Autonomous Software Agents project

Bozzo Francesco, Izzo Federico

2023-07-11

Contents

1 Agent

In the field of *Computer Science* an *agent* is an individual situated in some environment, and capable of flexible autonoumous action in that environment in order to meet its design objectives. With respect to a more traditional algorithm, there is no need of defining every edge case, a well defined agent should have a reasoning component capable of taking decisions even in unkown situations. The flexibility of an agent can be quantified across two main scales:

- reactive: delay required to respond to a change in the environment;
- proactive: ability of taking action in advance for the maximization of future goals.

Based on the environment and the agent itself there are different communication models but generally the agent perceives observations from the environment using sensors and performs actions against the environment employing actuators.

Another fundamental characteristic of an agent is its autonomy, the internal “brain” should handle the decision process with or without collected information, it should also evolve with respect to possible requirement changes.

Finally an agent can be designed to solve tasks or goals. When we talk about tasks we mean small objectives that must be completed to achieve the final bigger goal. On the other hand we have a goal agent, it takes the goal and it has to define autonomously a list of tasks to fulfill the assigned goal.

1.1 Multi-agent system

A mutli-agent system, as suggested by the name, is a group of agents placed in same environment. There are two main type of interactions: cooperative and competitive. During this project we will focus on both of them in several stages of the development.

A competitive system is composed of many agents acting against each other, the goal can be divided into to sub-goals, maximizing the personal reward and minimize the oppenents’ ones.

A cooperative system is composed of many agents acting to maximize the shared reward, each agent must be capable of cooperate, coordinate and negotiate as much as possible.

In a cooperative system an important implementation detail is the communication, it should be fast and reliable with a the lowest possible delay.

1.2 Architecture

There exist multiple architectures that can be used to build an agent capable of acting in a given environment. We have decided to opt for the architecture described in the following pseudocode

Algorithm 1 Agent control loop

procedure AGENTCONTROLOOP
  $B \leftarrow B_0$ ▷ Belief set initialization
  $I \leftarrow I_0$ ▷ Intention set initialization
  while true **do**
    perceive ρ
    $B \leftarrow$ update(B, ρ) ▷ Belief set update
    $D \leftarrow$ options(B) ▷ Desires computation
    $I \leftarrow$ filters(B, D, I) ▷ Intention update
    $\pi \leftarrow$ plan(B, I) ▷ Plan computation
    while not (empty(π) or succeeded(I, B) or impossible(I, B)) **do**
      $\alpha \leftarrow hd(\pi)$ ▷ Get next action
      execute(α) ▷ Execute action
      $\pi \leftarrow$ tail(π) ▷ Remove executed action
      perceive ρ
      $B \leftarrow$ update(B, ρ) ▷ Belief set update
      if reconsider(I, B) **then**
        $D \leftarrow$ options(B) ▷ Desires computation
        $I \leftarrow$ filters(B, D, I) ▷ Intention update
      end if
      if not sound(π, I, B) **then**
        $\pi \leftarrow$ plan(B, I) ▷ Plan computation
      end if
    end while
  end while
end procedure

A belief set B is udpated after every environment communication, it is then used to compute the desires set D which is filterd to create the intentions set I. Finally the intentions set is used in combination with the belief set to create a plan π to satisfy all the intensions. The plan is then execute action by action α. For every new information collected during the exectution the belief set, the intentions set, and the desires set are updated. After the update a replan can be done, during this phase a new plan is computed.

2 Deliveroo

Deliveroo is the environement provided to test the developed agents of the project, it is composed of a grid of tiles divided into two main categories: walkable (green and red) and not walkable (black). A walkable tile can be a delivery zone (red).

Several parcels, represented by cubes, are scattered around the map, the goal of the agent is to move across it collecting the largest number of parcels. The final objective is to deliver collected elements in one delivery zone.

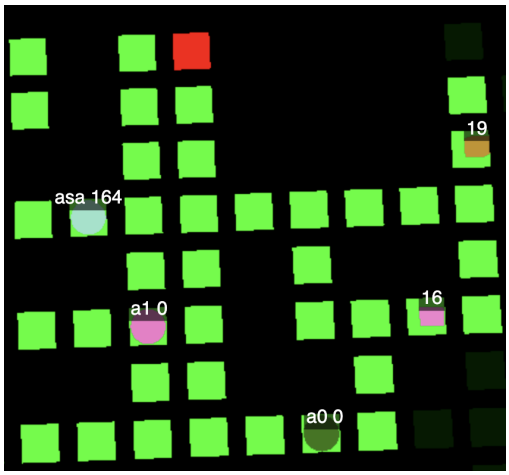

Figure 1: An example of grid map with different parcels and agents.

The agent can move up, down, left, and right. It can pass through parcels and delivery zones but it is blocked by other agents that can play the game at the same time.

Multiple parcels can be carried by one agent, and the value of them continues to decrease

The agent can sense other agents and parcels, respectively within two well defined radiuses.

There are several parameters associated to the environment associated to various game elements:

- *parcel*: generation internal, reward decay, maximum number, and variance;
- *player movement*: steps number and speed;
- *agent*: amount and speed;
- *sensing*: parcel and agent sensing radius;
- *game clock*.

Given all the customizable parameters and the size of the map, the number of different combinations is huge. An agent must be well developed in order to be able to act in all of them, both alone or cooperating with other agents.

There are two main strategies that can be employed for the solution of the game:

- *single-agent*: the agent acts alone;
- *multi-agent*: the agent communicates with other mates to share information for the maximization of the summuation of all the rewards;

3 Problem Analysis

Given the complexity of the environment several solutions have been implemented to overcome different difficulties and challenges.

3.1 Manhattan distance

The *Manhattan distance* is function that can provide a quick estimation of the distance between two points.

$$d_{ab} = |b_x - a_x| + |b_y - a_y|$$

In the deliveroo environment it is insufficient since many tiles can be non walkable, but at the same time it's fast heruistic to compute.

3.2 Problem parameters estimation

As explained in Section 2, multiple parameters can be modified during the game initialization. Some of them are given to the agent, others are obscure and can be only estimated. We mainly focused on the estimation of player speed and parcel decay for a more accurate computation of the potential reward associated to a given parcel.

3.2.1 Player speed

Each time the agent moves the new position is communicated by the environment to the agent, we have decided to keep track of the timestamp in which the information is perceived. The list of timestamps can then be used to estimate the player velocity using the following algorithm:

Algorithm 2 Player speed estimation

procedure UPDATEMAINPLAYERSPEEDESTIMATION($\mathcal{D}, s, \phi$)
  deltas $\leftarrow []$ ▷ Initialize an empty array of deltas
  for each timestamp $t_i \in \mathcal{D}$ **do**
    $deltas.append(t_i - t_{i-1})$ ▷ Delta of two consecutive timestamps
  end for
  $c \leftarrow s * (1 - \phi)$ ▷ Current speed contribution
  $n \leftarrow \text{avg}(deltas) * \phi$ ▷ New speed contribution
  $s \leftarrow c + n$ ▷ New estimation
  return c
end procedure

The hyper-parameter ϕ is the learning rate that can be used to regulate the impact of the contribution with respect to the current speed estimation.

3.2.2 Parcel decay

Another important parameter for the correct definition of an agent is the parcel decay, it is the number velelocity of the reward decrease. Similarly to the player speed estimation, the parcel decay estimation is computed from timestamp differences, where a timestamp is associated to an update of a visible parcel.

Algorithm 3 Get parcel decay estimation

procedure GETPARCELDECAYESTIMATION($\mathcal{D}$)
    deltas $\leftarrow$ [] ▷ Initialize an empty array of deltas
    for each timestamp $t_i \in \mathcal{D}$ **do**
        $deltas.append(t_i - t_{i-1})$ ▷ Delta of two consecutive timestamps
    end for
    return deltas
end procedure

Algorithm 4 Parcels decay estimation

procedure UPDATEPARCELSDECAYESTIMATION($\mathcal{P}, d, \phi_2$)
    deltas $\leftarrow$ [] ▷ Initialize an empty array of deltas
    for each parcel $p_i \in \mathcal{P}$ **do**
        $deltas.concat(getParcelDecayEstimation(p.timestamps))$ ▷
    end for
    $c \leftarrow d * (1 - \phi_2)$ ▷ Current decay contribution
    $n \leftarrow$ avg$(deltas) * \phi_2$ ▷ New decay contribution
    $d \leftarrow c + n$ ▷ New estimation
    return d
end procedure

The learning rate ϕ_2 can be used to regulate the impact of the contribution with respect to the current parcel decay estimation.

3.3 Probabilisitic model

In the environment there may be multiples competitive agents, their ability of picking up parcels highly infuences the value of a parcel. For this reason we have devised a penalty value based on a probabilistic model capable of takimg into consideration the possible opponents' plans.

The main idea behind the probabilistic model is the following assertion: if there is a parcel and I am the closest agent I can reach it faster than any other agents, consequentially that parcel should be taken more into consideration, even if its value is lower than other further parcels.

This assertion can be modeled as follow

$$\text{penalty probability} = \frac{\sum_{a \in \mathcal{A}} \frac{d_{max} - d_{pa}}{d_{max}}}{|A|}$$

with $\mathcal{A}$ the set of opponent agents, d_{max} the maximum distance betweent the parcel and the union between oppoents agents, main player, and cooperative agents, d_{pa} the distance parcel oppenent agent.

3.4 Potential parcel score

The decision process behind the choice of a parcel is one of the key elements for the definition of a good agent. Many elements and metrics have been taken into consideration to better estimate the potential gain of a parcel. The final reward is computed as:

$$r_f = r - \left(d_{ap} * \frac{s_a}{decay}\right) - \left(d_{min} * \frac{s_a}{decay}\right) - r * \text{penalty probability}$$

where d_{ap} is the distance between the agent and the parcel, s_a is the estimatated speed, d_{min} is the minimum distance between the parcel and any delivery zone, and penalty probability is the probability computed on Section 3.3.

The resulting formula takes into reward the lost of approaching the parcel and delivery it to the closest delivery zone, moreover the probabilistic model can provide a rough estimation of other agents' intentions.

3.5 Distances cache

A distance cache was maintained to save some computational power, every time a plan is computed the distance between the starting point and any other tile in the path is saved in the cache.

The cache is then used in many parts of the code.

3.6 Replan

Qui non so se per ora il replan dopo 5 tentativi sia interessante da menzionare.

4 Solutions

Qui spieghiamo le varie soluzioni presentate.

Forse questo capitolo e' ridondante, non saprei.

5 Benchmarking

6 Conclusion

7 References