Artificial Intelligence Laboratory 2: Agents
DT8042 HT22, Halmstad University

Fabrice Bodson, Lluís Picornell Company

November 2023

Introduction

The objective of this lab is to practice designing and implementing intelligent agents in two application domains

> **Q1**: What types of agents have you implemented in this lab, and in which two application domains?

The types of agents we have implemented in this lab are:

- Random agent: acts randomly, disregarding any external input.
- Fixed agent: acts based on a sequence of pre-specified actions, disregarding any external input.
- Reflex agent: acts based on the agent's current perception of the world, but not based on past perceptions.
- Agent with memory: acts based on the agent's current perception of the world and past perceptions.

And have implement them in two application domains:

- A mobile robot exploration that consists on a robot that has to move though an environment, has sensors and must collect energy blocs.
- A simple poker game and 4 agents that plays the game. This agents will bet on each hand and the winner will take it all, after 50 rounds we'll have a winner.

Task 1: Mobile robot exploration

For tasks 1a - 1d...

> **Q2**: Briefly explain each agent you have implemented for the mobile robot exploration task. Using diagrams/plots if necessary.

Random agent

For this first agent, we wrote a function named **random_agent()** where we randomly define the direction, movement time and speed of the robot. The movement time and the speed are defined at precise intervals so the robot does not go too fast or too long in the same direction.

Then, depending on the direction selected, the speed is applied to the wheels (to both wheels if it is forward or backward, one of the 2 wheels if it is a turn, etc.).
Finally, this function returns the speed (*motorSpeed*) and movement time (*moving_time*). These two values are retrieved in the main code. The time when the robot should stop the random move (*stop_time*) is calculated based on the current *simulationTime* plus the movement time.

As long as this stopping time is greater than the current *simulationTime*, the robot will move according to the speed of the wheels by executing the following line : *World.execute(motorSpeed, moving_time, -1)*.

Fixed agent

For the second agent, we create a class named **FixedAgent** so we can initialize the time of the 1st sequence and later define the time at which each sequence start. Therefore, when this object is created in the main code, we need to put the *simulationTime*.

In the *move()* method, we take the actual *simulationTime* and substract the start time of the sequence (*self.time_sequence*) to get the difference of time since the beginning of the sequence. So we can compare the time spent for each operation. By doing so, we define a sequence of operations that the robot performs. This makes the robot make a whole round of the maze. When the robot is done, the *self.time_sequence* is set to the current *simulationTime*.

Reflex agent

This third agent was created in a simple function named **reflex_agent**. We first get the sensors for the ultraSonic left and right and the energy sensor. Then we define a basic speed.

The *energy['direction']* is used to get to know where is the nearest energy block located (in which direction). Based on the results it returns, we adjust the robot to go more to the right or more to the left by giving different speed to the wheels : when the block is far from the *right_sensor*), we adjust the robot's position by giving more speed to the right wheel and less speed to the left wheel so it can turn left and get the right sensor closer to the block. Then, if the left sensor gets to far, the same operation is done to get this left sensor closer to the energy block.

Memory agent

The last agent was created with a class named **MemoryAgent** so we can initialize a dictionary used as memory. This class has the method *move()* that uses the same code as the Reflex Agent and the Random Agent. By default it will use the Reflex Agent to go to the nearest block. When one block is collected, its name and the *simulationTime* at which it was collected are saved into the memory. So we get the dictionary *self.memory* with the keys being the blocks name and the value the time they were collected. Then the robot moves as he does in the Reflex Agent.

When at least 1 block was collected (this check is done on line 155 of *model.py*), we get the time at which the last block was collected to see if it was collected more than 10 seconds ago. If it was, then it means the robot can be stuck somewhere or he is going in a wrong place, so we change the strategy to the random move as in the Random Agent. Once the random move has been done, we get back to the Reflex Agent.

Q3: Propose two additional performance measures and compare the performance of all four agents implemented.

Agent Types	Measure 1 #blocks collected in 1min	Measure 2 Time spent to collect all blocks	Measure 3 Time between 2 first blocks
Random	1	+5min	+5min
Fixed	3	+5min	31s
Reflex	2	+5min	16s
Memory	3	+5min	19s

Table 1: Performance comparison

Task 2: Poker game agent

Q4: Explain agents implemented for the poker application (task 2a, 2b, 2e).

The random agent (2a) is an agent that acts randomly, it doesn't "look at" the cards, it just bets a random quantity between 0 and 50.

The fixed agent (2b) is an agent that will always act the same regardless any external input. In this case it will always bet 25.

The reflex agent (2e) will act based on it's current hand, without looking at what the other has done. We are using the function y = 50/39*x being x the power of the hand and y the amount it will be betting.

Q5: Analyse the result of the game with a random agent vs. a fixed agent. Which agent is better? Why?

There is no clear winner for games against this two agents, because both of the agents bet without looking at their hands nor the other agent hand's.
As it's all because of luck each agent will win half of that games played and will get a random amount of coins plus the 25 coins that the fixed always bets.
Most of the games have a score similar to 3200-3400 going to either one. There are some games that there is a clear winner but is only because of luck.

Q6: Does the implemented reflex agent make better decisions based on the strength of its hand?

Yes, it wins most of the games against the random agent and the fixed agent, but if wither of those agents gets "lucky" then they beat the fixed agent.
Also it's a bit more rational, as it bets high when it has a good hand and low when it doesn't.
So yes, you can say that it makes better decisions based on the strength of its hands.

Q7: For 50-hand games, compare all agents implemented on how many coins they have won.

Agent vs.	Random	Fixed	Reflex
Random	± 500	± 200	Reflex + 300
Fixed		± 300	Reflex + 400
Reflex			±100

Table 2: Performance comparison (on amount of coins won)

Q8: (Extra credits) Explain the memory agent you have implemented. Have your poker agent with memory play against the three types of the agents 2a, 2b, and 2e and make it:

1. deduce which type of agent it is playing
2. outplays other agents, by adapting its strategy to exploit the weaknesses of the fixed and reflex agents

We implemented the memory agent. And this is how it performs against the other agents:

Agent vs.	Random	Fixed	Reflex
Memory	+300	+400	+1600

We can see that it does specially well against the Reflex agent, and that is because it's the only one with a clear strategy.
Our Memory agent first deduces against which agent it's playing. To do that stores the first two rounds bets. If in each rounds the other agent did the same then it guesses that it's the Reflex agent.
If in both rounds did exactly the same then it guesses that it's the Fixed agent.
And if in each turn it did a different thing then it guesses that it's the Random agent.

If it's the Random or Fixed then the Memory agent does the Reflex agent strategy, as the bet of the opponent won't help to guess which hand are they playing.
But if it's a Reflex agent then it watches which is the first bet, then it compares it with the bet it would make. If it's greater then it means that it has a better hand, therefore it goes all in.
If it's lower then it means it has a worst hand, therefore it bets 0.

Conclusion

This lab helped us to fully understand these different agents and how to integrate them into different context. Thanks to these contexts, this lab was easier and we enjoyed to do it. The algorithms we wrote could certainly be better and more optimal than what we did.

Summary on the contribution

We split the tasks as Fabrice made the 1st task and Lluis made the 2nd task (and the poker extra credits tasks)