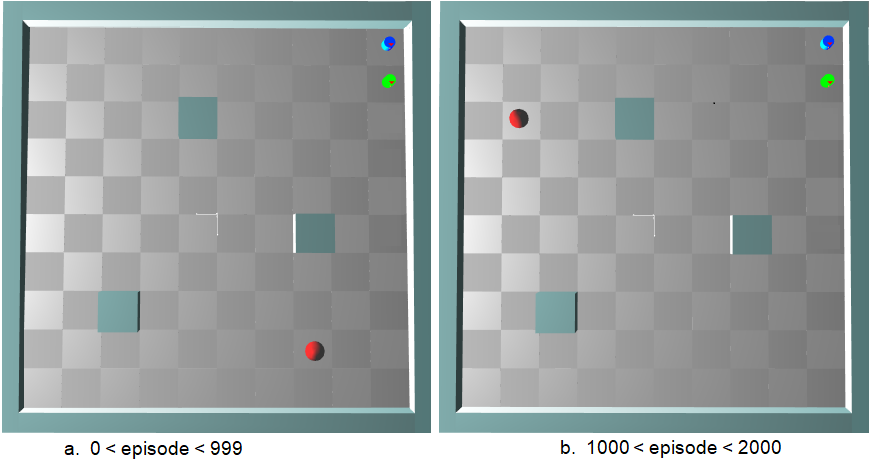
**Cooperative-RL-methods-for-Hunting-game**

**About Project**

This project is an improved version of Simbad Simulator (<http://simbad.sourceforge.net/>). It consists of a cooperative hunting task with a temporary moving target. It is a popular case study where some agents, called predators, are trying to capture one other agent, called the prey.

A new discrete environment is implemented here with several new distributed and cooperative multi-agent reinforcement learning methods. We principally propose:

* Two new action selection strategies: The TM maxNextQ Policy and The the TM maxNextQ LRVS Policy.
* A new distributed reinforcement learning approach for two agent systems.



This project is the codebase for the paper "[Cooperative](https://www.aclweb.org/anthology/D19-1399.pdf) Multi-Agent Systems Using Distributed Reinforcement Learning Techniques" in KES 2018, located at [https://www.sciencedirect.com/sc**ien**ce/article/pii/S1877050918312626](https://www.sciencedirect.com/science/article/pii/S1877050918312626). The usage code below makes sure you can reproduce almost same results as shown in the paper.

In this paper, the fully cooperative multi-agent system is studied, in which all of the agents share the same common goal. The main difficulty in such systems is the coordination problem: how to ensure that the individual decisions of the agents lead to jointly optimal decisions for the group? Firstly, a multi-agent reinforcement learning algorithm combining traditional Q-learning with observation-based teammate modeling techniques, called TM Qlearning, is presented and evaluated. Several new cooperative action selection strategies are then suggested to improve the multi-agent coordination and accelerate learning, especially in the case of unknown and temporary dynamic environments. The effectiveness of combining TM Qlearning with the new proposals is demonstrated using the hunting game.

**Setup**

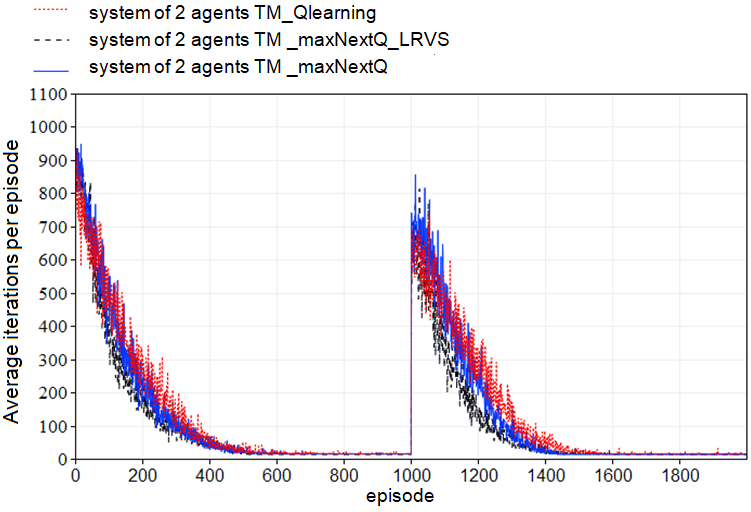
1. Install Java3d, you can get it here <https://www.oracle.com/java/technologies/javase/java-3d.html>
2. Download the simbad simulator, download links and helpful documentations can be found here <http://simbad.sourceforge.net/>
3. Download the Chartdirector Library for JAVA from <https://www.advsofteng.com/download.html>
4. Add simbad.jar and Chartdirector.jar to the current project.
5. Run the class MyProg, the main class of the project.

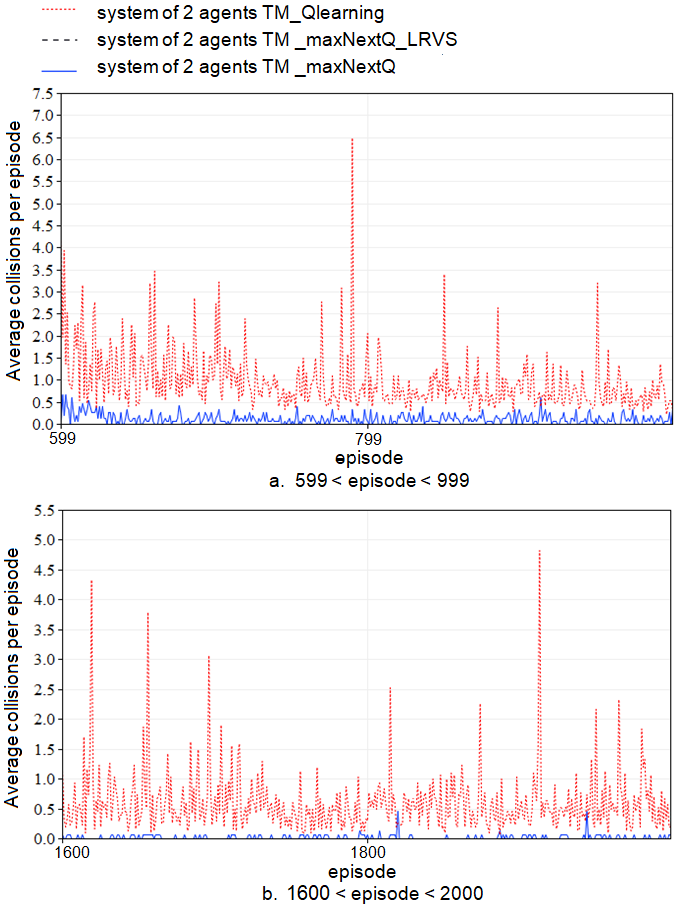
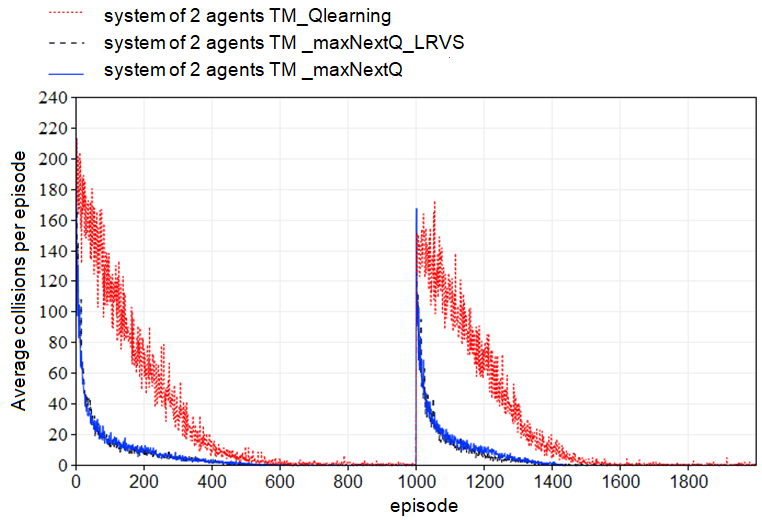
**Description of principle classes:**

A brief description of main classes is described below

* Class **MyProg**: it is the main class of the project. Here, we creates the environment by calling the class **MyEnv** and personalize its size.
* Class **MyEnv**: Here, we can personalize the number and starting positions of learning agents (class **MyRobot**), the shape and positions of obstacles (class **Box**) and the number and initial positions of the prey (class **CherryAgent**). We can also modify the number of learning experiences to be conducted.
* Class **MyDiscreteRobot:** it defines the behavior of the learning agents (the predators): the most important method here is **performBehavior():void.** As we adopt the reinforcement learning approach, the method performBehavior() describes an iteration of the learning algorithm:
  + Choose the action to perform: many action selection strategies are implemented and are located in the **LearningAgent class.**
  + displace the learning agent: the environment is discrete and only five actions are possible at each time steps: UP, DOWN, RIGHT, LEFT and NoMove.
  + acquisition of reward and next state
  + updating the value function Q and the teammate model P using methods **update\_Table\_Q():void** and **update\_Table\_P(): void** of the **LearningAgent class**, respectively.
* Class **LearningAgent:** here, we mainly define tables P and Q, their updating methods and the action selection methods.
* Class **MersenneTwister**: This class implements a powerful pseudo-random number generator developed by Makoto Matsumoto and Takuji Nishimura during 1996-1997. For more details, you can see <https://commons.apache.org/proper/commons-math/javadocs/api-3.3/org/apache/commons/math3/random/MersenneTwister.html>
* CLass **LearningGraph:** weuse the **ChartDirector** library to generates the learning graphs showing the results of conducted experimentations. These graphs principally show the average iterations or collisions per episode.

**Results**





**Citation**

@article{zemzem2018cooperative,

title={Cooperative multi-agent systems using distributed reinforcement learning techniques},

author={Zemzem, Wiem and Tagina, Moncef},

journal={Procedia Computer Science},

volume={126},

pages={517--526},

year={2018},

publisher={Elsevier}

}