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
```

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 [11]:

```
# arbitrary number of robots that, at the start, are uniformly distributed in tl
# 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 visuali

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

```
R0 0.11 0.44 0.79 0.21 1
R1 0.13 0.59 0.77 0.23 2
R2 0.29 0.49 0.61 0.39 3
R3 0.26 0.59 0.64 0.36 4
R4 0.16 0.53 0.74 0.26 5
R5 0.11 0.49 0.79 0.21 6
R6 0.29 0.57 0.62 0.38 7
R7 0.21 0.47 0.69 0.31 8
R8 0.11 0.60 0.80 0.20 9
R9 0.25 0.48 0.65 0.35 10
```

In [13]:

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

```
R0 0.21
R1 0.23
R2 0.39
R3 0.36
R4 0.26
R5 0.21
R6 0.38
R7 0.31
R8 0.20
R9 0.35
```

In [14]:

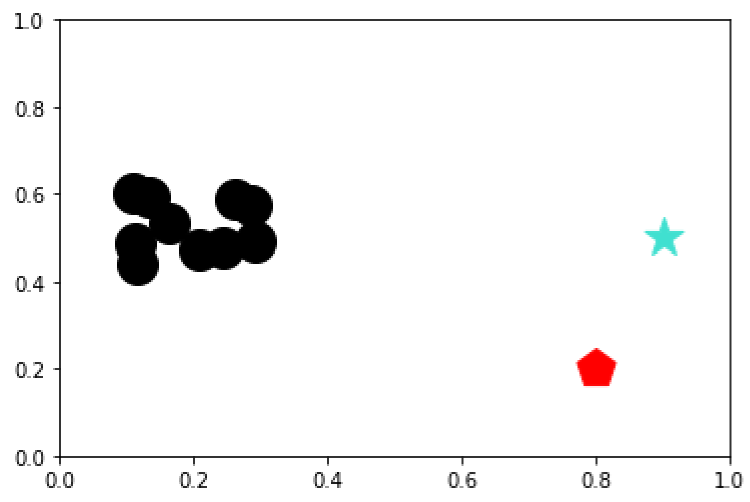
```
def plot_scatterplot():
    for i in  Robotx._registry:
        plt.scatter(i.betax, i.betay, s = 400, marker = 'o', color = 'black')

    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 [15]:

```python
# 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 0x7fac099dd1f0>
<pydub.audio_segment.AudioSegment object at 0x7fac099dd9a0>
<pydub.audio_segment.AudioSegment object at 0x7fac099dd1c0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353fa0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353310>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353c70>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353ca0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353bb0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a38a280>
<pydub.audio_segment.AudioSegment object at 0x7fac09c15ac0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a38a2e0>
1
2
3
4
5
6
7
8
9
10
<pydub.audio_segment.AudioSegment object at 0x7fac099dd9a0>
<pydub.audio_segment.AudioSegment object at 0x7fac099dd1c0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353fa0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353310>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353c70>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353ca0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353bb0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a38a280>
<pydub.audio_segment.AudioSegment object at 0x7fac09c15ac0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a38a2e0>
```

In [16]:

```python
# 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 0x7fac3a297f40>
<pydub.audio_segment.AudioSegment object at 0x7fac099dd610>
<pydub.audio_segment.AudioSegment object at 0x7fac099dd5b0>
<pydub.audio_segment.AudioSegment object at 0x7fac099dd880>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4ca550>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4cac70>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4cabe0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4cabb0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359f40>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3592b0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359160>
tB
tC#
tA#
tD
tC#
tB
tD
```

```
tA#
tC#
tA#
<pydub.audio_segment.AudioSegment object at 0x7fac099dd610>
<pydub.audio_segment.AudioSegment object at 0x7fac099dd5b0>
<pydub.audio_segment.AudioSegment object at 0x7fac099dd880>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4ca550>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4cac70>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4cabe0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4cabb0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359f40>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3592b0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359160>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4cafd0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a497ee0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a353430>
<pydub.audio_segment.AudioSegment object at 0x7fac381f4760>
<pydub.audio_segment.AudioSegment object at 0x7fac3a497d30>
<pydub.audio_segment.AudioSegment object at 0x7fac3a497cd0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4b7e20>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4ca070>
<pydub.audio_segment.AudioSegment object at 0x7fac3a29e2b0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359220>
<pydub.audio_segment.AudioSegment object at 0x7fac3a4cac10>
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 [17]:

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

```
R0 0.21 achtung!
R1 0.23 achtung!
R2 0.39 achtung!
R3 0.36 achtung!
R4 0.26 achtung!
R5 0.21 achtung!
R6 0.38 achtung!
R7 0.31 achtung!
R8 0.20 achtung!
R9 0.35 achtung!
```

In [18]:

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

In [19]:

```python
# 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 [20]:

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

```
R0 0.19 0.65 0.79 1
R1 0.69 0.83 0.77 2
R2 0.59 0.58 0.61 3
R3 0.21 0.65 0.64 4
R4 0.44 0.41 0.74 5
R5 0.91 0.41 0.79 6
R6 0.29 0.77 0.62 7
R7 0.96 0.90 0.69 8
R8 0.55 0.74 0.80 9
R9 0.97 0.48 0.65 10
```

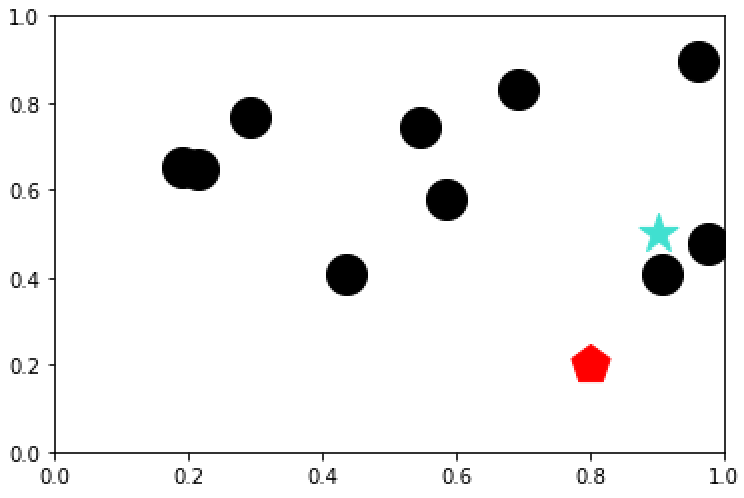
In [21]:

```python
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 0.27
R1 0.61
R2 0.68
R3 0.30
R4 0.53
R5 0.91
R6 0.34
R7 0.60
R8 0.57
R9 0.92
```

In [22]:

```
plot_scatterplot()
```



In [23]:

```
# 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 0x7fac3a353b50>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3599a0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359fd0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359f40>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359160>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302580>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3020a0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302790>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302940>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302970>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302910>
tD
tG
tG#
tD
tD#
tF#
tD
```

```
tF#
tG#
tF#
<pydub.audio_segment.AudioSegment object at 0x7fac3a3599a0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359fd0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359f40>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359160>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302580>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3020a0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302790>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302940>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302970>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302910>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359280>
<pydub.audio_segment.AudioSegment object at 0x7fac381cc7c0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3028e0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3025e0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302fd0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3028b0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3029d0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3022b0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3590d0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302160>
<pydub.audio_segment.AudioSegment object at 0x7fac3a302af0>
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 [24]:

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

In [25]:

```
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 [26]:

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

In [27]:

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

In [28]:

```
# 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: R9 0.97 0.48 0.92
[0.03 0.97]
[0.52 0.48]
[0.08 0.92]
[0.03 1.00]
[0.74 0.67]
[0.09 1.00]
```

In [29]:

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

In [30]:

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

Out[30]:

```
<qiskit.circuit.instructionset.InstructionSet at 0x7fac18db0e00>
```

In [31]:

```
# 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(0.025934,0.99966) ├─░───■────■─────────■──────
──────»
      ├────────────────────────────┬─┘ ░   │    │         │
»
 q_1: ┤ Initialize(0.74121,0.67127) ├──░───■────■─────────┼────■──────
──────»
      ├────────────────────────────┴─┤ ░   │    │         │    │  ┌───
┐     »
 q_2: ┤ Initialize(0.085123,0.99637) ├─░───┼────┼─────────■────■──┤ 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 [32]:

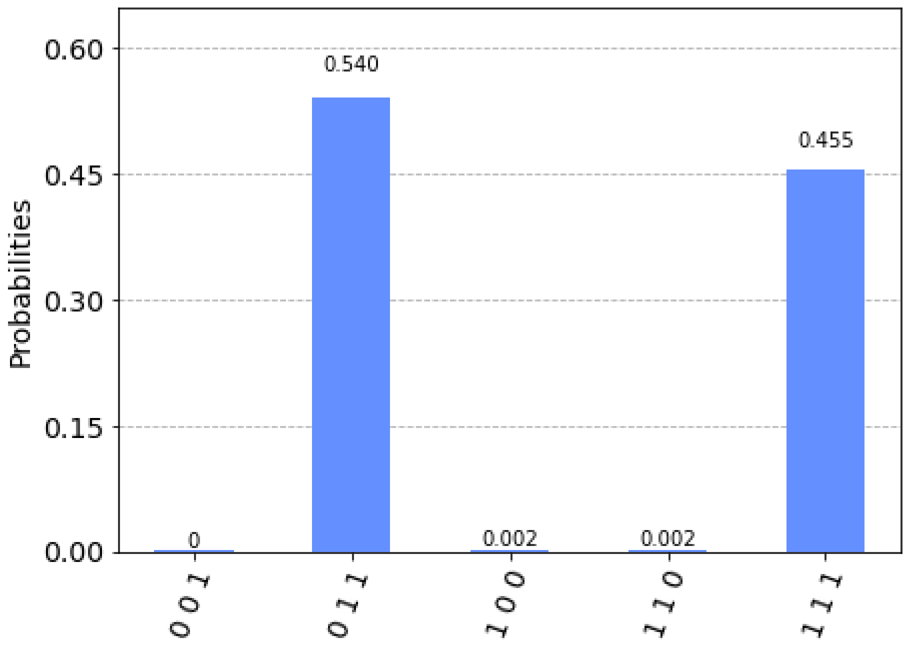
```
aer_sim = Aer.get_backend("aer_simulator")
transpiled_qc = transpile(qc3, aer_sim)
result = aer_sim.run(transpiled_qc).result()

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

plot_histogram(result.get_counts())
```

```
counts:  {'0 0 1': 1, '1 0 0': 2, '1 1 1': 466, '1 1 0': 2, '0 1 1': 5
53}
```

Out[32]:



In [33]:

```
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 [34]:

```
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 elemen
#
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: {'0 1 1': 553, '1 1 1': 466, '1 0 0': 2, '1 1
0': 2}
outcome0: 0.46
outcome1: 1.00
```

In [35]:

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

In [36]:

```
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 same 
        # 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 0.19 0.65
R1 0.69 0.83
R2 0.59 0.58
R3 0.21 0.65
R4 0.44 0.41
R5 0.91 0.41
R6 0.29 0.77
R7 0.96 0.90
R8 0.55 0.74
R9 0.97 0.48
```

In [37]:

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

```
R0 1.00 0.46 0.73 1
R1 1.00 0.46 0.39 2
R2 1.00 0.46 0.32 3
R3 1.00 0.46 0.70 4
R4 1.00 0.46 0.47 5
R5 1.00 0.46 0.09 6
R6 1.00 0.46 0.66 7
R7 1.00 0.46 0.40 8
R8 1.00 0.46 0.43 9
R9 0.97 0.48 0.08 10
```

In [38]:

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

```
before the gate: R0 0.27
before the gate: R1 0.61
before the gate: R2 0.68
before the gate: R3 0.30
before the gate: R4 0.53
before the gate: R5 0.91
before the gate: R6 0.34
before the gate: R7 0.60
before the gate: R8 0.57
before the gate: R9 0.92
```

In [39]:

```
# 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 0.89
after the gate: R1 0.89
after the gate: R2 0.89
after the gate: R3 0.89
after the gate: R4 0.89
after the gate: R5 0.89
after the gate: R6 0.89
after the gate: R7 0.89
after the gate: R8 0.89
after the gate: R9 0.92
```

In [40]:

```
# 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 0x7fac3a345be0>
<pydub.audio_segment.AudioSegment object at 0x7fac3bd7eb50>
<pydub.audio_segment.AudioSegment object at 0x7fac3a32e070>
<pydub.audio_segment.AudioSegment object at 0x7fac3a32ed60>
<pydub.audio_segment.AudioSegment object at 0x7fac3bd7ebb0>
<pydub.audio_segment.AudioSegment object at 0x7fac09b62100>
<pydub.audio_segment.AudioSegment object at 0x7fac09b62040>
<pydub.audio_segment.AudioSegment object at 0x7fac09b623a0>
<pydub.audio_segment.AudioSegment object at 0x7fac09b62af0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3592b0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359fd0>
tF#
tF#
tF#
tF#
tF#
tF#
tF#
tF#
```

```
tF#
tF#
<pydub.audio_segment.AudioSegment object at 0x7fac3bd7eb50>
<pydub.audio_segment.AudioSegment object at 0x7fac3a32e070>
<pydub.audio_segment.AudioSegment object at 0x7fac3a32ed60>
<pydub.audio_segment.AudioSegment object at 0x7fac3bd7ebb0>
<pydub.audio_segment.AudioSegment object at 0x7fac09b62100>
<pydub.audio_segment.AudioSegment object at 0x7fac09b62040>
<pydub.audio_segment.AudioSegment object at 0x7fac09b623a0>
<pydub.audio_segment.AudioSegment object at 0x7fac09b62af0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a3592b0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359fd0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a2dee80>
<pydub.audio_segment.AudioSegment object at 0x7fac3a2de0a0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a2defd0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a2ded30>
<pydub.audio_segment.AudioSegment object at 0x7fac3bd5d310>
<pydub.audio_segment.AudioSegment object at 0x7fac3a38a0a0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a2de5e0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a2de520>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359160>
<pydub.audio_segment.AudioSegment object at 0x7fac3a2dec40>
<pydub.audio_segment.AudioSegment object at 0x7fac09b77190>
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 [41]:

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

In [42]:

```
r now

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

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

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

 Robotx._registry:
 get other robots following it:
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

ulate the rewards here:

  Robotx._registry: # recalculate the rewards
lta = reward(T, k.betax, k.betay)
mma = 1 - k.delta
t(f"{k.name} {k.delta:.2f}")
```

```
max_attr_.delta: 0.92
Most successful robot: R9 0.97 0.48 0.92
R0 0.93
R1 0.92
R2 0.93
R3 0.93
R4 0.92
R5 0.93
R6 0.92
R7 0.93
R8 0.93
R9 0.92
```

In [43]:

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

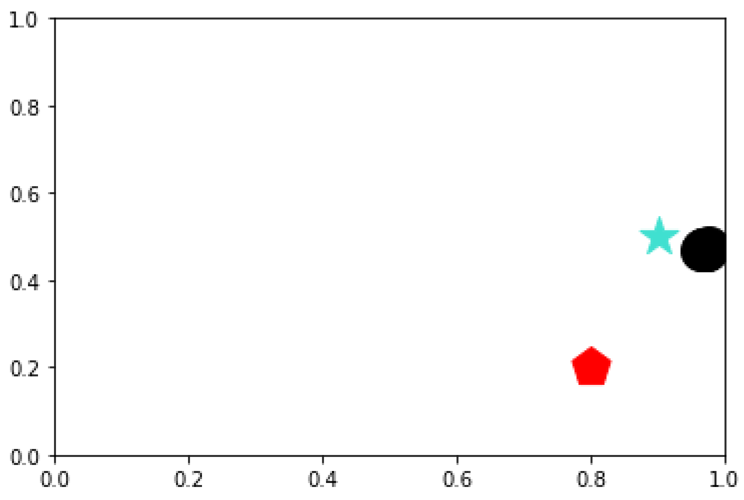
In [44]:

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

```
R0 0.97 0.47 0.07 0.93 1
R1 0.97 0.47 0.08 0.92 2
R2 0.97 0.47 0.07 0.93 3
R3 0.97 0.47 0.07 0.93 4
R4 0.97 0.47 0.08 0.92 5
R5 0.97 0.47 0.07 0.93 6
R6 0.97 0.47 0.08 0.92 7
R7 0.97 0.47 0.07 0.93 8
R8 0.96 0.47 0.07 0.93 9
R9 0.97 0.48 0.08 0.92 10
```

In [45]:

```
plot_scatterplot()
```



In [ ]:

```

```

In [46]:

```
# 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 0x7fabe8048550>
<pydub.audio_segment.AudioSegment object at 0x7fac3a38a400>
<pydub.audio_segment.AudioSegment object at 0x7fac3bd7eb50>
<pydub.audio_segment.AudioSegment object at 0x7fabe8048580>
<pydub.audio_segment.AudioSegment object at 0x7fac3a38a550>
<pydub.audio_segment.AudioSegment object at 0x7fac3bd7ebb0>
<pydub.audio_segment.AudioSegment object at 0x7fabe8035670>
<pydub.audio_segment.AudioSegment object at 0x7fabe8035640>
<pydub.audio_segment.AudioSegment object at 0x7fabe8035760>
<pydub.audio_segment.AudioSegment object at 0x7fac3bd8eb80>
<pydub.audio_segment.AudioSegment object at 0x7fabe80356a0>
tF#
tF#
tF#
tF#
tF#
tF#
tF#
tF#
```

```
tF#
tF#
<pydub.audio_segment.AudioSegment object at 0x7fac3a38a400>
<pydub.audio_segment.AudioSegment object at 0x7fac3bd7eb50>
<pydub.audio_segment.AudioSegment object at 0x7fabe8048580>
<pydub.audio_segment.AudioSegment object at 0x7fac3a38a550>
<pydub.audio_segment.AudioSegment object at 0x7fac3bd7ebb0>
<pydub.audio_segment.AudioSegment object at 0x7fabe8035670>
<pydub.audio_segment.AudioSegment object at 0x7fabe8035640>
<pydub.audio_segment.AudioSegment object at 0x7fabe8035760>
<pydub.audio_segment.AudioSegment object at 0x7fac3bd8eb80>
<pydub.audio_segment.AudioSegment object at 0x7fabe80356a0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a345be0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a38a850>
<pydub.audio_segment.AudioSegment object at 0x7fac481a6640>
<pydub.audio_segment.AudioSegment object at 0x7fac09b62b50>
<pydub.audio_segment.AudioSegment object at 0x7fac09b62310>
<pydub.audio_segment.AudioSegment object at 0x7fac481b3760>
<pydub.audio_segment.AudioSegment object at 0x7fac09b62040>
<pydub.audio_segment.AudioSegment object at 0x7fac3a359c40>
<pydub.audio_segment.AudioSegment object at 0x7fac481b3c10>
<pydub.audio_segment.AudioSegment object at 0x7fac3a2defd0>
<pydub.audio_segment.AudioSegment object at 0x7fac3a2de520>
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 [ ]:

```

```

In [ ]:

```

```