AASMA 2017 - Pig Chase in Minecraft

Competition Team Name: AASMA

Course Group Number: 20

Luís Sá Couto
ist179078

Telma Correia
ist178572

Gonçalo Rodrigues
ist178958

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, the two agent's position in the grid and their orientations. We also store the opponent's past move to make some reasoning about its intentions. So, the state can be represented as follows:

$$\begin{aligned} s_t = \{ & Xagent_t, \\ & Yagent_t, \\ & OrientationAgent_t, \\ & Xopponent_t, \\ & Yopponent_t, \\ & OrientationOpponent_t, \\ & Xpig_t, \\ & Ypig_t \\ & Aopponent_{t-1}\} \end{aligned}$$

Where the x axis ranges from 0 to 6 and the y axis ranges from 0 to 6 given that wall positions and obstacle positions are unavailable, so:

$$\begin{aligned} & Xagent_t, Xopponent_t, Xpig_t \in X \\ & X = \{0, 1, 2, 3, 4, 5, 6\} \\ & Yagent_t, Yopponent_t, Ypig_t \in Y \\ & Y = \{0, 1, 2, 3, 4, 5, 6\} \end{aligned}$$

The orientations assume that agent always rotates 90 degrees, so they can be represented as follows:

$$\begin{aligned} & OrientationAgent_t, OrientationOpponent_t \in Orientations \\ & Orientations = \{Up, Down, Left, Right\} \end{aligned}$$

The positions of the doors never change and can be represented as follows:

$$\begin{aligned} Door1 &= (0, 3) \\ Door2 &= (6, 3) \end{aligned}$$

All other positions where $x = 0$, or $x = 6$, or $y = 0$, or $y = 6$ represent the walls.

The obstacles' positions also never change and can be represented as follows:

$$\begin{aligned} Obstacle1 &= (2,2) \\ Obstacle2 &= (2,4) \\ Obstacle3 &= (4,2) \\ Obstacle4 &= (4,4) \end{aligned}$$

From the set of walkable positions, the subset where the pig is never catchable is the set whose positions are have more than two adjacent walkable positions. Said set of positions can be represented as follows:

$$notCatchable = \{(1,3),(3,1),(3,3),(3,5),(5,3)\}$$

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:

$$\tau : s_t, Aagent_t, Aopponent_t \rightarrow s_{t+1} = \Phi(S)$$

Although the actions never fail, we can't predict the pigs movement since it can be any. This behavior introduces noise to the actions because the same actions can lead to different states. Which means that the transition function yields a probability distribution Φ over the state space.

We also note that 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:

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.

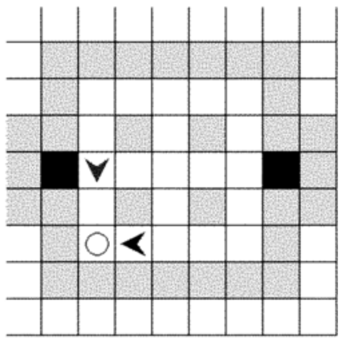

Figure 1: A possible state of the environment. Black squares are the doors, grey squares are obstacles and walls.

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 desires to either collaborate on catching the pig or to flee the scene.

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

$$OD = \{ChasePig, GoToDoor1, GoToDoor2, Random\}$$

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

$$B = \begin{bmatrix} P[OD = ChasePig] \\ P[OD = GoToDoor1] \\ P[OD = GoToDoor2] \\ P[OD = Random] \end{bmatrix}$$

3.1.2 Desires and Intentions

In this context, to ease understanding, we model the problem in a way where the desires space is composed by the three possible desires of our agent, as follows:

$D = \{$

$$ChasePig,$$
$$GoToDoor1,$$
$$GoToDoor2$$

$\}$

The intentions space models the specific positions that can be chosen from each desire and can be represented as follows:

$I = \{$

$$GoToDoor1,$$
$$GoToDoor2,$$
$$GoToPigAdjacentPosition1,$$
$$GotoPigAdjacentPosition2,$$
$$GotoPigAdjacentPosition3,$$
$$GotoPigAdjacentPosition4$$

$\}$

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

Algorithm 1 [3] 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.

Algorithm 1 Beliefs, Desires, Intentions [3]

```
1: $(B,I) \leftarrow getInitialStrategy()$;
2: while $true$ do
3:     $s \leftarrow getWorldState()$;
4:     $B \leftarrow updateBeliefs(B, s)$;
5:     $D \leftarrow options(B, I, s)$;
6:     $I \leftarrow filter(B, D, I, s)$;
7:     $\pi \leftarrow plan(B, I)$;
8:     while not($succeed(I, B)$ or $impossible(I, B)$) do
9:         $\beta \leftarrow first(\pi)$;
10:        $execute(\beta)$;
11:        $\pi \leftarrow rest(\pi)$;
12:        $s \leftarrow getWorldState()$;
13:        $B \leftarrow updateBeliefs(B, s)$;
14:        if $reconsider(I, B)$ then
15:            $D \leftarrow options(B, I, s)$;
16:            $I \leftarrow filter(B, D, I, s)$;
17:        if $sound(\pi, I, B)$ then
18:            $\pi \leftarrow plan(B, I)$;
```

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.

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.

The estimated reward for each of the two possible actions ($Cooperate$ or $Flee$), yelds a Q-vector that can be updated, at the end of each episode, as follows:

$$Q(s_t, a_t) =$$
$$Q(s_t, a_t) + \eta * (reward_{t+1} + \gamma * \max Q(s_{t+1}, a_t) - Q(s_t, a_t))$$

Assuming that the next opponent is not likely to be a random agent. We can take the Q-vector and compute the initial strategy as follows:

$$B_0 = \begin{bmatrix} \frac{Q(s_t,Cooperate)}{Q(s_t,Cooperate)+Q(s_t,Flee)} \\ \frac{Q(s_t,Flee)}{2(Q(s_t,Cooperate)+Q(s_t,Flee))} \\ \frac{Q(s_t,Flee)}{2(Q(s_t,Cooperate)+Q(s_t,Flee))} \\ 0 \end{bmatrix}$$

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.

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

3.3 Observing the Opponent

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

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

At time step t, we can define a set R that describes the opponent's run in the last n actions (in terms of state-action pairs):

$$R_t = \{(s_{t-n}, a_{t-n}), (s_{t-(n-1)}, a_{t-(n-1)}), ..., (s_{t-1}, a_{t-1})\}$$

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.

At 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 desires.

We represent the policies as follows:

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

At each time step t we could also have access to the policies from the last three steps ($policies(t-n)$, ..., $policies(t-2)$, $policies(t-1)$) to analyze the opponent's behavior.

We can infer the opponent's desire as the conditional probability of R given each desire's policy, as follows (where γ provides a discount factor to give more weight to more recent decision):

$$P(R|ChasePig) = \prod_{i=0}^{n-1} \frac{1}{\gamma^i} * \pi_{t_{pig}}(s_{t-(n-i)}, a_{t-(n-i)})$$

$$P(R|GoToDoor1) = \prod_{i=0}^{n-1} \frac{1}{\gamma^i} * \pi_{t_{door1}}(s_{t-(n-i)}, a_{t-(n-i)})$$

$$P(R|GoToDoor2) = \prod_{i=0}^{n-1} \frac{1}{\gamma^i} * \pi_{t_{door2}}(s_{t-(n-i)}, a_{t-(n-i)})$$

The previous set of equations provides a way to evaluate the likelihood of the opponent having each of the desires.

If all likelihoods are equal to zero, than we assume that the agent is acting randomly. Otherwise we assume the more likely desire (i.e. the one with the highest likelihood). So the opponent's desire can be extracted as follows:

1. If we have:

$$\begin{aligned} P(R_t|ChasePig) &= \\ &= P(R_t|GoToDoor1) \\ &= P(R_t|GoToDoor2) \\ &= 0 \end{aligned}$$

Then,

$$OponentDesire = Random$$

2. Otherwise:

$$OponentDesire = argmax_{d \in OD}\{P(R_t|d)\}$$

After computing the opponent's desire, with α defined as the learning rate, we can update the belief by summing the learning rate to the respective desire and normalizing.

We note that, as referred in section 1, we only store the last opponent's action. This happens because in a rapidly changing environment like this one, the history has very low relevance when compared to the last action (i.e. historical actions must be weighted by a very small - approximately zero - discount factor).

1. **if** $OponentDesire_t = ChasePig$**:**

$$B_{t+1} = updateBeliefs(B_t, s_t) = \frac{\begin{bmatrix} B_t(ChasePig) + \alpha \\ B_t(GoToDoor1) \\ B_t(GoToDoor2) \\ B_t(Random) \end{bmatrix}}{\alpha + \sum_{d' \in OD} B_t(d')}$$

2. **if** $OponentDesire_t = GoToDoor1$**:**

$$B_{t+1} = updateBeliefs(B_t, s_t) = \frac{\begin{bmatrix} B_t(ChasePig) \\ B_t(GoToDoor1) + \alpha \\ B_t(GoToDoor2) \\ B_t(Random) \end{bmatrix}}{\alpha + \sum_{d' \in OD} B_t(d')}$$

3. **if** $OponentDesire_t = GoToDoor2$**:**

$$B_{t+1} = updateBeliefs(B_t, s_t) = \frac{\begin{bmatrix} B_t(ChasePig) \\ B_t(GoToDoor1) \\ B_t(GoToDoor2) + \alpha \\ B_t(Random) \end{bmatrix}}{\alpha + \sum_{d' \in OD} B_t(d')}$$

4. **if** $OponentDesire_t = Random$**:**

$$B_{t+1} = updateBeliefs(B_t, s_t) = \frac{\begin{bmatrix} B_t(ChasePig) \\ B_t(GoToDoor1) \\ B_t(GoToDoor2) \\ B_t(Random) + \alpha \end{bmatrix}}{\alpha + \sum_{d' \in OD} B_t(d')}$$

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.

In our approach, we use the *options* function to generate the possible desire giving a specific perception and the current belief. Thus, the *options* function stores some domain specific knowledge that enriches the agent's behavior and works as follows:

1. The current belief affects the *options* function in a kind of $\epsilon - greedy$ [5] way, as follows:

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

   $$B_t == \begin{bmatrix} P[OD = ChasePig] \\ P[OD = GoToDoor1] \\ P[OD = GoToDoor2] \\ P[OD = Random] \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

   Than it will assume:

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

   (b) with a probability of $b + c$, that the opponent wants to flee.

   (c) with a probability of d, that the opponent is behaving randomly.

2. After deciding on the opponent's desires, the *options* 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. $ChasePig$ 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. $ChasePig$ otherwise.

      ii. $ChasePig$ otherwise.

A more detailed pseudo-code for the *options* function can be seen in Algorithm 2, where the *distance* function between two positions represents the minimum number of time steps needed to go from one position to the other (so it includes rotations).

In the chosen architecture, the last step in the way to choose an intention is to use the $filter$ function to calculate from the agent's desire what is the intention (i.e. the specific position to go to). This function also stores some domain specific knowledge that enriches the agent's behavior and works as follows:

1. If the agent has the desire $GoToDoor1$ than it will:

   (a) Have the intention $GoToDoor1$.

2. If the agent has the desire $GoToDoor2$ than it will:

   (a) Have the intention $GoToDoor2$.

Algorithm 2 Options Function Described in Pseudo-code

```
function OPTIONS(B,I,$s$)
    $door1 \leftarrow (0,2)$;
    $door2 \leftarrow (6,2)$;
    $opponent \leftarrow (s.Xopponent_t, s.Yopponent_t)$;
    $agent \leftarrow (s.Xagent_t, s.Yagent_t)$;
    $pig \leftarrow (s.Xpig_t, s.Ypig_t)$;
    $opponentDesire \leftarrow \epsilon - greedy(B)$;
    if $opponentDesire == Flee$ then
        $agentToDoor1 = distance(agent, door1)$;
        $agentToDoor2 = distance(agent, door2)$;
        $opponentToDoor1 = distance(opponent, door1)$;
        $opponentToDoor2 = distance(opponent, door2)$;
        if $agentToDoor1 < opponentToDoor1$ then
            if $agentToDoor1 < opponentToDoor2$ then
                return GoToDoor1
        if $agentToDoor2 < opponentToDoor1$ then
            if $agentToDoor2 < opponentToDoor2$ then
                return GoToDoor2
        return ChasePig
    else
        if $isCatchable(pig)$ then
            return ChasePig
        $agentToDoor1 = distance(agent, door1)$;
        $agentToDoor2 = distance(agent, door2)$;
        $opponentToDoor1 = distance(opponent, door1)$;
        $opponentToDoor2 = distance(opponent, door2)$;
        if $agentToDoor1 < opponentToDoor1$ then
            if $agentToDoor1 < opponentToDoor2$ then
                return GoToDoor1
        if $agentToDoor2 < opponentToDoor1$ then
            if $agentToDoor2 < opponentToDoor2$ then
                return GoToDoor2
        return ChasePig
```

3. If the agent has the desire $ChasePig$ than it will:
    (a) Have the intention $GoToPigAdjacentPositionN$ (where N is not the goal position of the optimal policy that guides the opponent to a pig adjacent position).

A more detailed pseudo-code for the *options* function can be seen in Algorithm 3.

Algorithm 3 Filter Function Described in Pseudo-code

```
function FILTER(B,D,I,$s$)
    $agent \leftarrow (s.Xagent_t, s.Yagent_t)$;
    $pig \leftarrow (s.Xpig_t, s.Ypig_t)$;
    if $D == GoToDoor1$ then
        return GoToDoor1
    if $D == GoToDoor2$ then
        return GoToDoor2
    $pathOpponentToPig = last(policies(s))$;
    $opponentGoal = last(pathOpponentToPig)$;
    $pigAdjacents = freeAdjacent(pig)$;
    $pigAdjacents.remove(opponentGoal)$;
    return $chooseClosest(pigAdjacents, agent)$;
```

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

The approach used in Microsoft challenge doesn't care about learning a specific opponent's strategy from one episode to the next one because opposing agents change every round and, for that reason, our agent resets the belief to the learned initial strategy on every round. We decided to implement an agent that learns the opponent's strategy during a specific episode. As stated before, to do so, we use a global belief that changes according to the opponent's actions at the end of each round. In order to validate our approach, several experiments were conducted with several kinds of opposing agents. Each of the following subsections focuses on one of each of those face-offs with other agents.

All graphics show the reward per episode and 500 actions are considered.

The Random Agent and AStar Agent used in these experiments were implemented by the challenge promoters. While the Defective Agent and Tit-For-Tat Agent were implemented by us in order to have more evaluation cases.

4.1 Vs. Random Agent

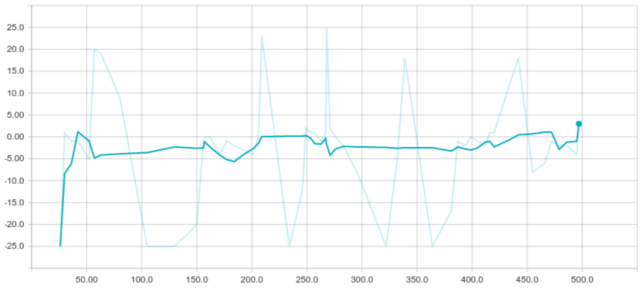

Figure 2: Reward per episode against a Random Agent

As we expected our agent doesn't work very well against a random agent because the opponent actions are totally random, so the learning is totally ineffective as the opponent doesn't really have desires, and therefore the belief doesn't match what the opponent is thinking.

4.2 Vs. AStar Agent

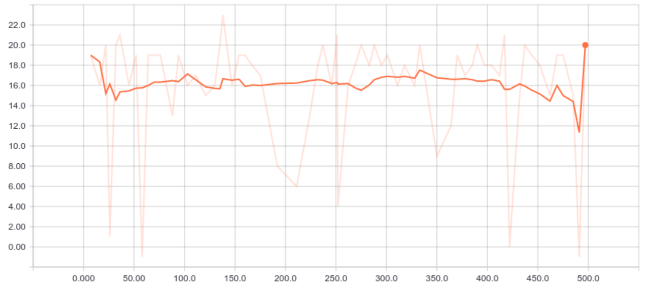

Figure 3: Reward per episode against a AStar Agent

Our agent behavior performs well against the AStar Agent, because it is always cooperative and so the reward is maximized. Whenever the pig is impossible to catch our agent goes to the closest door if it is closer than the opponent but the opponent doesn't lose trust because, as said before, it is always cooperative.

4.3 Vs. Defective Agent

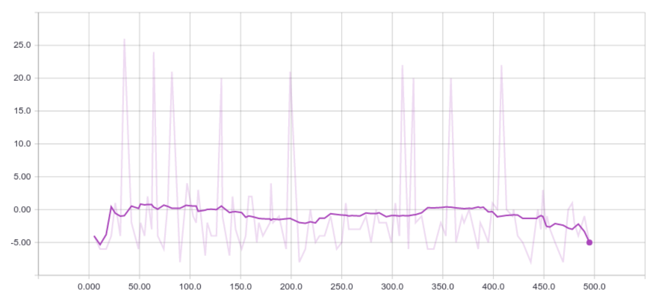

Figure 4: Reward per episode against a Defective Agent

When our agent plays against an always defective agent the reward is minimized because it never trusts in the opponent and always tries to escape. One can note that the overall performance is slightly better than with the random agent because our agent can deal better with this agent and run more often instead of wasting time.

4.4 Vs. Focused TIT-FOR-TAT Agent

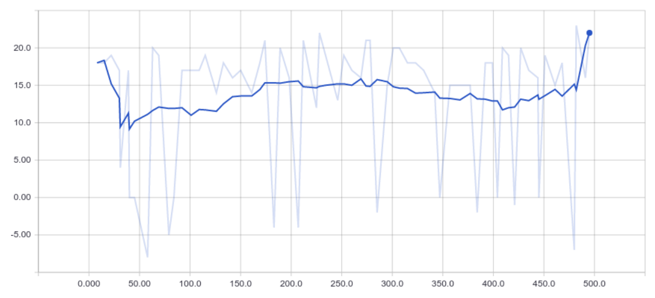

Figure 5: Reward per episode against a Tit-For-Tat Agent

In this case the reward per episode is generally good because our agent understands that opponent is mostly cooperative. When the agent escapes because the pig is impossible to catch, the opponent loses trust in our agent and for that reason, the results are worse than they were against the AStar based opponent.

4.5 Vs. Itself

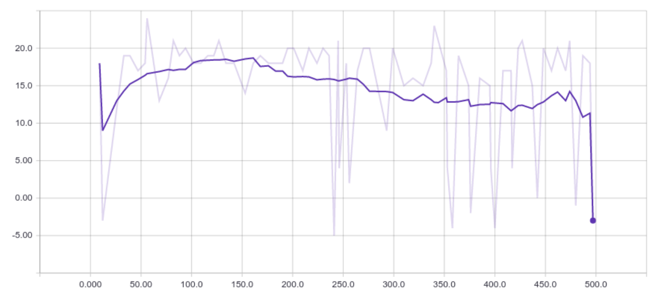

Figure 6: Reward per episode against itself

In this case the agents are mostly cooperatives and the reward is maximized. An interesting behavior appears whenever one of the two agents escape. Our agent only escapes when he distrusts the other agent or the pig is impossible to catch, and he forgives the other's defection if the pig is not catchable. When playing against itself, it is always the case that running implies that the pig is not possible to corner, and so an environment of distrust never happens. In practice, we can see this happens because the belief doesn't change.

4.6 Belief analysis

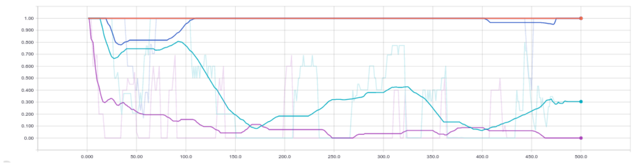

Figure 7: Beliefs of the agent over time

In this figure we can see, for all agents mentioned before, a representation of the beliefs over time, where 1 indicates maximum cooperation belief and 0 indicates the minimum. The main interesting observation is when the agent plays against itself as our agent believes that the other agent will cooperate because we forgive the agent if he runs when pig is impossible to catch and this is the only case that he actually escapes. The same logic is applied when our agent plays versus AStar Agent because he cooperates every time.

With Tit-For-Tat our agent starts with the belief that the opponent will cooperate and we can see that our agent escapes and the opponent loses the confidence in our agent. Over about 100 episodes the agents didn't trust each other. For some reason they end up cooperating and the belief goes to 1 until the end.

When our agent plays against Defective Agent the beliefs are kept very low over time, because the opponent always tries to defect and our agent doesn't trust in the opponent.

Finally the least interesting result is when our agents plays against the random agent because the beliefs don't show any kind of pattern and it's very hard to extract conclusions.

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.