ASSIGNMENT 2

G97

Ángela María Durán Pinto:100472766
Paula Gutiérrez Arroyo:100472845

PHASE 1: Selection of the state information and reward function

We have constructed different agents, selecting different sets of attributes, in order to discover which one makes the pac-man have a better performance.

Our state is defined by the following attributes (all are boolean):

- `Gnorht, GWest`: Give the relative position of the Pac-man with respect to the closest ghost. If Gnorth is True, pacman is above the ghost, if it is False it is bellow. The same happens for GWest, if it is True, pacman is on the left, otherwise it is on the right.

- `wallNorth,wallSouth,wallEast,wallWest`:They say whether there is a wall or not.

- `ghost_living` : It gives a boolean to know if the closest ghost is alive or dead.

1º set: `state_tick = (Gnorht, Gwest, ghost_living, wallNorth, wallSouth, wallEast, wallWest)`
2º set: `state_tick = (Gnorht, Gwest, ghost_living)`

We start working with the 1º set, in order to construct all the functions needed and follow the procedure, and then we will start comparing how different states can influence the behavior of the pac-man.

On the other hand, for the action we just call the getAction function, passing as a parameter the current state.

```
action = agent.getAction(observation)
```

We create a function getState which, given the current state of the game, returns `state_tick`.
For the wall attributes, we get the legal actions that the pacman has in the current state, and if the action is allowed it means that there is not a wall on that side.

Additionally, for the attributes of the relative position of the pacman, we get the index of the closest ghost, and then we compare the x and y coordinates of both agents, obtaining in this way the values for the attributes.

Regarding the ghost_living, as we already have the index of our target ghost,using the getLivingGhosts() function, we get if the

This function is called in computePosition as

```
tupleState= self.getState(state)
```

Basically, we iterate through the tuple using a bitwise shift left operation and sum them to calculate the contribution of each element to the row.

PHASE 2:Generation of the agent

update()

Basically, the update function is responsible for changing the values of Q in our Q table, depending on the state and if it is terminal or not. Let us get into detail:

In this code, it firstly determines the legal actions that the Pacman can take in our current state. Then, we take the index in the Q-table of the row and column of our state and we check whether there are legal actions available or not. If there are not, we update the Q-value:

```
self.q_table[position][column] = (1 - a) * self.getQValue(state, action) + a * (reward)
```

where a is alpha, which we will change in our experiments to know which is the best value for it.

If it is not a terminal state, we update the q-value using the Q-learning update equation.

```
self.q_table[position][column] = (1 - a) * self.getQValue(state, action) + a * (reward + self.discount * self.computeValueFromQValues(nextState))
```

computePosition()

It is used to compute the row index in the Q-table.

The loop iterates over each element in tuplaState, which is a tuple of boolean values representing the state of the game. For each boolean value in the tuple, we create a localAux variable and set it to 0. If the boolean value is True, we set localAux to 1. This is done to convert the boolean value to a binary digit (0 or 1) so that we can perform the weighted sum. The formula used to compute the row value is (2**n)*localAux, where n is the index of the current boolean value in the tuple and localAux is either 0 or 1 depending on the boolean value.

```
tuplaState= self.getState(state)

aux = 0

for n in range(len(tuplaState)):
   localAux = 0
   if tuplaState[n] == True: localAux = 1

   aux += (2 ** n) * localAux
```

```
myRow = aux

return myRow
```

getReward()

Defining a good reward function is crucial to the success of a reinforcement learning agent. The reward function should incentivize the agent to achieve the desired objective (eating the ghosts in this case) while avoiding undesirable behavior (e.g., getting stuck in a loop, running into walls).

In order to design our getReward function, we have defined different reward values. Which means that, depending on the action made on the certain agent (among the ones we chose), it would add a certain value or we penalize the opposite effect.

For instance, when the pacman is getting further away from the ghost, we penalize it and when it gets closer, we reward it. That way, it can learn that the goal here is to get closer to it rather than further away.

On the other hand, we give to pac man a higher reward when it eats a ghost, as it is the goal of the game. For that, we compute if the target ghost (the closest one) is alive in the current state, and in the net state. If we get
True and in the next state changes to False, it has eaten the ghost, and therefore it receives the reward.

Experimentation

For the experimentation we have followed an iterative process, in which we have trained our agents using initially high values for our parameters in order to let it learn from the environment, and then we start decreasing them. We have done this whole process twice, because we have implemented two agents. In the following tables we summarize our experimentation, and it helps to compare the performance of both agents.

labAA1

iteration	epsilon	alpha	gamma	score A1	score A2
1	1	0.8	0.8	-577	75
2	1	0.8	0.8	-583	81
3	1	0.8	0.8	57	103
4	1	0.8	0.8	1	139

iteration	epsilon	alpha	gamma	score A1	score A2
5	0.8	0.8	0.8	-11	39
6	0.8	0.8	0.8	57	71
7	0.8	0.8	0.8	71	29
8	0.8	0.8	0.8	81	67
9	0.8	0.8	0.8	135	41
10	0.8	0.8	0.8	-161	13
11	0.8	0.8	0.8	135	55

iteration	epsilon	alpha	gamma	score A1	score A2
12	0.6	0.8	0.8	99	163
13	0.6	0.8	0.8	161	167
14	0.6	0.8	0.8	153	151
15	0.6	0.6	0.8	151	165
16	0.6	0.6	0.8	163	171
17	0.6	0.6	0.8	119	161
18	0.6	0.6	0.8	113	125
19	0.6	0.6	0.8	155	143
20	0.6	0.6	0.8	147	173

iteration	epsilon	alpha	gamma	score A1	score A2
21	0.4	0.6	0.8	175	175
22	0.4	0.6	0.8	163	177
23	0.4	0.6	0.8	173	175
24	0.4	0.6	0.8	175	171
25	0.4	0.6	0.8	161	167

26	0.4	0.4	0.8	177	165
27	0.4	0.4	0.8	179	169
28	0.4	0.4	0.8	171	163
29	0.4	0.4	0.8	155	173
30	0.4	0.4	0.8	137	165

iteration	epsilon	alpha	gamma	score A1	score A2
31	0.2	0.4	0.8	183	179
32	0.2	0.4	0.8	179	175
33	0.2	0.4	0.8	181	177
34	0.2	0.4	0.8	179	183
35	0.2	0.4	0.8	179	177
36	0.2	0.2	0.8	177	179
37	0.2	0.2	0.8	181	179
38	0.2	0.2	0.8	183	177
39	0.0	0.2	0.8	183	183
40	0.0	0.2	0.8	183	183

We have tried also with random ghosts and the iterations which we are not going to represent but it has learned a lot about which way to go to (in our case, to eat the ghost) and it performs quite well even with a randomly moving ghost.

labAA2

iteration	epsilon	alpha	gamma	score A1	score A2
1	1	0.8	0.8	-677	-207
2	1	0.8	0.8	-979	-63
3	0.8	0.8	0.8	230	164
4	0.8	0.8	0.8	99	271

5	0.8	0.8	0.8	289	353
6	0.8	0.8	0.8	314	295
7	0.8	0.8	0.8	256	274
8	0.6	0.8	0.8	339	357
9	0.6	0.8	0.8	325	331
10	0.6	0.8	0.8	371	341
11	0.6	0.8	0.8	339	266
12	0.6	0.6	0.8	363	157
13	0.6	0.6	0.8	325	333
14	0.6	0.6	0.8	257	309
15	0.6	0.6	0.8	355	349

iteration	epsilon	alpha	gamma	score A1	score A2
16	0.4	0.6	0.8	343	341
17	0.4	0.6	0.8	371	369
18	0.4	0.6	0.8	365	355
19	0.4	0.4	0.8	358	355
20	0.2	0.4	0.8	381	375
21	0.2	0.4	0.8	377	383
22	0.2	0.2	0.8	381	383
23	0.2	0.2	0.8	377	377
24	0.0	0.2	0.8	383	383
25	0.0	0.2	0.8	383	383
26	0.0	0.0	0.8	383	383
27	0.0	0.0	0.8	383	383

Again, using the appearance of random ghosts and epsilon = 0.0 = alpha, we have obtained the following results:

Score A1: 357, 366, 311,349, 361

Score A2: 335, 334, 276, 358, 359

Which are still pretty high.

labAA3

iteration	epsilon	alpha	gamma	score A1	score A2
1	0.8	0.8	0.8	409	299
2	0.8	0.8	0.8	271	407
3	0.8	0.8	0.8	151	331
4	0.8	0.8	0.8	123	383
5	0.8	0.8	0.8	59	437
6	0.6	0.8	0.8	443	503
7	0.6	0.8	0.8	451	405
8	0.6	0.8	0.8	449	533
9	0.6	0.6	0.8	459	455
10	0.6	0.6	0.8	479	431
11	0.6	0.6	0.8	487	471
12	0.4	0.6	0.8	479	527
13	0.4	0.6	0.8	513	521
14	0.4	0.6	0.8	529	527

iteration	epsilon	alpha	gamma	score A1	score A2
15	0.4	0.4	0.8	495	537
16	0.4	0.4	0.8	481	515
17	0.2	0.6	0.8	517	501
18	0.2	0.6	0.8	513	505
19	0.2	0.4	0.8	537	479
20	0.2	0.4	0.8	519	529
21	0.2	0.4	0.8	541	513
22	0.2	0.2	0.8	523	513
23	0.2	0.2	0.8	527	537
24	0.2	0.2	0.8	529	525

25	0.0	0.2	0.8	539	Problem
26	0.0	0.0	0.8	485	Problem
27	0.0	0.0	0.8	531	Problem

`labAA4`

iteration	epsilon	alpha	gamma	score A1	score A2
1	0.8	0.8	0.8	231	535
2	0.8	0.8	0.8	61	215
3	0.6	0.8	0.8	343	597
4	0.6	0.8	0.8	563	461
5	0.6	0.8	0.8	503	539
6	0.6	0.6	0.8	483	383
7	0.6	0.6	0.8	497	711
8	0.6	0.6	0.8	625	499
9	0.6	0.6	0.8	483	667
10	0.6	0.6	0.8	467	463
11	0.4	0.6	0.8	507	713
12	0.4	0.6	0.8	551	661
13	0.4	0.6	0.8	605	623
14	0.4	0.6	0.8	499	541

iteration	epsilon	alpha	gamma	score A1	score A2
15	0.4	0.4	0.8	593	531
16	0.4	0.4	0.8	517	731
17	0.4	0.4	0.8	599	547
18	0.4	0.4	0.8	537	521
19	0.2	0.4	0.8	125	543

20	0.2	0.4	0.8	357	547
21	0.2	0.4	0.8	515	711
22	0.2	0.4	0.8	595	541
23	0.2	0.2	0.8	485	535
24	0.2	0.2	0.8	499	547
25	0.2	0.2	0.8	507	543
26	0.0	0.0	0.8	465	559
27	0.0	0.0	0.8	421	553
28	0.0	0.2	0..8	383	551

Also trying with random ghosts we have obtained:
Score A1: 512, 518, 540
Score A2: 507, 558, 581

`labAA5`

iteration	epsilon	alpha	gamma	score A1	score A2
1	0.8	0.8	0.8	586	264
2	0.8	0.8	0.8	792	448
3	0.8	0.8	0.8	862	666
4	0.6	0.8	0.8	658	754
5	0.6	0.8	0.8	498	762
6	0.6	0.8	0.8	719	708
7	0.6	0.8	0.8	569	670
8	0.6	0.6	0.8	750	500
9	0.6	0.6	0.8	554	460
10	0.4	0.6	0.8	298	606
11	0.4	0.6	0.8	636	630
12	0.4	0.6	0.8	599	726
13	0.4	0.4	0.8	614	640

14	0.4	0.4	0.8	732	534

iteration	epsilon	alpha	gamma	score A1	score A2
15	0.4	0.4	0.8	626	496
16	0.4	0.4	0.8	729	866
17	0.4	0.4	0.8	664	622
18	0.4	0.4	0.8	735	712
19	0.2	0.4	0.8	810	930
20	0.2	0.4	0.8	804	714
21	0.2	0.4	0.8	789	892
22	0.2	0.4	0.8	512	702
23	0.2	0.2	0.8	257	902
24	0.2	0.2	0.8	problem	644
25	0.2	0.2	0.8	128	626
26	0.0	0.2	0.8	Proble	910
27	0.0	0.0	0.8	Problem	Problem
28	0.0	0.0	0.8	Problem	Problem

RandomGhosts
ScoreA1: 789, 678, 729

RandomGhosts
Score A2: 935, 762, 701

Which one is better?

We have followed the same approach for training the agent using different sets of attributes. The performance of the second one is quite good until we play in labAA4 and labAA5 when we select alpha or epsilon as 0. It sometimes struggles and starts moving in a repetitive way, but it is not able to finish the game .

We also have found some problems with the first agent.Which seem to aggravate in this last map, maybe due to the dimension of the q-tableTherefore, we have selected the second agent. Because of these problems that we have found, and after a lot of iterations, we have realized that the best values for our parameters are alpha = 0.2 and epsilon = 0.2.

Conclusions

In this practice, we have learned how exactly AIs work, how a simple agent can learn its tasks and perform them automatically in the best way possible, using the least amount of time.

It takes a lot of work to achieve it and it is a little time consuming since it must be played over and over again until it learns the best path to take depending on the attributes that are chosen or that are being taken into account.

Overall, it was interesting to see firsthand how an agent can learn by itself and how useful it could be in the long-run. We could make agents that automatically do stuff for us and even better in the least amount of time.

The only con we could give is the time you are consuming into making a lot of different iterations in order for your agent to learn and actually think about the functions that are going to be used, which could take even more time to achieve rather than all of those previously mentioned experiments.

Compared to practice 1, we have spent less time, as in the supervised approach, the phase of gathering the data and preprocessing it, took a lot of time. Although, in reinforcement you also need to spend a lot of time, as you have to train different agents changing the sets, attributes, and functions; we think that Reinforcement learning is a better application in this particular case.