https://stackoverflow.com/questions/31044711/method-to-get-t

# with the Euclidean distance rather than the simple difference

def distance_B(listX, listY):
    return (max((b - a)**2 for (a,b) in pairwise(listX)) + max((c - d)**2 for (c,d)
```

In [17]:

```
# September 13

# classic one! "within-cluster distance"
# distance between the swarm barycenter (as a centroid) and each element

# distance_A can be seen as a particular case of between-cluster distance (dista
# where the second cluster is actually only a point (the target)

# "within cluster sum of squares"

def distance_C(listX, listY):
    sum_x = sum(listX)
    sum_y = sum(listY)
    center_x = sum_x/num_of_robots
    center_y = sum_y/num_of_robots
    return (max((center_x - a)**2 for a in listX) + max((center_y - b)**2 for b

print(distance_C(listX, listY))
```

```
0.13416068899847813
```

In [18]:

```
# September 13

print(distance_A(T, listX, listY), distance_B(listX, listY), distance_C(listX, l
```

```
0.6841873579013458 0.13321618576736521 0.13416068899847813
```

In [ ]:

```

```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [19]:

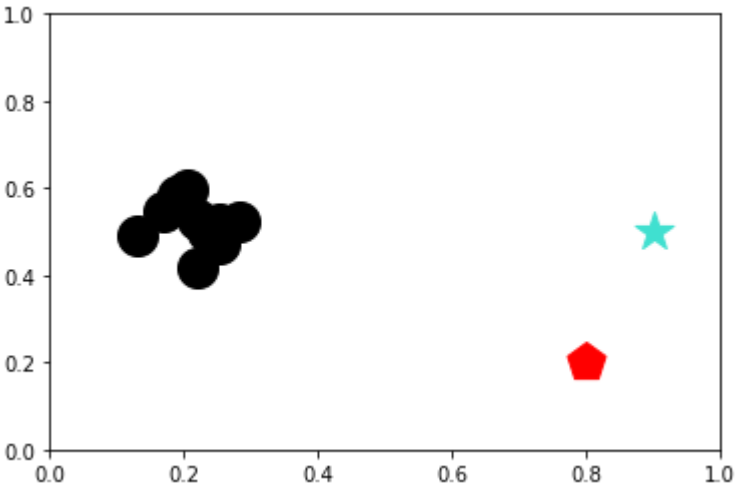
```
def plot_scatterplot():
    for i in  Robotx._registry:
        plt.scatter(i.betax, i.betay, s = 400, marker = 'o', color = 'black')

    plt.scatter(T.x, T.y, s = 400, marker = '*', color = 'turquoise')
    plt.scatter(O.x, O.y, s = 400, marker = 'p', color = 'red')

    plt.axis([0, 1, 0, 1])

    plt.show()

plot_scatterplot()
```



In [20]:

```
# initialization of sound parameters


# we need 'append' to create such a list!

l = []
for x in range(11):
    value = AudioSegment.from_file("notes_/tC.mp3")
    l.append(value)
for i in range(11):
    print(l[i])

for k in Robotx._registry:
    print(k.position)

for k in Robotx._registry:
    print(l[k.position])
```

```
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05250>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d058b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1ce2fd0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1ce2910>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d052b0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0067f40>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0067eb0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0067df0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801be0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801c70>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801430>
1
2
3
4
5
6
7
8
9
10
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d058b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1ce2fd0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1ce2910>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d052b0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0067f40>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0067eb0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0067df0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801be0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801c70>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801430>
```

In [21]:

```
# audio block #1

# audio 1

# we can define "audio" as an attribute... no, better not.

audio = []

for x in range(11): # it should be between 1 and 11
    valuex = AudioSegment.from_file("notes_/tC.mp3")
    audio.append(valuex)
for i in range(11):
    print(audio[i]) # at this stage, they are supposed to all give tC.mp3

for i in Robotx._registry:
    if (i.betax == 0):
        if (i.betay == 0.5):
            valuex = AudioSegment.from_file("notes_/tC.mp3") # i.audio
            audio.append(valuex)
            print("tC")
    if (i.betax > 0 and i.betax <= 0.17):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tB.mp3")
            audio.append(valuex)
            print("tB")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tC#.mp3")
            audio.append(valuex)
            print("tC#")
    if (i.betax > 0.17 and i.betax <= 0.3):
        if (i.betay < 0.5): # if (R1.betay >= 0.17 and R1.betay < 0.3):
            valuex = AudioSegment.from_file("notes_/tA#.mp3")
            audio.append(valuex)
            print("tA#")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tD.mp3")
            audio.append(valuex)
            print("tD")
    if (i.betax > 0.3 and i.betax <= 0.5):
        if (i.betay < 0.5): # (R1.betay == 1):
            valuex = AudioSegment.from_file("notes_/tD#.mp3")
            audio.append(valuex)
            print("tD#")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tA.mp3")
            audio.append(valuex)
            print("tA")
    if (i.betax > 0.5 and i.betax <= 0.64):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tE.mp3")
            audio.append(valuex)
            print("tE")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tG#.mp3")
            audio.append(valuex)
            print("tG#")
    if (i.betax > 0.64 and i.betax <= 0.84):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tF.mp3")
```

```
            audio.append(valuex)
            print("tF")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tG.mp3")
            audio.append(valuex)
            print("tG")
    if (i.betax > 0.84 and i.betax <= 1):
        #if (R1.betay == 0.5):
        valuex = AudioSegment.from_file("notes_/tF#.mp3")
        audio.append(valuex)
        print("tF#")



for i in Robotx._registry:
    print(audio[i.position]) # at this stage, they are supposed to all give tC.




mix = []

for s in range(11): # it should be between 1 and 11
    #values = (audio[s].overlay(audio[s+1])).overlay(audio[s+3])

    # is there a more synthetic way to write this??
    values = audio[s].overlay(audio[s+1])
    values2 = values.overlay(audio[s+2])
    values3 = values2.overlay(audio[s+3])
    values4 = values3.overlay(audio[s+4])
    values5 = values4.overlay(audio[s+5])
    values6 = values5.overlay(audio[s+6])
    values7 = values6.overlay(audio[s+7])
    values8 = values7.overlay(audio[s+8])
    values9 = values8.overlay(audio[s+9])
    mix.append(values9)
    print(mix[s])

mix[10].export("notes_/10_robot_sound/mixed_time_1.mp3", format='mp3') # export
play(mix[10])

```

```
<pydub.audio_segment.AudioSegment object at 0x7f8cd1ce2fa0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d058e0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05df0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05b80>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38e58b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38e5a90>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38e55b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38e5f70>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38e5bb0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38010d0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca00722b0>
tD
tA#
tA#
tD
tC#
tD
tD
tD
```

```
tB
tD
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d058e0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05df0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05b80>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38e58b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38e5a90>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38e55b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38e5f70>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38e5bb0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38010d0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca00722b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801640>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38015b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801e80>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801100>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38018b0>
<pydub.audio_segment.AudioSegment object at 0x7f8ce093bc10>
<pydub.audio_segment.AudioSegment object at 0x7f8ce08b37c0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3853340>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801700>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801df0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f2e0>
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]:

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

```
R0 0.34 achtung!
R1 0.32 achtung!
R2 0.35 achtung!
R3 0.32 achtung!
R4 0.27 achtung!
R5 0.35 achtung!
R6 0.38 achtung!
R7 0.30 achtung!
R8 0.23 achtung!
R9 0.28 achtung!
```

In [ ]:

```

```

In [23]:

```
# Reshuffling ----------------------------------------------------------------
```

In [ ]:

```

```

In [24]:

```
# I'm adding this one as the only non-quantum thing:

result = all(i.delta < 0.8 for i in Robotx._registry)

# Printing result
print("Do all the robots have a reward lower than 0.8? : " + str(result))

# if True: reshuffle positions
# if False: do nothing

if result == True:
    flag = True
    while flag:
        flag = False
        for i in Robotx._registry:
            i.alphax = np.random.uniform(0,0.9)
            i.betax = 1 - i.alphax
            i.alphay = np.random.uniform(0,0.9)
            i.betay = 1 - i.alphay
            if (i.betax - O.x <= 0.2 and i.betay - O.y <= 0.2 <= 0.2):
                flag = True
```

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

In [25]:

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

```
R0 0.91 0.63 0.66 1
R1 0.31 0.40 0.68 2
R2 0.72 0.67 0.65 3
R3 0.97 0.74 0.68 4
R4 0.70 0.65 0.73 5
R5 0.87 0.65 0.65 6
R6 0.71 0.57 0.62 7
R7 0.23 0.47 0.70 8
R8 0.81 0.64 0.77 9
R9 0.96 0.75 0.72 10
```

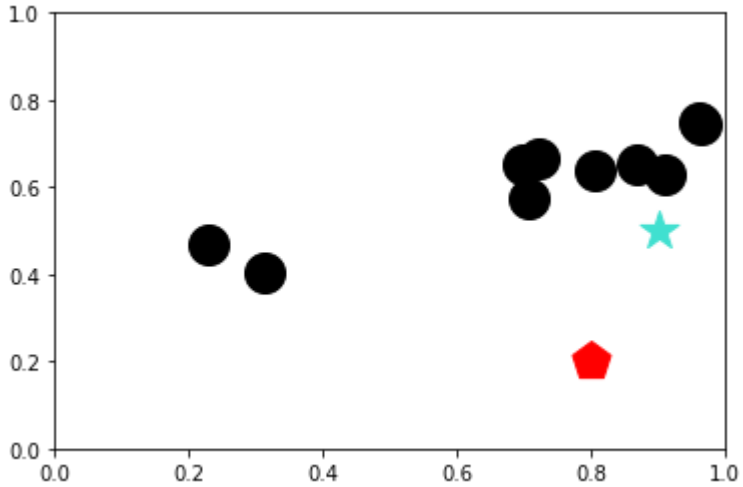
In [26]:

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

```
R0 0.87
R1 0.40
R2 0.76
R3 0.75
R4 0.75
R5 0.85
R6 0.79
R7 0.33
R8 0.83
R9 0.74
```

In [27]:

```
1 plot_scatterplot()
```



In [28]:

```
# audio block #2


# audio 2

audio = []

for x in range(11): # it should be between 1 and 11
    valuex = AudioSegment.from_file("notes_/tC.mp3")
    audio.append(valuex)
for i in range(11):
    print(audio[i]) # at this stage, they are supposed to all give tC.mp3

for i in Robotx._registry:
    if (i.betax == 0):
        if (i.betay == 0.5):
            valuex = AudioSegment.from_file("notes_/tC.mp3") # i.audio
            audio.append(valuex)
            print("tC")
    if (i.betax > 0 and i.betax <= 0.17):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tB.mp3")
            audio.append(valuex)
            print("tB")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tC#.mp3")
            audio.append(valuex)
            print("tC#")
    if (i.betax > 0.17 and i.betax <= 0.3):
        if (i.betay < 0.5): # if (R1.betay >= 0.17 and R1.betay < 0.3):
            valuex = AudioSegment.from_file("notes_/tA#.mp3")
            audio.append(valuex)
            print("tA#")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tD.mp3")
            audio.append(valuex)
            print("tD")
    if (i.betax > 0.3 and i.betax <= 0.5):
        if (i.betay < 0.5): # (R1.betay == 1):
            valuex = AudioSegment.from_file("notes_/tD#.mp3")
            audio.append(valuex)
            print("tD#")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tA.mp3")
            audio.append(valuex)
            print("tA")
    if (i.betax > 0.5 and i.betax <= 0.64):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tE.mp3")
            audio.append(valuex)
            print("tE")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tG#.mp3")
            audio.append(valuex)
            print("tG#")
    if (i.betax > 0.64 and i.betax <= 0.84):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tF.mp3")
            audio.append(valuex)
```

```
            print("tF")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tG.mp3")
            audio.append(valuex)
            print("tG")
    if (i.betax > 0.84 and i.betax <= 1):
        #if (R1.betay == 0.5):
        valuex = AudioSegment.from_file("notes_/tF#.mp3")
        audio.append(valuex)
        print("tF#")



for i in Robotx._registry:
    print(audio[i.position]) # at this stage, they are supposed to all give tC.




mix = []

for s in range(11): # it should be between 1 and 11
    #values = (audio[s].overlay(audio[s+1])).overlay(audio[s+3])

    # is there a more synthetic way to write this??
    values = audio[s].overlay(audio[s+1])
    values2 = values.overlay(audio[s+2])
    values3 = values2.overlay(audio[s+3])
    values4 = values3.overlay(audio[s+4])
    values5 = values4.overlay(audio[s+5])
    values6 = values5.overlay(audio[s+6])
    values7 = values6.overlay(audio[s+7])
    values8 = values7.overlay(audio[s+8])
    values9 = values8.overlay(audio[s+9])
    mix.append(values9)
    print(mix[s])

mix[10].export("notes_/10_robot_sound/mixed_time_2.mp3", format='mp3') # export
play(mix[10])

# I'm trying to use the same code, but saving the file as another one.
```

```
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801940>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38010d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d053d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05b80>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0fb20>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f610>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0faf0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f490>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f9d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f1f0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f940>
tF#
tD#
tG
tF#
tG
tF#
tG
tA#
```

```
tG
tF#
<pydub.audio_segment.AudioSegment object at 0x7f8cd38010d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d053d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05b80>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0fb20>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f610>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0faf0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f490>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f9d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f1f0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f940>
<pydub.audio_segment.AudioSegment object at 0x7f8ce093bd00>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38532e0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f550>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f7f0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801100>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38015b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3853310>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801df0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38018b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38016a0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3801310>
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]:

```
# September 13

# Trying to understand how to update distances
# through the update of listX, listY

listX = list(k.betax for k in Robotx._registry)
listY = list(k.betay for k in Robotx._registry)

#print(listX)
#print(listY)

print(distance_A(T, listX, listY), distance_B(listX, listY))
```

```
0.2160222450900774 0.6544918773859241
```

In [30]:

```
# Quantum circuit construction ----------------------------------------------------
```

In [31]:

```
q = QuantumRegister(5, 'q') # qubits # changed to 9, formerly 15
m2 = ClassicalRegister(1, 'c1') # classical bits (separated is better)
m3 = ClassicalRegister(1, 'c2')
m4 = ClassicalRegister(1, 'c3')

qc3 = QuantumCircuit(q, m2, m3, m4) # to reach the target
qc4 = QuantumCircuit(q, m2, m3, m4) # to get back to the nest
```

In [32]:

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

In [33]:

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

In [34]:

```
# in case of ties on delta score, the max() function outputs the first maximum i
closest_robot = max(Robotx._registry, key=attrgetter('delta'))
print(f"Closest robot to the target: {closest_robot.name} {closest_robot.betax:.

# and then it enters the gate
vector0 = [closest_robot.alphax, closest_robot.betax]
vector1 = [closest_robot.alphay, closest_robot.betay]
vector3 = [closest_robot.gamma, closest_robot.delta]

normalized_v0 = vector0/np.linalg.norm(vector0)
normalized_v1 = vector1/np.linalg.norm(vector1)
normalized_v3 = vector3/np.linalg.norm(vector3)

print_formatted_vector(vector0, vector1, vector3)
print_formatted_vector(normalized_v0, normalized_v1, normalized_v3)
```

```
Closest robot to the target: R0 0.91 0.63 0.87
[0.09 0.91]
[0.37 0.63]
[0.13 0.87]
[0.10 1.00]
[0.51 0.86]
[0.15 0.99]
```

In [35]:

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

In [36]:

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

Out[36]:

```
<qiskit.circuit.instructionset.InstructionSet at 0x7f8cc014cb80>
```

In [37]:

```
# this is the core code, and it is unchanged across time

qc3.barrier(q)
qc3.ccx(q[0],q[1],q[3])
qc3.ccx(q[0],q[1],q[4])

qc3.reset(q[3])
qc3.reset(q[4])

qc3.ccx(q[0],q[2],q[3])
qc3.ccx(q[1],q[2],q[4])

qc3.x(q[2])

qc3.ch(q[2],q[3])
qc3.ch(q[2],q[4])

qc3.x(q[2])

qc3.barrier(q)

# perform measurements and store them in classical bits

qc3.measure(q[2],m2[0])
qc3.measure(q[3],m3[0])
qc3.measure(q[4],m4[0])

# visualization of the ciruit

# qc3.draw(fold=-1, output="mpl")
# plt.show();

print(qc3)
```

```
      ┌──────────────────────────────────┐ ░ 
»
 q_0: ┤ Initialize(0.096637,0.99532) ├─░─■──■─────────■──────────
─────»
      ├──────────────────────────────────┤ ░ │  │         │          
»
 q_1: ┤ Initialize(0.50705,0.86192) ├─░─■──■─────────┼─────■──────
─────»
      └┬─────────────────────────────────┘ ░ │  │         │     │  ┌─
┐     »
 q_2: ─┤ Initialize(0.14794,0.989) ├─░─┼──┼─────────■─────■──┤ X
├──■──»
       └──────────────────────────────────┘ ░ ┌─┴─┐  │       ┌─┴─┐  │  └─
┘┌─┴─┐»
 q_3: ──────────────────────────────────────░─┤ X ├──┼──|0>──┤ X ├──┼────
─┤ H ├»
                                            ░ └───┘┌─┴─┐     └───┘┌─┴─┐
└───┘»
 q_4: ──────────────────────────────────────░────┤ X ├──|0>────┤ X ├────
─────»
                                            ░    └───┘          └───┘
»
c1: 1/═══════════════════════════════════════════════════════════════
════»
```

```
»
c2: 1/═══════════════════════════════════════════════════════════════════════
═══════»

»
c3: 1/═══════════════════════════════════════════════════════════════════════
═══════»

»
«
« q_0: ──────────────────────────────────
«
« q_1: ──────────────────────────────────
«
« q_2: ──────■─────X─────M──────────────
«            │           │
« q_3: ──────┼─────────────M──────────────
«            │           ┃
« q_4: ──────H─────────────┼─────M────────
«                          ┃     ┃
«c1: 1/═══════════════════╩═════╪═══════
«                          0     ┃
«c2: 1/═══════════════════════════╩═══════
«                                0     ┃
«c3: 1/═══════════════════════════════════
«                                      0
```

In [38]:

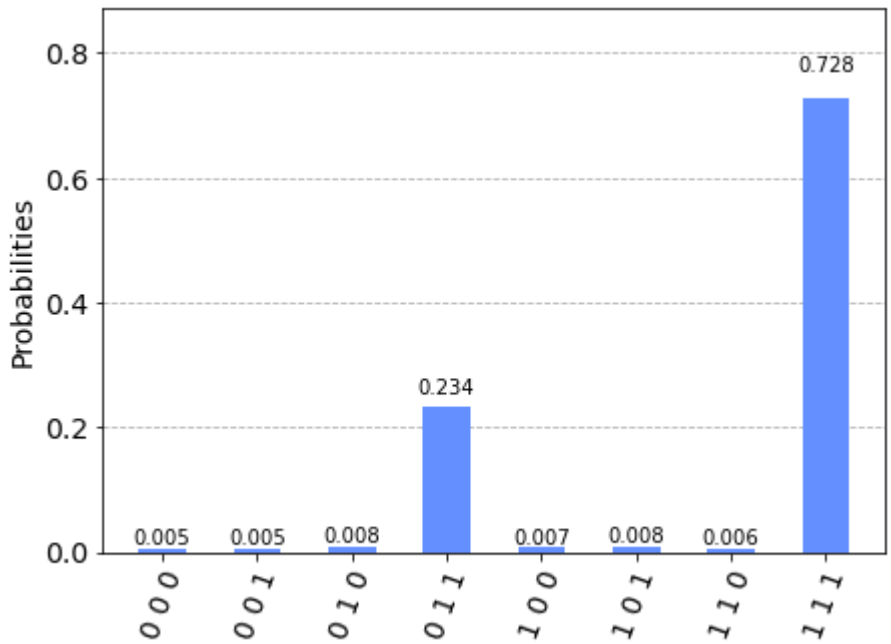
```
aer_sim = Aer.get_backend("aer_simulator")
transpiled_qc = transpile(qc3, aer_sim)
result = aer_sim.run(transpiled_qc).result()

counts = result.get_counts()
print("counts: ", counts)

plot_histogram(result.get_counts())
```

```
counts:  {'0 0 1': 5, '1 1 0': 6, '1 0 0': 7, '0 1 1': 240, '1 1 1': 7
45, '1 0 1': 8, '0 0 0': 5, '0 1 0': 8}
```

Out[38]:



In [39]:

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

In [40]:

```
num_most_prob_states = 4

# https://docs.python.org/3/library/heapq.html:
#
# heapq.nlargest(n, iterable, key=None) returns a list with the n largest elemer
#
most_prob_dict = dict(heapq.nlargest(num_most_prob_states, counts.items(), key=i
print(f"{num_most_prob_states} most probable states: {most_prob_dict}")

outcome0, outcome1 = eval_outcome(most_prob_dict, 0), eval_outcome(most_prob_dic

print(f"outcome0: {outcome0:.2f}\noutcome1: {outcome1:.2f}")

```

```
4 most probable states: {'1 1 1': 745, '0 1 1': 240, '1 0 1': 8, '0 1
0': 8}
outcome0: 0.75
outcome1: 0.99
```

In [41]:

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

In [42]:

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

```
R0 0.91 0.63
R1 0.31 0.40
R2 0.72 0.67
R3 0.97 0.74
R4 0.70 0.65
R5 0.87 0.65
R6 0.71 0.57
R7 0.23 0.47
R8 0.81 0.64
R9 0.96 0.75
```

In [43]:

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

```
R0 0.91 0.63 0.13 1
R1 0.99 0.75 0.60 2
R2 0.99 0.75 0.24 3
R3 0.99 0.75 0.25 4
R4 0.99 0.75 0.25 5
R5 0.99 0.75 0.15 6
R6 0.99 0.75 0.21 7
R7 0.99 0.75 0.67 8
R8 0.99 0.75 0.17 9
R9 0.99 0.75 0.26 10
```

In [44]:

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

```
before the gate: R0 0.87
before the gate: R1 0.40
before the gate: R2 0.76
before the gate: R3 0.75
before the gate: R4 0.75
before the gate: R5 0.85
before the gate: R6 0.79
before the gate: R7 0.33
before the gate: R8 0.83
before the gate: R9 0.74
```

In [45]:

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

```
after the gate: R0 0.87
after the gate: R1 0.73
after the gate: R2 0.73
after the gate: R3 0.73
after the gate: R4 0.73
after the gate: R5 0.73
after the gate: R6 0.73
after the gate: R7 0.73
after the gate: R8 0.73
after the gate: R9 0.73
```

In [46]:

```
# audio block #3

# audio 3

# we can define "audio" as an attribute... no, better not.

audio = []

for x in range(11): # it should be between 1 and 11
    valuex = AudioSegment.from_file("notes_/tC.mp3")
    audio.append(valuex)
for i in range(11):
    print(audio[i]) # at this stage, they are supposed to all give tC.mp3

for i in Robotx._registry:
    if (i.betax == 0):
        if (i.betay == 0.5):
            valuex = AudioSegment.from_file("notes_/tC.mp3") # i.audio
            audio.append(valuex)
            print("tC")
    if (i.betax > 0 and i.betax <= 0.17):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tB.mp3")
            audio.append(valuex)
            print("tB")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tC#.mp3")
            audio.append(valuex)
            print("tC#")
    if (i.betax > 0.17 and i.betax <= 0.3):
        if (i.betay < 0.5): # if (R1.betay >= 0.17 and R1.betay < 0.3):
            valuex = AudioSegment.from_file("notes_/tA#.mp3")
            audio.append(valuex)
            print("tA#")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tD.mp3")
            audio.append(valuex)
            print("tD")
    if (i.betax > 0.3 and i.betax <= 0.5):
        if (i.betay < 0.5): # (R1.betay == 1):
            valuex = AudioSegment.from_file("notes_/tD#.mp3")
            audio.append(valuex)
            print("tD#")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tA.mp3")
            audio.append(valuex)
            print("tA")
    if (i.betax > 0.5 and i.betax <= 0.64):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tE.mp3")
            audio.append(valuex)
            print("tE")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tG#.mp3")
            audio.append(valuex)
            print("tG#")
    if (i.betax > 0.64 and i.betax <= 0.84):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tF.mp3")
```

```
            audio.append(valuex)
            print("tF")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tG.mp3")
            audio.append(valuex)
            print("tG")
    if (i.betax > 0.84 and i.betax <= 1):
        #if (R1.betay == 0.5):
        valuex = AudioSegment.from_file("notes_/tF#.mp3")
        audio.append(valuex)
        print("tF#")



for i in Robotx._registry:
    print(audio[i.position]) # at this stage, they are supposed to all give tC.




mix = []

for s in range(11): # it should be between 1 and 11
    #values = (audio[s].overlay(audio[s+1])).overlay(audio[s+3])

    # is there a more synthetic way to write this??
    values = audio[s].overlay(audio[s+1])
    values2 = values.overlay(audio[s+2])
    values3 = values2.overlay(audio[s+3])
    values4 = values3.overlay(audio[s+4])
    values5 = values4.overlay(audio[s+5])
    values6 = values5.overlay(audio[s+6])
    values7 = values6.overlay(audio[s+7])
    values8 = values7.overlay(audio[s+8])
    values9 = values8.overlay(audio[s+9])
    mix.append(values9)
    print(mix[s])

mix[10].export("notes_/10_robot_sound/mixed_time_3.mp3", format='mp3') # export
play(mix[10])

```

```
<pydub.audio_segment.AudioSegment object at 0x7f8cc0128d60>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0051fa0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd4c93cd0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd4c936d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd4c93f40>
<pydub.audio_segment.AudioSegment object at 0x7f8cd4c933d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05df0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05520>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d053d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d055b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d058e0>
tF#
tF#
tF#
tF#
tF#
tF#
tF#
tF#
```

```
tF#
tF#
<pydub.audio_segment.AudioSegment object at 0x7f8ca0051fa0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd4c93cd0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd4c936d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd4c93f40>
<pydub.audio_segment.AudioSegment object at 0x7f8cd4c933d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05df0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d05520>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d053d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d055b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d058e0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0051e80>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0072400>
<pydub.audio_segment.AudioSegment object at 0x7f8ce1cd5fd0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0072040>
<pydub.audio_segment.AudioSegment object at 0x7f8ce1ceaaf0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca00518b0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0072430>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38015b0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38530d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f9d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f2e0>
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]:

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

In [48]:

```
# not for now

for i in  Robotx._registry: # recalculate the rewards
    i.delta = reward(T, i.betax, i.betay)
    i.gamma = (1 - i.delta, 3)

max_attr_ = max(Robotx._registry, key=attrgetter('delta'))
print(f"max_attr_.delta: {max_attr_.delta:.2f}")

for i in Robotx._registry:
    if (i.delta == max_attr_.delta):
        print(f"Most successful robot: {i.name} {i.betax:.2f} {i.betay:.2f} {i.d

for j in Robotx._registry:
    # to get other robots following it:
    if (j != max_attr_): # changed here
        flag = True
        while flag:
            flag = False
            j.alphax = max_attr_.alphax + np.random.uniform(0,0.01)
            j.betax = 1 - j.alphax
            j.alphay = max_attr_.alphay + np.random.uniform(0,0.01)
            j.betay = 1 - j.alphay
            if (j.betax - O.x <= 0.2 and j.betay - O.y <= 0.2):
                flag = True

# recalculate the rewards here:

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

```
max_attr_.delta: 0.87
Most successful robot: R0 0.91 0.63 0.87
R0 0.87
R1 0.88
R2 0.88
R3 0.88
R4 0.88
R5 0.88
R6 0.88
R7 0.87
R8 0.87
R9 0.88
```

In [49]:

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

```
R0 0.87
R1 0.88
R2 0.88
R3 0.88
R4 0.88
R5 0.88
R6 0.88
R7 0.87
R8 0.87
R9 0.88
```

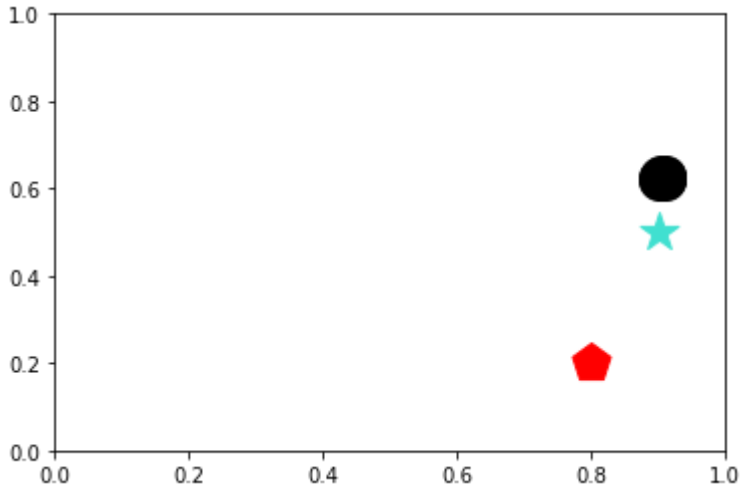
In [50]:

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

```
R0 0.91 0.63 0.13 0.87 1
R1 0.91 0.62 0.12 0.88 2
R2 0.90 0.62 0.12 0.88 3
R3 0.90 0.62 0.12 0.88 4
R4 0.90 0.62 0.12 0.88 5
R5 0.90 0.62 0.12 0.88 6
R6 0.91 0.62 0.12 0.88 7
R7 0.91 0.63 0.13 0.87 8
R8 0.90 0.63 0.13 0.87 9
R9 0.91 0.62 0.12 0.88 10
```

In [51]:

```
plot_scatterplot()
```



In [52]:

```
# September 13

# Trying to understand how to update distances
# through the update of listX, listY

listX = list(k.betax for k in Robotx._registry)
listY = list(k.betay for k in Robotx._registry)

#print(listX)
#print(listY)

print(distance_A(T, listX, listY), distance_B(listX, listY))
```

```
0.12436949064428683 0.011448679801670152
```

In [53]:

```
# audio block #4

# audio 4

# we can define "audio" as an attribute... no, better not.

audio = []

for x in range(11): # it should be between 1 and 11
    valuex = AudioSegment.from_file("notes_/tC.mp3")
    audio.append(valuex)
for i in range(11):
    print(audio[i]) # at this stage, they are supposed to all give tC.mp3

for i in Robotx._registry:
    if (i.betax == 0):
        if (i.betay == 0.5):
            valuex = AudioSegment.from_file("notes_/tC.mp3") # i.audio
            audio.append(valuex)
            print("tC")
    if (i.betax > 0 and i.betax <= 0.17):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tB.mp3")
            audio.append(valuex)
            print("tB")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tC#.mp3")
            audio.append(valuex)
            print("tC#")
    if (i.betax > 0.17 and i.betax <= 0.3):
        if (i.betay < 0.5): # if (R1.betay >= 0.17 and R1.betay < 0.3):
            valuex = AudioSegment.from_file("notes_/tA#.mp3")
            audio.append(valuex)
            print("tA#")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tD.mp3")
            audio.append(valuex)
            print("tD")
    if (i.betax > 0.3 and i.betax <= 0.5):
        if (i.betay < 0.5): # (R1.betay == 1):
            valuex = AudioSegment.from_file("notes_/tD#.mp3")
            audio.append(valuex)
            print("tD#")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tA.mp3")
            audio.append(valuex)
            print("tA")
    if (i.betax > 0.5 and i.betax <= 0.64):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tE.mp3")
            audio.append(valuex)
            print("tE")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tG#.mp3")
            audio.append(valuex)
            print("tG#")
    if (i.betax > 0.64 and i.betax <= 0.84):
        if (i.betay < 0.5):
            valuex = AudioSegment.from_file("notes_/tF.mp3")
```

```
            audio.append(valuex)
            print("tF")
        if (i.betay >= 0.5):
            valuex = AudioSegment.from_file("notes_/tG.mp3")
            audio.append(valuex)
            print("tG")
    if (i.betax > 0.84 and i.betax <= 1):
        #if (R1.betay == 0.5):
        valuex = AudioSegment.from_file("notes_/tF#.mp3")
        audio.append(valuex)
        print("tF#")



for i in Robotx._registry:
    print(audio[i.position]) # at this stage, they are supposed to all give tC.




mix = []

for s in range(11): # it should be between 1 and 11
    #values = (audio[s].overlay(audio[s+1])).overlay(audio[s+3])

    # is there a more synthetic way to write this??
    values = audio[s].overlay(audio[s+1])
    values2 = values.overlay(audio[s+2])
    values3 = values2.overlay(audio[s+3])
    values4 = values3.overlay(audio[s+4])
    values5 = values4.overlay(audio[s+5])
    values6 = values5.overlay(audio[s+6])
    values7 = values6.overlay(audio[s+7])
    values8 = values7.overlay(audio[s+8])
    values9 = values8.overlay(audio[s+9])
    mix.append(values9)
    print(mix[s])

mix[10].export("notes_/10_robot_sound/mixed_time_4.mp3", format='mp3') # export
play(mix[10])

```

```
<pydub.audio_segment.AudioSegment object at 0x7f8cd4c933d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f1f0>
<pydub.audio_segment.AudioSegment object at 0x7f8cb052aac0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0fb20>
<pydub.audio_segment.AudioSegment object at 0x7f8cb052aa00>
<pydub.audio_segment.AudioSegment object at 0x7f8c60015370>
<pydub.audio_segment.AudioSegment object at 0x7f8cb052aa90>
<pydub.audio_segment.AudioSegment object at 0x7f8c600153d0>
<pydub.audio_segment.AudioSegment object at 0x7f8c600152b0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0051520>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0051610>
tF#
tF#
tF#
tF#
tF#
tF#
tF#
tF#
```

```
tF#
tF#
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0f1f0>
<pydub.audio_segment.AudioSegment object at 0x7f8cb052aac0>
<pydub.audio_segment.AudioSegment object at 0x7f8cd1d0fb20>
<pydub.audio_segment.AudioSegment object at 0x7f8cb052aa00>
<pydub.audio_segment.AudioSegment object at 0x7f8c60015370>
<pydub.audio_segment.AudioSegment object at 0x7f8cb052aa90>
<pydub.audio_segment.AudioSegment object at 0x7f8c600153d0>
<pydub.audio_segment.AudioSegment object at 0x7f8c600152b0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0051520>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0051610>
<pydub.audio_segment.AudioSegment object at 0x7f8cd4c93f40>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0051070>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0051a60>
<pydub.audio_segment.AudioSegment object at 0x7f8cc0128d60>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38015b0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0072430>
<pydub.audio_segment.AudioSegment object at 0x7f8cd38530d0>
<pydub.audio_segment.AudioSegment object at 0x7f8cb0542e20>
<pydub.audio_segment.AudioSegment object at 0x7f8cc0116af0>
<pydub.audio_segment.AudioSegment object at 0x7f8ca0072040>
<pydub.audio_segment.AudioSegment object at 0x7f8cd3853280>
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]:

```
# September 13

# Trying to understand how to update distances
# through the update of listX, listY

listX = list(k.betax for k in Robotx._registry)
listY = list(k.betay for k in Robotx._registry)

#print(listX)
#print(listY)

print(distance_A(T, listX, listY), distance_B(listX, listY))

print("Euclidean", Euclidean_distance(T, listX, listY))
print("Manhattan", Manhattan_distance(T, listX, listY))
print("Cosine", Cosine_distance(T, listX, listY))

```

```
0.12436949064428683 0.011448679801670152
Euclidean 0.12436949064428683
Manhattan 0.13063307328428486
Cosine 0.005663149737921036
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```