# Autonomous Agents
# Assignment 1 Report

Nicolò Girardi, Jorge Sáez Gómez, Johan Sundin

September 20, 2013

## 1 Introduction

In this assignment we study a predator vs. prey grid world Markov Decision Process, in which the predator's task is to catch the prey as quickly as possible. A schematic view of a MDP is depicted in figure 1. The agent (in this case the predator) is given a state as input from the environment, transfers an action back to the environment and receives a reward and a new state as a result of that action. The reward depends on the state of the environment after the agent has chosen a move.

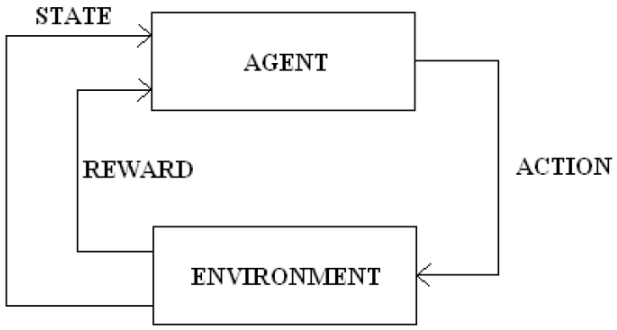


Figure 1: Transfer diagram of the MDP

We have implemented and tested the Iterative Policy Evaluation, Value Iteration and Policy Iteration algorithms, and also created a more efficient representation of the state-space. The results of our work are reported in the rest of this document. Our implementation of the algorithms makes use of the JAVA programming language, as it was one of the recommended languages for this task and the members of this group were familiar with it. Figure 2 shows the overall structure of the code of our project.

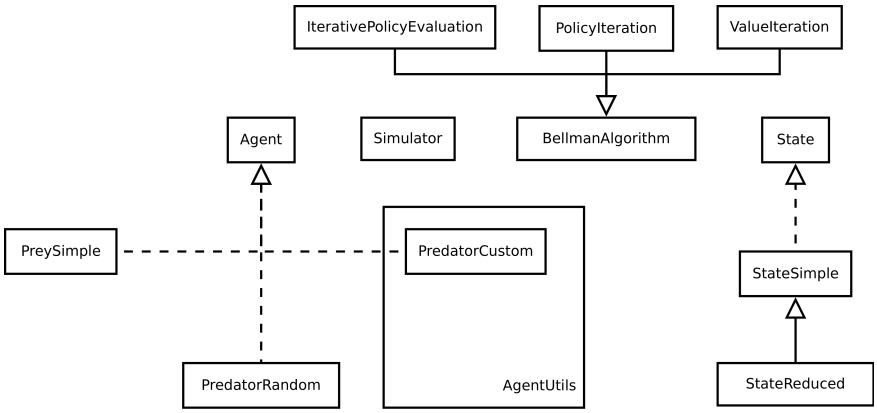


Figure 2: UML class diagram of our project. A more detailed description of the shown components lies within the included *Javadoc*.

## 2 Single Agent Planning Description

The objective of the assignment is to model a predator vs. prey grid world MDP containing one Agent and one Prey. The modelled environment is an 11 by 11 donut-shaped world space, meaning that its left and right edges are connected, as well as its top and bottom edges, also refereed to as toroidal grid. The world is visualized in figure 3.

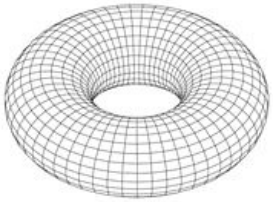

Figure 3: Simplified Visualization of the grid world model

The Prey and Agent both have 5 possible moves: Wait, where the position doesn't change, and North, South, East and West, in which the Predator or Prey moves one square in the specified direction (i.e. Up, Down, Right or Left). The Predator and Prey are allowed to move freely, with the constraint that they cannot occupy the same position at the same time. The Prey moves randomly through the environment. At every time step it chooses to wait with a probability of 0.8, or to move with a probability of 0.2. The prey is not allowed to move into a Predator. This implies that the probabilities of moves can change in different states. For example: in a state where the Predator and Prey do not occupy adjacent positions, the Prey has a probability of 0.8 of staying in the same position, and a probability of 0.05 of moving in any of the four directions. If a predator is North of the prey, the prey has a probability of 0.8 of staying

in the same position, and a probability of $0.2/3 = 1/15$ of moving either South, West or East.

The predator receives a reward of 10 for a move that results in catching the prey, and a reward of 0 for any other move. The objective of the predator is thus to catch the prey quickly and efficiently, this is done by catching it in as few moves as possible. Since the MDP is fully known to the predator it can catch the pray by first doing planning. In this assignment, several planning algorithms using dynamic programming are implemented, their effects on the behavior of the predator are reported.

# 3 Environment simulator

These are the results we obtained, averaged over one million games, when using a random policy:

- Time the predator needs to catch the prey: 130 turns.
- Standard deviation: 218

It must be noted that, in order to increase the accuracy of our results, we chose a larger number of runs than the one suggested in the assignment.

# 4 Iterative Policy Evaluation

The IPE algorithm converges in 54 iterations with $\gamma = 0.8$ and $\theta = 10^{-9}$ when evaluating the random policy for the predator. It follows from 4.5 in [1] that we want to update the value function $V^{\pi}(s)$, which tells us how good a policy $\pi$ is, but it does not specify when the algorithm should stop. That is why $\theta$ is used as a stopping condition. It also follows that the smaller the value of $\theta$, the larger the number of iterations it takes before the algorithm stops, and we could indeed appreciate this behaviour across all the algorithms we implemented.

Figure 4 shows a table of different states and their respective values, determined by our IPE algorithm. Additionally, the plots in figure 5 give a graphical view of the time to convergence depending on the chosen value for $\gamma$.

| Predator | Prey | Value |
|---|---|---|
| (0, 0) | (5, 5) | 0.005724149 |
| (2, 3) | (5, 4) | 0.18195084 |
| (2, 10) | (10, 0) | 0.18195081 |
| (10, 10) | (0, 0) | 1.1945857 |

Figure 4: Four different states and their respective values obtained by Iterative Policy Evaluation of the random policy for the predator.

[1] http://webdocs.cs.ualberta.ca/ sutton/book/ebook/node41.html

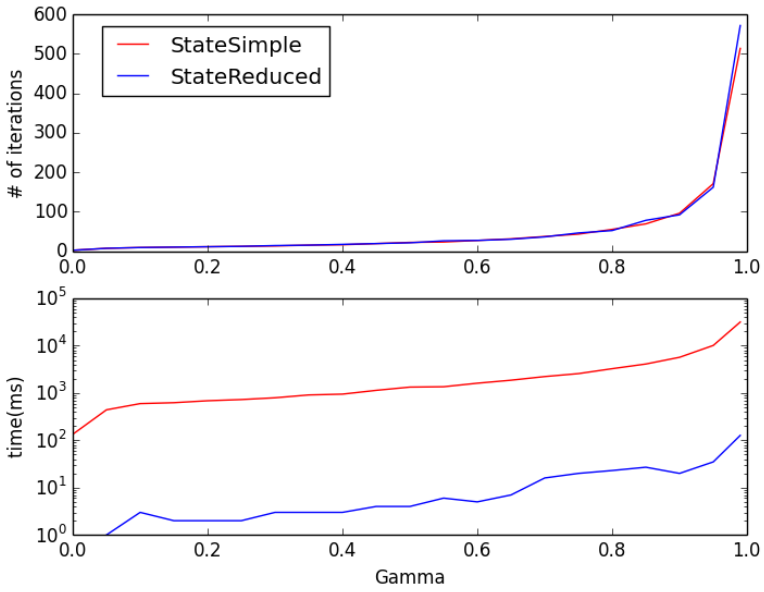


Figure 5: Performance results for the Iterative Policy Evaluation algorithm as a function of the parameter $\gamma \in [0, 0.99]$. The first graph shows the number of iterations the algorithm needs to converge, while the second one shows the average execution time in milliseconds.

# 5 Policy Iteration

Policy Iteration converges in 3 iterations when using the same parameters as for the previous algorithm. The convergence speed for different discount factors can be found in figure 6. Furthermore, the final state-values for all states in which the predator is located at (5, 5) can be found in figure 7.

# 6 Value Iteration

Value Iteration converges in 19 iterations when using the same parameters as for the previous algorithm. The convergence speed for different discount factors can be found in figure 8. Furthermore, the final state-values for all states in which the predator is located at (5, 5) can be found in figure 7.

# 7 Efficient State-Space Encoding

In order to design an efficient encoding of the state-space for the proposed problem, we first realized that the actual positions of the prey and the predator

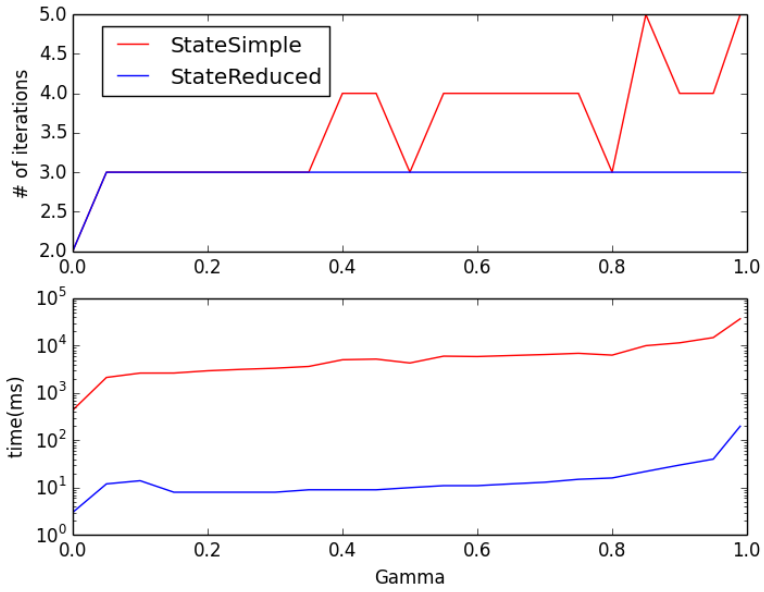


Figure 6: Performance results for the Policy Iteration algorithm as a function of the parameter $\gamma \in [0, 0.99]$. The first graph shows the number of iterations the algorithm needs to converge, while the second one shows the average execution time in milliseconds.

within the grid are not relevant for a solution to the problem: only their relative distance to each other matters. By distance we effectively mean here the *Manhattan distance*. This property arises from the toroid-shaped world: since it wraps in every direction, no pair of positions present an intrinsic advantage or disadvantage over each other, as long as their relative distance is kept constant.

For example, if the prey is located at (2, 3) and the predator is at (5, 0), the distance between the two agents is then $(5 - 2, 0 - 3) = (3, -3)$. We refer to this distance as the *inner-distance* between the agents, since we did not "wrap" along some edge of the world in order to measure it. However, it is important to note that two different distances may be measured between any two given cells of the world: we will refer to these as the *outer-distance* and the *inner-distance*, depending on whether we "wrap" along some edge to measure it or not, respectively. For instance, in our previous example the *outer-distance* becomes $(3-11, -3+11) = (-8, 8)$. However, both distances encode the same situation anyway, and we would also like to collapse equivalent distances in this sense to a single state. We can archieve this by simply using the shortest of the two distances.

Below we show an extract of the code that is actually used in our project to calculate this minimum distance between the agents. Only the calculations

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3.88 | 4.29 | 4.74 | 5.24 | 5.79 | 6.25 | 5.79 | 5.24 | 4.74 | 4.29 | 3.88 |
| 4.29 | 4.71 | 5.23 | 5.8 | 6.44 | 7.0 | 6.44 | 5.8 | 5.23 | 4.71 | 4.29 |
| 4.74 | 5.23 | 5.8 | 6.44 | 7.15 | 7.84 | 7.15 | 6.44 | 5.8 | 5.23 | 4.74 |
| 5.24 | 5.8 | 6.44 | 7.15 | 7.94 | 8.78 | 7.94 | 7.15 | 6.44 | 5.8 | 5.24 |
| 5.79 | 6.44 | 7.15 | 7.94 | 8.78 | 10.0 | 8.78 | 7.94 | 7.15 | 6.44 | 5.79 |
| 6.25 | 7.0 | 7.84 | 8.78 | 10.0 | 10.0 | 10.0 | 8.78 | 7.84 | 7.0 | 6.25 |
| 5.79 | 6.44 | 7.15 | 7.94 | 8.78 | 10.0 | 8.78 | 7.94 | 7.15 | 6.44 | 5.79 |
| 5.24 | 5.8 | 6.44 | 7.15 | 7.94 | 8.78 | 7.94 | 7.15 | 6.44 | 5.8 | 5.24 |
| 4.74 | 5.23 | 5.8 | 6.44 | 7.15 | 7.84 | 7.15 | 6.44 | 5.8 | 5.23 | 4.74 |
| 4.29 | 4.71 | 5.23 | 5.8 | 6.44 | 7.0 | 6.44 | 5.8 | 5.23 | 4.71 | 4.29 |
| 3.88 | 4.29 | 4.74 | 5.24 | 5.79 | 6.25 | 5.79 | 5.24 | 4.74 | 4.29 | 3.88 |

Figure 7: Table of state-values for an optimal policy of the predator, as calculated by the Value Iteration and Policy Iteration algorithms. The prey is assumed to be at (5, 5), while the predator is located at the equivalent cell of the world with respect to the table cell itself.

for the horizontal dimension are included, since those of the vertical one are equivalent:

```
int innerX = x[Agent.TYPE_PREY] - x[Agent.TYPE_PREDATOR];
int outterX;

if (innerX < 0)
{
	outterX = innerX + ENVIRONMENT_SIZE;
}
else
{
	outterX = innerX - ENVIRONMENT_SIZE;
}

if (Math.abs(innerX) < Math.abs(outterX))
{
	distanceX = innerX;
}
else
{
	distanceX = outterX;
}
```

This reduced state-space is implemented in our code within the class *StateReduced*. We took great care to make every other component of the code compatible with both state-spaces. Actually, both the simulator and all the algorithms we implemented can be used with either state-space by simply passing them a state object of the desired class.

By using this proposed encoding, the original state space is greatly reduced from $11^4$ different states to only $11^2$. It can be seen that this is indeed the case because the original state space directly encodes 4 coordinates that can range between 0 and 10, hence $11^4$ states in total, while there are only $11^2$ different distances between the agents, which can be generated by positioning one agent in a fixed position of the world and then change the position of the other agent.

# 8 Conclusions

In this assignment we have made use of the Bellman equation in the terms of dynamic programming. In other words, we used a recursive nested loop to update a policy by learning its value function. This could be done due to the premises that the MDP model is fully known to the agent. Furthermore, these

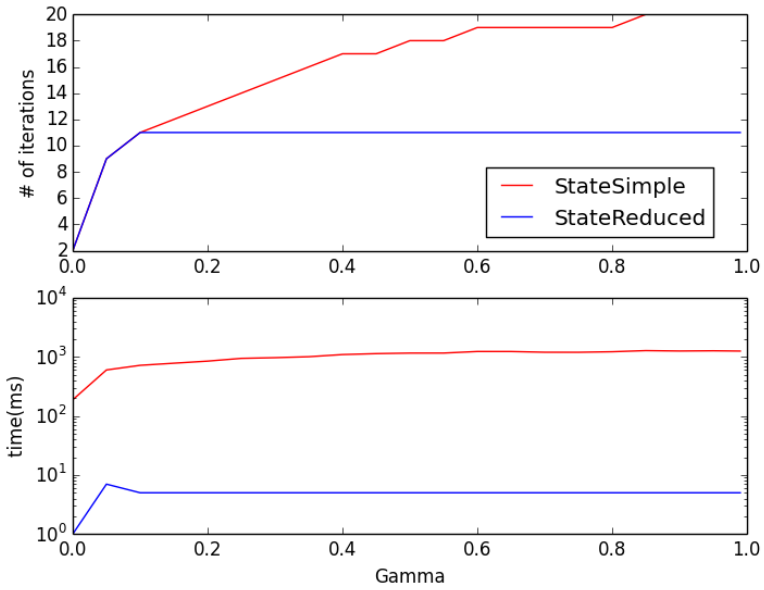


Figure 8: Performance results for the Value Iteration algorithm as a function of the parameter $\gamma \in [0, 0.99]$. The first graph shows the number of iterations the algorithm needs to converge, while the second one shows the average execution time in milliseconds.

are the points we consider more worth mentioning as the conclusions for this assignment:

- The optimal policy for this task, once calculated by our algorithms, turned out to be simply going straight away for the prey, which is indeed the expected, intuitive result.
- It can be seen from the figures that our proposal for a reduced state-space greatly and consistently reduces the execution time of all algorithms, which is an important practical factor for their use.
- It can also be checked in the plots that higher values for $\gamma$ result in higher execution times of the algorithms. This result makes sense since the higher this value is, the more the algorithm must "look ahead" into the future to calculate the state values, and thus the convergence becomes slower. This is always true except from the plot representing our test results using Value iteration with the reduced space representation, where we can see a decrease in the convergence time for a certain gamma that could be caused by the fact that the algorithm is so fast that small external factors can noticeably influence the performance.
- We could check in our internal tests that lower values for $\theta$ also result in higher execution times of the algorithms, because the stop condition becomes then more tight.
- The Value Iteration algorithm performs better for this particular problem than the Policy Iteration algorithm. However, we do not have any real base to explain why this is the case, and we believe that answering this question would need a more extensive analysis than the one presented in this document.