AASMA 2017 - Pig Chase in Minecraft

Luís Sá Couto
ist1xxxxx
xxxxx@tecnico.ulisboa.pt

Telma Correia
ist1xxxxx
xxxxx@tecnico.ulisboa.pt

Gonçalo Rodrigues
ist1xxxxx
xxxxx@tecnico.ulisboa.pt

ABSTRACT

With this paper we intend to present a possible approach to solve the Pig Chase Challenge proposed by Microsoft in [1].

The approach appears in the context of the Multi-Agent Systems course of the Masters degree in Computer Science in Instituto Superior Técnico.

Our solution is based on several concepts learned on the said course, such as agent's architectures, game theory, oppositional strategies and also machine learning.

1. INTRODUCTION

The challenge is a classical cooperation vs. competition dilemma with previously known utilities. In it two agents try to maximize their reward by either fleeing the scene or by working together to catch a pig.

Several types of issues rise when we first look at this problem. The five main ones are 1) choosing an architecture 2) choosing the initial strategy, 3) observing the opponent's behavior, 4) using it to adapt the strategy and 5) planning the movement in order to maximize the potential gain.

Such issues are a relatively complete set of problems to solve and the goal of this paper is to guide the reader through a structured approach to solve each one. The next section focuses on formally describing the environment and the evaluation set, while the following section is divided into five subsections in which each of the previously mentioned problems are solved.

2. THE CHALLENGE

Microsoft's challenge lays on a setting of a pigsty with a fence which has two open doors and where exists a pig, two agents and four obstacles.

The main goal of the game is to catch the pig by trapping it into a position without a free adjacent cell. It is only possible to achieve such state with the collaboration of both agents.

In order to explain this paper's proposed approach, we must clarify two key subjects on which the next two subsection focus. The next subsection presents a formal description of the environment in terms of its properties, states, transitions and reward (or utilities). Afterwards the rules of the competition and evaluation setting are explained.

2.1 Environment

We present Figure 1 as a possible state of the environment to provide the reader a mental picture of the subject of this section. The environment is composed by a grid with 21 walkable positions. Where both agents and pig move, one position at a time.

2.1.1 States

In this environment, the state is characterized by the pig's position and the two agent's position in the grid. So it can be represented by a sextuple:

$$s_t = \{Xagent_t, Yagent_t, Xopponent_t, Yopponent_t, Xpig_t, Ypig_t\}$$

Where the x axis ranges from 0 to 6 and the y axis ranges from 0 to 4 given that wall positions and obstacle positions are unavailable.

2.1.2 Actions

In each state, the agents can choose to **rotate right (rr)**, to **rotate left (rl)** and to move **forward (f)**. So the action space is given by:

$$A = \{rr, rl, f\}$$

2.1.3 State Transition Function

The transition function depends on the actions of both agents and the pig's. Hence:

$$\tau : s_t, Aagent_t, Aopponent_t, Apig_t \rightarrow s_{t+1}$$

We must also denote that the actions don't have noise, which means that they never fail, but whenever an agent moves towards an obstacle or a wall, it stays in the same position.

2.1.4 Rewards

The final game reward is defined in terms of the final state of one run plus the number of time steps it took to reach that run.

Either both agents receive a $+25$ reward whenever the environment is in a state where the pig has no free adjacent cells, or one of the agents receives a $+5$ reward (while the other gets nothing) for reaching one of the pigsty's doors.

Also, both agents are penalized with -1 reward for each time step that goes by before the run ends.

2.1.5 Properties

According to Wooldridge [6] an environment can be classified according to five key properties. To ease the understanding of the environment itself, we provide a classification for each of the mentioned properties:

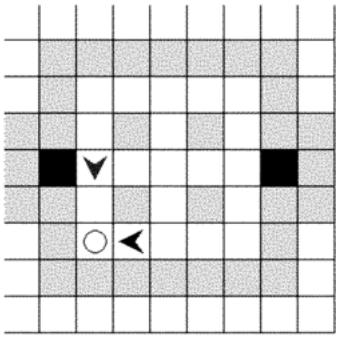

Figure 1: A possible state of the environment. Black squares are the doors, grey squares are obstacles and walls.

1. **Stochastic:** since the agent only controls its own action, if an agent acts in a certain way there is no guarantee that the environment will end up in a given state, because this transition depends also on the competitor's action and the pig movement.

2. **Fully Observable:** at any point in time, the agent can see the full state of the environment. In practice, the agent has full access to the world's current map, knowing the exact position of every object, including itself.

3. **Dynamic:** Although it is true that the agents wait for their turn to act. The pig can move at any moment since it is supported on the Minecraft Engine. This implies that the environment can change during the deliberation process.

4. **Discrete:** since the pigsty is described by a grid, we can conclude that there is a finite number of states in this problem.

5. **Non-Episodic:** despite the fact that every time the game ends (i.e. one of the agents leaves the pigsty or that the pig is caught) the whole environment resets, the agent can learn from one game (a possible episode) to the next one, which means that there is no independence between episodes.

2.2 Evaluation Setting

Microsoft defined the competition in a way which every agent will face each other. Whenever two agents face their final rewards is store. In the end, the winner will be the agent with the highest cumulative reward.

This kind of setting doesn't allow the agent to tune itself to a specific kind of opponent. It urges the agent to be adaptive and flexible, both in deciding the initial strategy to use against an unknown opponent and in being able to rapidly to study and adapt to an opponent during their face-off.

All of these considerations will be taken into account in the several decisions made in the next section.

3. THE APPROACH

This section focuses on explaining the approach each one of the five main issues that the challenge presents. We start by defining an architecture for the agent and then we define the reasoning and planning necessary to solve this problem.

3.1 Choosing an Architecture

Under the uncertainty of the opponent's and the pig's actions the problem requires an architecture that provides a way to express probabilistic beliefs about other's intentions and to formally model the practical reasoning issues.

So, we developed our approach in terms of a BDI architecture as defined in [3]. This model allows us to define an agent that can deliberate about the observations of the world, in order to choose the best plan to accomplish goal-oriented behavior.

We finish this section by specifying what is, in this context, a belief, a desire and an intention.

3.1.1 Beliefs

Our approach deals with second order beliefs that reflect how much the agent believes that the opponent intends to either collaborate on catching the pig or to flee the scene.

To formally describe the belief, we must first define the random variable OI which stores the opponent's possible intentions as follows:

$$OI = [Cooperate, Flee]$$

So, the belief can be represented as a distribution of probabilities for the random variable OI as follows:

$$B = [P[OI = Cooperate]P[OI = Flee]]$$

3.1.2 Desires and Intentions

In this context, to ease understanding, we model the problem in a way that the desires space and the intentions space are identical, as follows:

D = { Go to door 1, Go to door 2, Go to Pig Adjacent Position 1, Go to Pig Adjacent Position 2, ..., Go to Pig Adjacent Position n }

I = { Go to door 1, Go to door 2, Go to Pig Adjacent Position 1, Go to Pig Adjacent Position 2, ..., Go to Pig Adjacent Position n }

These desires and intentions detail the agent's desire and intention to cooperate or not into a specific world position.

Algorithm 1 presents the pseudo-code that resumes the practical reasoning process of our agent. For the next sections we set the goal of describing each of the remaining functions used in the algorithm, their purpose and how they influence the agent's behavior.

3.2 The Initial Strategy

If we look at the game itself as a "black box", we can look at it as a kind of game theory dilemma, where both agents have to choose between cooperate and flee.

It is also necessary to add that there is some uncertainty in the result of the actions. First of all, the rewards depend on the number of steps because both agents are penalized with a -1 reward for each step. Second of all, even if both agents decide to flee, only one will manage to flee first.

We can express the reward of a game with a typical game theory payoff matrix (see table 1).

With that said, it is important to approach the problem of deciding the initial strategy isolating the problem of the number of steps to the next sections.

Table 1: Payoff Matrix of a Game where nsteps is the number of time steps since the beginning of the game until the end

		Opponent: *Cooperate*	Opponent: *Flee*
Agent	*Cooperate*	$(25 - nsteps, 25 - nsteps)$	$(-nsteps, 5 - nsteps)$
	Flee	$(5 - nsteps, -nsteps)$	$(5 - nsteps, -nsteps)$ or $(-nsteps, 5 - nsteps)$

Algorithm 1 Beliefs, Desires, Intentions

```
1: (B,I) ← getInitialStrategy();
2: while true do
3:     x ← getWorldState();
4:     B ← updateBeliefs(B, x);
5:     D ← options(B, I);
6:     I ← filter(B, D, I);
7:     π ← plan(B, I);
8:     while not(succeed(I, B)orimpossible(I, B)) do
9:         β ← first(π);
10:        execute(β);
11:        π ← rest(π);
12:        x ← getWorldState();
13:        B ← updateBeliefs(B, x);
14:        if reconsider(I, B) then
15:            D ← options(B, I);
16:            I ← filter(B, D, I);
17:        if sound(π, I, B) then
18:            π ← plan(B, I);
```

Since the competition is set in a way where every agent plays against each other several times in a random way, the goal here is to learn the initial strategy that yields, on average, the best results. To this end, we set our agent's $getInitialStrategy()$ function to be learned with experience.

For learning, we applied an individual learner technique, with a Q-learning algorithm as described in [4]. The goal is to learn along the several games what is the best (in average) belief, and as a consequence of it, the strategy for the agent to start the game.

This approach has some shortcomings, because it is applied to a game where there are no clear Nash equilibriums and, hence, no convergence is guaranteed. But we accept this limitation, because what is expected is a merely indicative average strategy, expressed through an initial belief.

3.3 Observing the Opponent

This section focuses on presenting our model's way to address the problem of updating the beliefs.

The $updateBeliefs(B, x)$ function's (see Algorithm 1) goal is to update our agent's internal state after observing the opponent's last movement.

Since each step punishes both agents, we assume that, in the general case, the opponents will try to follow their strategies using the shortest possible paths in the environment, in order to minimize such punishment.

In each time step we compute the opponent's shortest path to the pig and to both doors (using an a-star search algorithm as presented in [2]). This way we can infer an optimal policy for each of the opponent's possible intentions.

We represent the policies as follows:

$$policies(t) = \{\pi_{t_{door1}}, \pi_{t_{door2}}, \pi_{t_{pig}}\}$$

At each time step t we use the policies from the previous step ($policies(t-1)$) to analyze the opponent's behavior in the following way with α defined as the learning rate:

1. **if** $Aopponent_t$ **follows** $\pi_{t_{door1}}$**:** the belief that the opponent intends to flee is strengthened (by summing a vector with weight α and normalizing):

$$B_{t+1} = updateBeliefs(B_t, x_t) = \frac{\begin{bmatrix}0 & \alpha\end{bmatrix} + B_t}{||\begin{bmatrix}0 & \alpha\end{bmatrix} + B_t||}$$

2. **if** $Aopponent_t$ **follows** $\pi_{t_{door2}}$**:** the belief that the opponent intends to flee is strengthened (by summing a vector with weight α and normalizing):

$$B_{t+1} = updateBeliefs(B_t, x_t) = \frac{\begin{bmatrix}0 & \alpha\end{bmatrix} + B_t}{||\begin{bmatrix}0 & \alpha\end{bmatrix} + B_t||}$$

3. **if** $Aopponent_t$ **follows** $\pi_{t_{pig}}$**:** the belief that the opponent intends to cooperate is strengthened (by summing a vector with weight α and normalizing):

$$B_{t+1} = updateBeliefs(B_t, x_t) = \frac{\begin{bmatrix}\alpha & 0\end{bmatrix} + B_t}{||\begin{bmatrix}\alpha & 0\end{bmatrix} + B_t||}$$

4. **Otherwise:** The belief stays the same:

$$B_{t+1} = updateBeliefs(B_t, x_t) = B_t$$

The learning rate can't be too small because the environment is rapidly changing and the agent must adapt quickly. The problem of a high learning rate is that it can leave the agent more susceptible to "bluffers" and, also, more undecided because the belief can change drastically at every time step.

3.4 Adjusting the Strategy

This section looks over the problem of choosing an intention and the level of commitment to it.

To our approach, regardless of the current belief and intention, all desires can make sense at any time. Thus, the *options* function is constant and alway generates all of the desire space:

1. Go to door 1
2. Go to door 2
3. Go to Pig Adjacent Position 1
4. Go to Pig Adjacent Position 2
5. ...
6. Go to Pig Adjacent Position n

In the chosen architecture, the last step in the way to choose an intention is to use the $filter$ function to calculate from all possible agent's desires what is the intention. This function stores some domain specific knowledge that enriches the agent's behavior and works as follows:

1. The current belief affects the $filter$ function in an $\epsilon - greedy$ [5] way, as follows:

   If at a given time step t the agents has the belief:

   $$B_t = [P[OI = Cooperate]P[OI = Flee]] = \begin{bmatrix}\theta & 1-\theta\end{bmatrix}$$

   Than it will assume:

   (a) with a probability of θ, that the opponent wants to cooperate.

   (b) with a probability of $1 - \theta$, that the opponent wants to flee.

2. After deciding on the opponent's intentions, the $filter$ function works as follows:

   (a) If the agent assumes that the opponent wants to flee, the agent will:

      i. Go to $door_n$ if it is closer to it than the opponent is to both of the doors.

      ii. Go to Pig Adjacent Position n (random n) otherwise.

   (b) If the agent assumes that the opponent wants to cooperate, the agent will:

      i. If the pig has more than two adjacent positions, then it is not catchable so the agent will:

         A. Go to $door_n$ if it is closer to it than the opponent is to both of the doors.

         B. Go to Pig Adjacent Position n otherwise (where n is not the goal position of the optimal policy that guides the opponent to a pig adjacent position).

      ii. Go to Pig Adjacent Position n otherwise (where n is not the goal position of the optimal policy that guides the opponent to a pig adjacent position).

Having the intention, the commitment to it must be established. In our model, that commitment is expressed in the *reconsider* function (see Algorithm 1).

Since our agent will live in a rapidly changing environment, it is fundamental to reconsider intentions at every time step, so the agent can adapt to the opponent's strategy. So, the *reconsider* function is defined as follows:

$$reconsider(I, B) = true$$

3.5 Planning the Movement

This section looks over the problem of planning towards an intention and the level of commitment to such plan.

So, we clarify the roles of the functions:

$$plan(B, I)$$

and

$$sound(\pi, I, B)$$

Since every step is penalized with a -1 reward, we find it crucial that our agent moves through the shortest possible parts in most cases.

To achieve this behavior, at each time step, we can calculate the shortest paths from the agent to the pig and to both doors, depending on the intention, by using an a-star algorithm [2]. This way, the agent behaves using the *plan* function as follows:

1. Find shortest path to intended position.

2. Store path in a list of actions.

3. Return the list of actions as the optimal policy for the intention.

Having the means-ends reasoning settled, the level of commitment to a plan must be established. Since our agent reconsiders at every time step, the agent will never commit to a plan for more than one step. This way, the function *sound* has no impact on the solution.

4. RESULTS

In order to validate our approach, several experiments were conducted with several kinds of agents. Each of the following subsections focuses on one of each of those face-offs with other agents.

4.1 Vs. Random Agent

fff

4.2 Vs. Focused Always Cooperate Agent

fff

4.3 Vs. Focused Always Flee Agent

fff

4.4 Vs. Focused TIT-FOR-TAT Agent

fff

4.5 Vs. Itself

fff

4.6 Vs. Human Opponent

fff

5. CONCLUSIONS

First, we believe that our approach gathers a set of knowledge that crosses-over several types of single-agent and multi-agent results and algorithms. Also, The model is based both on strong scientific background taught in the course, but also on some domain specific knowledge that enriches the performance. Last but not least, the approach proved to be strong enough in terms of results to enter the competition.

REFERENCES

[1] The malmo collaborative ai challenge.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[3] A. S. Rao, M. P. Georgeff, et al. Bdi agents: From theory to practice. In *ICMAS*, volume 95, pages 312–319, 1995.

[4] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[5] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[6] M. Wooldridge and M. J. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.