## I. Learning

Learning and planning in this setting is not trivial and can only be achieved through several techniques that allow us to represent the huge state and action space of the *Domination Game* in a minimalistic way. First of all, we are dealing with a continuous state space in respect to the agents' positions, the observations about our opponent agents, the amount of ammo that we have - can be an integer from zero to infinite theoretically, and the general statistics about the game which could be represented by a huge number of discrete integer values. As an example, we can represent the score of the game as a really informative feature of our state space. However, to represent the score you need all 2-decimal point float numbers between 0.00 and 1.00, which are 101 discrete states. In this case, techniques like keeping only the first decimal point of the score value leads to a huge decrease in the number of states, i.e. 11 states. Hence the state space of this game grows rapidly in respect to the number of features we use to represent the state and the action space of our agents. As it is referred in [**boutilier2011decision** ], the biggest difficulty when we are dealing with state-based problem formulations, is the *curse of dimensionality*, the state action space grows exponentially with the number of features. We can see this *Domination Game* problem as a *Markov Decision Process*, which is fully observable in respect to our agents' positions, the control points' state, and the game state. However, we have partial observability for the opponent agents, which of course can be represented as full observable case only when opponents are in our field of view.

### A. Grid World

In this subsection, we formalize the problem, and we define the state and action state space, that we used in the first and most simple approach.

*1) State-Action Space:* Our first approach to deal with the problem of the continuous state space was to split the map into tiles of size $16 \times 16$, which was also the original tile size of the game itself. A method which is similar with *tile coding*, as it is described in Chapter 7 of *Hado Van Hasselt* book, about *Continuous State and Action Spaces*. So, the first feature of our state space is the position of our agent in this grid world, meaning the closest from our agent tile coordinate in the 2D space, given by the *Euclidean distance*. For example, the states of the agent position can be given by,

$$S_{position} = \{(0,0), \ldots, (World_{width}/16, World_{height}/16)\} \tag{1}$$

The second feature of the state space is the *Control Points State*, each control point can be the following states,

$$S_{cps} = \{-1, 0, 1\} \tag{2}$$

$-1$, when is dominated by the opponent team, 0, when is neutral, and 1 when is occupied y our team. Given that we have two control points in our map the state that is able to describe the *Cps* state space is the following,

$$S_{full\_cps} = S_{cps} \times S_{cps} \tag{3}$$

, which is the cross product between the two control points' states.

For the action space, each agent had all neighboring tiles to drive there or to stay in the same position, resulting to 9 actions for each given tile. However, actions which would lead the agent hit a wall in the map excluded from selection. So the final state space will be:

$$S = S_{position} \times S_{full\_cps} \times A \tag{4}$$

*2) Reward Function:* The reward function in this approach was obtained by a single function, which was checking the difference between the previous and the current condition of the *Cps* state.

$$Reward = (S^{t}_{full\_cps(i)} - S^{t-1}_{full\_cps(i)}) \times 10, \forall i \in \{0, 1\} \tag{5}$$

$$Reward = Reward + (Ammo^{t} - Ammo^{t-1}) \times 4 \tag{6}$$

There was also a reward when the agent succeed to obtain ammo, but weighted less that the reward for capturing a control point.

*3) Learning Method:* We decided, to implement *independent Q-Learning* for each one of our agents. Each agent holds its own *Q-table*. The states that an agent can be are the possible position in the grid world, together with the state of the control points. Furthermore, in each of these states, it has a set of possible action that can lead the agent to drive in a neighboring tile. The update rule of the *Q-Learning*, was updating the table of each independent agent every time the agent reached its target location, i.e. its previous action. We used *Q-Learning* with 0.7 learning rate-$\alpha$ and the same value for discount factor-$\gamma$.

*4) Results:* The results of this approach was tested in two settings. The first setting was on the same map but on a *1vs1* game. The second approach, was the original game *3vs3*. The results of *independent Q-Learning* in this grid world representation of the problem were really bad and we think that are not even worth to be mentioned here. One explanation is that we have no information in the state space referring to the agent's orientation. So, each agent can easily choose target locations that are located in its backward direction and this caused the agents to perform really slow turns and causing a delay to the completion of their actions, as well as the update rule for several game steps.

### B. Interesting Points as States

In the previous subsection, we described an easy way to formalize the problem of learning in this Markov decision process setting. Our approach for selecting an ideal state and action space was not really applicable to this game. In this subsection, we are presenting a better way to represent the state and the action space.

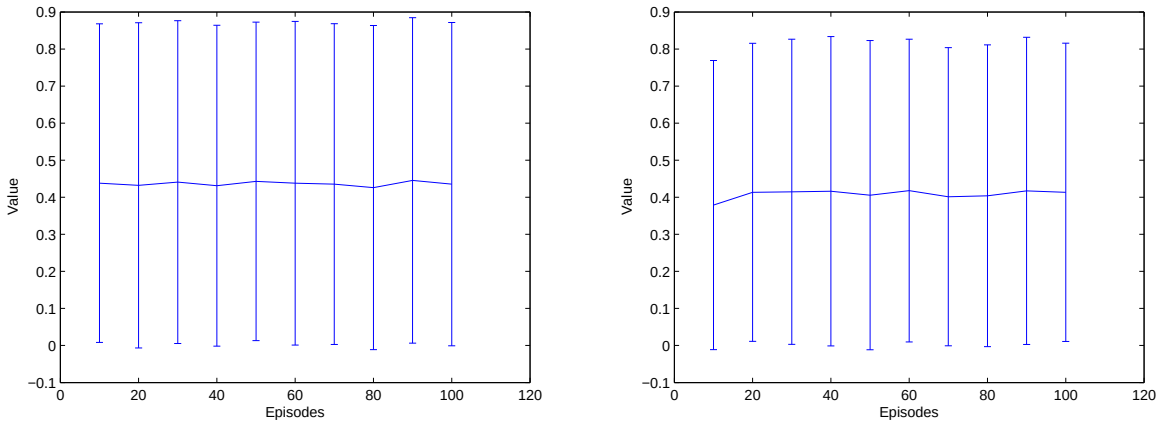


Fig. 1. Experimental results. Left, $1vs1$ agents setting. Right, $3vs3$ agents setting.

*1) State-Action Space:* In *Domination Game*, there are four important points in the map, the two control points and the two ammo spots. Furthermore, the initial positions of the agents can be described as an important point as well. Here, we are investigating the use of these important spots into the map as states in the MPD representation of our learning problem. Hence the feature describing the position is an important point of the map. This can be described as:

$$S_{position} = \{CP_1, CP_2, AM_1, AM_2, INIT\} \quad (7)$$

But, this is a really tiny representation in contrast to the previous one, which had a grid over all map positions. So, we decided to represent a joint state space which includes information for all the three agents. Agents now, will decide their action not only depending on their own locations, but also on where their teammates are located.

$$S'_{position} = S_{position} \times S_{position} \times S_{position} \quad (8)$$

The first position in this product is always referring to the agent who updates its *Q-table*, and the other two positions are referring to the other two agents in lexicographic order.

The current position state of each agent is given always by its previous visited state in this state space and the action is represented by the driving goal of the agent. We also used the state of the control points as they were described in equation (3). Third component of our state space is the opponents' positions. However, our agents are not always aware of the opponents' positions in the map, due to the limited range they can observe. As a result, we decided to represent our knowledge about the opponents as a vector of four elements, as many as the interesting points in the map except from our initial position. Each element in this vector can take integer values from 0 to 3, representing how many opponent agents are located near to each state.

$$S_{foes} = \{< 0,0,0,0 >, < 0,0,0,1 >, \ldots, < 0,0,0,3 >\} \quad (9)$$

The knowledge of the opponents' positions is known to all of our agents, even only one agent from our team can see opponents.

For the action space, each agent had all possible position states except from the initial position. The action set was:

$$A = \{CP_1, CP_2, AM_1, AM_2\} \quad (10)$$

The state-action space is now:

$$S = S'_{position} \times S_{full\_cps} \times S_{foes} \times A \quad (11)$$

Comparing with the previous approach described in the above subsection, we can realize that this one is more compact and it allow to us use more information.

*2) Reward Function:* The reward function is the same as described in the grid world approach.

*3) Learning Method:* One more time, we decided, to implement *independent Q-Learning* for each one of our agents. Now, each agent does not hold its own *Q-table*, but all agents update the same table. We can see it as a group of agents try to learn a symmetric optimal policy. As an example, imagine there are two agents in states A,B respectively. The optimal action for agent one, in state A and agent two in state B, is the same as the optimal action for the second agent in state A and agent one in state B. Again, the update rule of the *Q-Learning*, updates the table of each independent agent every time the agent reaches its target state, i.e. its previous action. We used *Q-Learning* with 0.7 learning rate-$\alpha$ and the same value for discount factor-$\gamma$.

*4) Results:* Finally, the results of this approach are illustrated in the Figure 1. We tested our approach against the random-default agent. The reason why we did that, was that the handcrafted agents from other teams were too good to be explored by the learning agent. The value represented by the y-axis in these figures is:

$$step = (max\_steps - end\_steps)/max\_steps \quad (12)$$

$$score = (score - max\_score/2)/max\_score \quad (13)$$

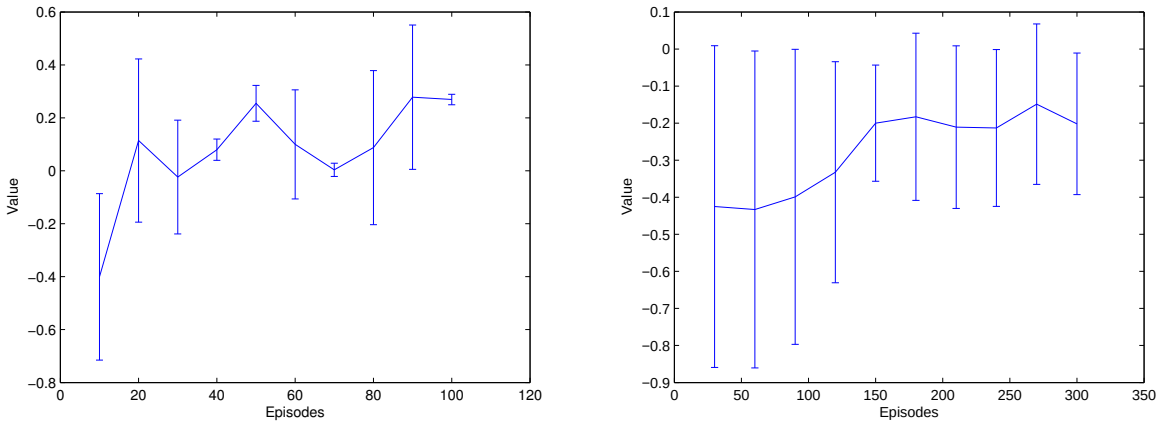


Fig. 2. Experimental results. Left, $1vs1$ agents setting. Right, $3vs3$ agents setting.

$$value = step \times score \tag{14}$$

From the last three equations, we can refer that a win counts more if it is early in respect to the total steps that the game can continue, and a lost in the lately steps is better than a lost in the first steps. We tune the settings of the game, and these episodes last for 3000 steps, and the maximum score is 1000, in order to have a faster learning process. In every number of episodes in the x-axis the average of the previous 10 episodes is computed. The whole simulation repeated three times, the mean and the standard deviation are presented. The left graph in figure 1, is about an one vs. one game. We can see that our learning agent obtains a high reward even from the first ten completed episodes. This is normal, because the agents chooses action among only interesting points, causing to a behavior similar to the random-default agent. In addition, it learns to choose actions that are profitable for it. The same applies for the right graph, where we have a complete team of three agents against the default team. We can see only a small reduce to the value that the agents obtained over the first 10 episodes.

### C. Interesting Points and Transitions as States

We saw, how the performance of our agent improved using interesting points as the state space of our MDP framework. Here, we are investigating one more improvement that can lead us to a better learning scheme.In general, considering only interesting points as states, causes really delays at the update rule of *Q-learning* for example. Furthermore, there might be also a lot of changes in the world state while an agent is in a transition from its previous state to another. Imagine for example, that agent A chooses an action towards the control point 1. Simultaneously, agent B chooses the same action, agent A reaches control point 1 first, but agent B cannot change its action and is forced to go to the control point no matter if it is taken by its teammate. Thus, there is a need of additional states which are going to describe transitions from one state to another. This idea is exploited through the next parts of this paper, as our last and best hope to acquire a better performance for the agent.

*1) State-Action Space:* As in previous part in this section, we are going to describe each part of the MDP representation. First of all, the feature of the state space which is going to represent the position of each agent is now different. Given a set of states as in eq. (7), and a set of transitions T, the new state space will be:

$$S_{position} = \{CP_1, CP_2, AM_1, AM_2, INIT\} \tag{15}$$

$$T_{position} = permutations(S_{position}, S_{position}) \tag{16}$$

$$S'_{position} = chain(S_{position}, T_{position}) \tag{17}$$

So, the agent now, can be in state:

- I am in state $CP_1$ or,
- I am going from $CP_1$ to $CP_2$.

As you can realize, this is a simple way to add more information about states. Unfortunately, this is the state space now is impossible to contain the same amount of information about the teammates of our agent. So, we decided, in order to keep it feasible computationally, to represent the state of our teammates with their goal, again, in lexicographical order.

$$S''_{position} = S'_{position} \times S_{position} \times S_{position} \tag{18}$$

In this test case, we are not using the information for the opponent agents' positions, in order to keep it computationally tractable, but we use the Cps states from eq. (3).

For the action space, each agent has all possible position states except from the initial position. The action set is the same as before.

$$A = \{CP_1, CP_2, AM_1, AM_2\} \tag{19}$$

The state-action space is now:

$$S = S''_{position} \times S_{full\_cps} \times A \tag{20}$$

*2) Reward Function:* The reward function is the same as described in the grid world approach. This function is described in eq. (5-6).

*3) Learning Method:* The reason why we chose to implement *Independent Q-Learning* not only in this test case, but in the previous test cases as well, was the fact that the state-action space then was really intractable. Almost $500Mb$ was the value table when we tried to include joint actions in our state and action space, which is indicative about how much the state space grows in a multi-agent setting. Again, the update rule of the *Q-Learning*, updates the table of each independent agent every time the agent reaches its target state, i.e. its previous action. We used *Q-Learning* with 0.7 learning rate-$\alpha$ and the same value for discount factor-$\gamma$. Each update rule is performed after each game-step and not like the previous case, when the update rule was only performed after each arrival in the target state.

*4) Results:* In figure 2, the result of the discussed approach are presented. The same values are presented here as in the previous figure for x and y axis. As it is explained before, now agents when they are choosing an action, towards a state, in the next steps they are going to be in a transition state. From this transition state, they can also choose an action, towards another state than their initial target. Hence at the beginning they preform really poor. Again, in the left graph we have only two agents playing against each other. After 100 games-episodes, the learning agent obtain a value of greater than 0.2, which is lower than the previous test case. Thing are worse for the three versus three game. After a huge number of episodes, 300, the only thing that the learning team of agent learned was how not to lose quick. One possible explanation is that now the state space is really bigger, and the agents have not complete information about the opponents, and their teammates' position as in the previous test case. One possible improvement would be to include joint actions in our state-action space. Then, the policy would have been better, and the learning process would not have lead the agents to learn a symmetric optimal policy to act. However, these are the difficulties when you are dealing with huge state spaces, this framework really gave us the change to understand why learning and planning in huge state spaces is challenging, especially with the presence of multiple agents.

*D.*