SINCITY: A PEDAGOGICAL TESTBED FOR CHECKING MULTI-AGENT LEARNING TECHNIQUES

A.M. Peleteiro-Ramallo, J.C. Burguillo-Rial, P.S. Rodríguez-Hernández, E. Costa-Montenegro
Department of Telematic Engineering
University of Vigo
36310, Vigo, Spain
E-mail: jrial@det.uvigo.es

KEYWORDS
Multi-agent learning, urban traffic models, NetLogo.

ABSTRACT

In this paper we present SinCity, a pedagogical testbed to compare multi-agent learning strategies. SinCity has been developed in NetLogo and it can be considered as an extension of the simple predator-prey pursuit problem. In our case, we model a police/thief pursuit in an urban grid environment where several elements (cars, traffic lights, etc.) may interact with the agents in the simulation. SinCity allows to model, in a graphical friendly environment, different strategies for both, the Police and the thief, also implying coordination and communication among the agent set. SinCity is oriented mainly as a pedagogical testbed for learning techniques. The main contributions of the paper are a graphical simulating environment in NetLogo (distributed as open source), a pedagogical testbed for simulating multi-agent learning strategies, and the results concerning the simulations performed.

INTRODUCTION

The predator-prey pursuit problem (Benda et al. 1985) is one of the first and well-known testbed for learning strategies in multi-agent systems. Basically it consists in a set of agents, named predators, that aim to surround another agent, named prey, that must escape from them. This problem has been addressed many times by the AI Community. Initially, (Korf 1992) proposed a solution without multi-agent communication. A great number of alternatives have emerged since Korf's one, involving genetic programming (Haynes and Sen, 1995), Petri nets (Chainbi et al. 1996), reinforcement learning (Tan 1997), blackboard architectures (Jim and Giles, 2000), profit sharing (Katayama et al. 2005), and many more.

Many other learning simulations have appeared in multiple domains (social sciences, economy, biology, e-commerce, etc.) and with much more complex scenarios. Presently one of the most popular ones are the Robocop Simulation Leagues for Soccer and Rescue (Robocop 2006). The simulators and techniques used for these two competitions are brand new approaches but usually very complex to be managed as a pedagogical testbed for comparison of new learning techniques.

In this paper we propose a simulator named SinCity, modeling a city, and a new pursuit problem, which is a more complex version of the predator-prey. SinCity is developed in an open source, simple, popular and user friendly environment (Netlogo) that may be used as a testbed for checking multiple multi-agent learning techniques.

The rest of the paper is organized as follows. First we introduce multi-agent systems and the Netlogo environment for multi-agent simulation. Then we describe the simulation scenario developed. Next, we introduce our police-thief pursuit model. In next section we describe the learning techniques used in our simulations and the results we have obtained. Finally, we present the conclusions and future research work.

MAS AND NETLOGO

In this section we introduce Multi-agent Systems (MAS) and Netlogo, which are the basic elements used to build the SinCity simulator.

Multi-agent Systems

Before introducing Multi-agent Systems (MAS), we need to define what we understand by an agent. Unfortunately, there is no general agreement in the research community about what an agent is. Therefore we cite a general description (Wooldridge and Jennings, 1995), and according to it, the term agent refers to a hardware or (more usually) software-based computer system characterized by the well-known properties: *autonomy, social ability, reactivity,* and *pro-activeness*. There are some other attributes that can be present, but usually they are not considered as a requisite: *mobility, veracity, benevolence, rationality and adaptability (or learning)*; see (Wooldridge 2002) for more details. A system consisting of an interacting group of agents is called a Multi-agent System (MAS), and the corresponding subfield of Artificial Intelligence (AI) that deals with principles and design of MAS is called Distributed AI.

Proceedings 23rd European Conference on Modelling and Simulation ©ECMS Javier Otamendi, Andrzej Bargiela, José Luis Montes, Luis Miguel Doncel Pedrera (Editors)
ISBN: 978-0-9553018-8-9 / ISBN: 978-0-9553018-9-6 (CD)

Netlogo

We implemented our city model using NetLogo (Wilensky 2007), a multi-agent modeling environment for simulating natural and social phenomena. It is particularly well suited for modeling complex systems that evolve. Modelers can give instructions to hundreds of agents operating independently. This makes possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from the interaction of many individuals.

NetLogo is an easy-to-use development environment. It allows to launch simulations and play with them by exploring their behavior under various conditions. Custom models can be created easily for quick tests of hypotheses about self-organized systems.

NetLogo has extensive documentation and tutorials. It also comes with a Models Library, which is a large collection of pre-written simulations that can be used and modified. These simulations address many areas in natural and social sciences, including biology and medicine, physics and chemistry, mathematics and computer science, and economics and social psychology.

NetLogo is a 2D world made of agents that simultaneously carry out their own activity. There are three types of agents:

- Patches: stationary agents that make up the background or "world". They have integer coordinates.
- Turtles: mobile agents that move around on top of the patches, not necessarily in the center, so they have decimal coordinates and orientation.
- The observer: oversees everything going on in the world. It can create new turtles and has read/write access to all the agents and variables.

An agentset can be created using a subset of the agents. There is also the possibility to create breed, a "natural" agentset of turtles, which come with automatically derived primitives.

NetLogo uses a simple scripting language to define the systems, and it also has a user-friendly graphical interface to interact with the system. The graphical interface consists of three elements:

- Controls: they allow to run and control the flow of execution. There are two types: buttons and command center. The buttons are used to initialize, start, or stop the model, and step through it. They can be either "once", which execute one action (a piece of code) or "forever", which repeat the same action (the same piece of code) until pressed again. The command center allows to ask observer, patches or turtles to execute specific commands "on the fly", while the model is running, so it is possible observe how these modifications alter the model.
- Settings: they allow to modify parameters. There are sliders (to set a quantity from a minimum to maximum by incremental steps), switches (to set a boolean variable true or false) and choosers (to select a value from a list).
- Views: they allow to display information. There are monitors (to display the current value of variables), plots (to display the history of a variable in a graphic), output text areas (text log information) and graphics window (main view of the 2D NetLogo world).

As common programming languages, NetLogo has variables, i.e., value containers. There are global variables (a single value for the variable that every agent can access it), turtle and patch variables (each turtle/patch has its own value for every turtle/patch variable), local variables (defined and accessible only inside a procedure) and built-in variables (already defined in NetLogo for turtles and patches).

To model the behavior of the agents, NetLogo has different procedures: commands (actions for the agents to carry out, i.e. "void" functions), reporters (to report a result value, i.e. functions with return type), primitives (built-in commands or reporters, i.e. language keywords), procedures (user-made custom commands or reporters) and "ask" (to specify commands to be run in parallel by a set of turtles or patches).

CITY MODEL

Previous models

The first approach to traffic models, which is included in the NetLogo distribution, is Traffic Basic (Wiering et al. 2004) (figure 1). It models the movement of cars on a highway. Each car follows a simple set of rules: it slows down (decelerates) if it sees a close car ahead, and it speeds up (accelerates) if it does not see a car ahead. It demonstrates how traffic jams can form even without any "centralized cause".

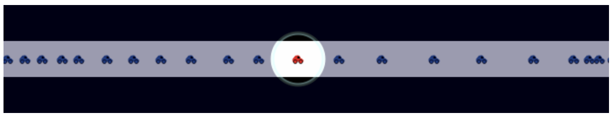


Figure 1: Traffic Basic Model

Using the movement of the cars in the previous model, a small city with traffic lights is modeled in Traffic Grid (Wilensky 2005) (figure 2, left), also included in the NetLogo distribution. It consists of an abstract traffic grid with intersections between cyclic single-lane arteries of two types: vertical or horizontal. It is possible to control traffic lights, speed limit and the number of cars, creating a real-time traffic simulation. This allows the user to explore traffic dynamics and develop strategies to improve traffic and to understand the different ways to measure the quality of the traffic.

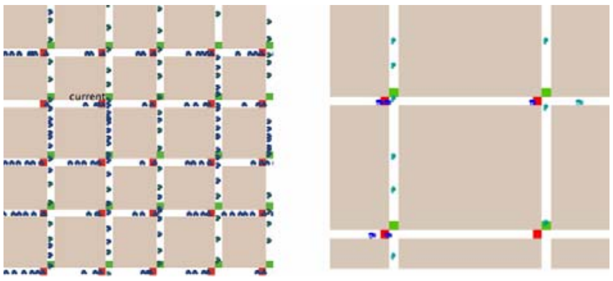


Figure 2: Traffic Grid (left) and SOTL (right) Models

Using the Traffic Grid model as a starting point, a more complex model is presented in (Gershenson 2004), called Self-Organizing Traffic Lights. Cars flow in a straight line, eastbound or southbound by default. Each crossroad has traffic lights that only allow traffic flow in one of the arteries that intersect it with a green light. Yellow or red lights stop the traffic.

The light sequence for a given artery is green-yellow-red-green. Cars simply try to drive at a maximum speed of a "patch" per time step, but they stop when a car or a red or yellow light is in front of them. Time is discrete, but space is continuous. A "patch" is a square of the environment with the size of a car. The environment is shown in figure 2 (right). The user can change different parameters, such as the number of arteries or cars.

Different statistics are shown: the number of stopped cars, their average speed, and their average waiting times. In this scenario, the author presents three self-organizing methods for traffic light control outperforming traditional ones, since the agents are "aware" of changes in their environment, and therefore they can adapt to new situations.

City Model Improvements

Our model is a more realistic city scenario, as we will explain in this section. To model the 2D scenario, different agentsets for the patches are used. The main ones are:

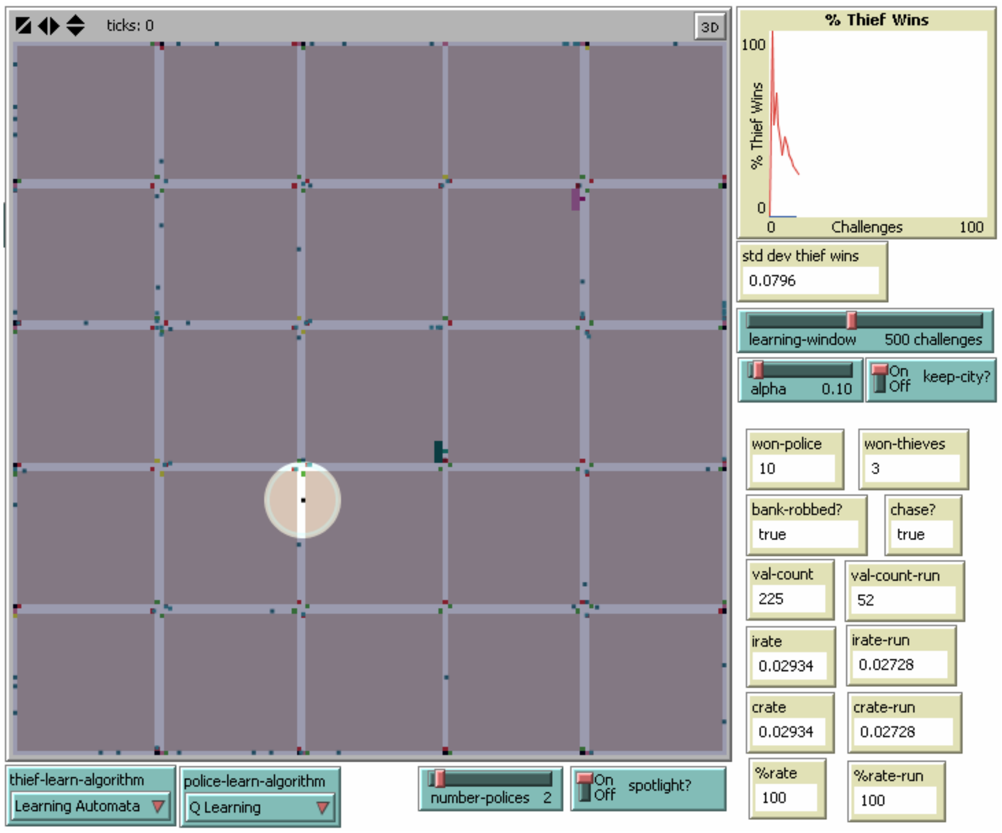
- intersections: agentset containing the patches that are intersections of two roads.
- controllers: agentset containing the intersections that control traffic lights. Only one patch per intersection.
- roads: agentset containing the patches that are roads. The rest of the patches are buildings. There will be four sub-agentsets, depending if the road is southbound, northbound, eastbound or westbound.
- exits: agentset containing the patches where cars will go when leaving the simulation.
- gates: agentset containing the patches where cars will sprout from.

Given the previous model of the traffic in a city, described in [10], we have made some improvements to represent a more realistic scenario. In the previous model, the roads have a single lane and direction, and there are only two directions by default, south and east, although they can be changed to four, by adding north and west. A car only changes road or direction depending on a probability prob-turn, which means that the cars move at random. Also, in the previous model a torus was used by default, which means that when a given car coming from the west to the east arrives to the end of the scenario at the east, the same car will appear at the beginning of the scenario at the west.
To increase realism, we remove the torus and impose four directions (north, east, south and west), and we are able to shape traffic thanks to sliders *vertical*, *southbound* and *eastbound*. Also, a by-pass road is created to improve traffic, which is the outermost in the scenario.

We have also changed the car creation and elimination scheme. Now, for every car, we define a source (a random road patch) and a destination (another random road patch), such that every car is created at a source, and it moves (following the shortest path) to its destination, where it is eliminated. The sources and destinations may be outside the world, depending on the value of the sliders *origin-out* and *destination-out*, leading to some cars appearing and disappearing at the borders of the world.

We have added the possibility of bidirectional roads, which are controlled with slider *bidirectional*, and roads with two lanes in the same direction, controlled with slider *2-lane*.

We have modified all the control methods so that, instead of just one yellow light cycle, now there are as many yellow light cycles as patches in the intersection, i.e. if the traffic light protects a bidirectional road, with two lanes in each direction, there will be four yellow lights cycles. In order to try to correct deadlocks at the intersections, a deadlock algorithm has been implemented. If a given car at an intersection after a given time has not moved, it tries to change direction in order to keep moving and to try to exit the deadlock. This movement could affect other cars and help finishing the current deadlock.



std dev thief wins 0.0796
learning-window 500 challenges
alpha 0.10
keep-city? On Off
won-police 10
won-thieves 3
bank-robbed? true
chase? true
val-count 225
val-count-run 52
irate 0.02934
irate-run 0.02728
crate 0.02934
crate-run 0.02728
%rate 100
%rate-run 100
thief-learn-algorithm Learning Automata
police-learn-algorithm Q Learning
number-polices 2
spotlight? On Off

Figure 3: The SinCity Model

Due to all these improvements, specially the possibility of an origin and a destination and bidirectional roads, a complex algorithm to guide the cars is needed. Whenever a car is in a patch that is an intersection (it belongs simultaneously to a horizontal and a vertical road), it runs the guiding algorithm in order to know if a change of direction is necessary, before moving on. If not, the car will keep the same direction, at least until the next intersection.

As seen in figure 3, with these changes we obtain a more realistic scenario where we can notice the different widths of the streets, depending if they are bidirectional and single or dual lane streets. We can also see the distribution of the traffic lights, and the by-pass road surrounding the city.

THE SINCITY PURSUIT MODEL

Our SinCity simulation is an extension of the predator-prey pursuit problem, where the prey is substituted by a thief car and the predators by a set of police cars.

On every challenge, the thief car starts driving at normal speed to a city bank. It stops in front of the bank, does the theft and getaway to its hideout at double speed. On the other hand, police cars patrol the city before the theft is done. When the thief car robs the bank an alarm is triggered, and police cars double their speed and patrol along the city trying to identify the thief's car.

The chase begins when any police sees the thief, before it arrives to its hideout, in the same road and at a distance of two blocks or less. On the one hand, if the thief's car is seen, then all police cars know its position. On the other hand, if it is lost, we keep the point where the thief was last seen. If any police car arrives to that point but the thief car is not in sight, then the chase stops, prevent the others to go to that place and the patrol continues.

We consider that the thief is captured if it is surrounded by police in a road (two police cars) or in an intersection (4 police cars). We consider that the thief escapes when it reaches its hideout and enters into it without to be seen by any police car. Besides, there are more cars in the city that cause thief or police cars to reduce their speed in the prosecution, from double to a normal one, if they are in front of the car.

Figure 3 shows a snapshot of the SinCity map and data with the thief car is highlighted. The bank is marked in red at the upper right and the hideout is marked in green at the center. At every challenge the position of the bank and hideout changes but they must have a distance greater than the 25% of the map size. At the bottom and the right of the map we can see a graphic display, some outputs related with the simulation, and also configure several parameters; related with the algorithms explained in next sections.

ALGORITHMS AND LEARNING TECHNIQUES

In this section we describe the learning techniques implemented in the SinCity simulator to compare their results. We have implemented the non-learning Korf's Algorithm, a self-organizing neural network (SOM) and two reinforcement learning algorithms: Learning Automata and Q-Learning. All these techniques are described next.

Korf's Algorithm

(Korf 1992) approach was to use a fitness function that combined two forces: each predator was "attracted" by the prey and "repelled" from the closest other predator. This solution kept predators away from other predators while they got closer to the prey; the idea was to chase the prey arranging predators in an stretching circle. Korf concluded that the pursuit domain was easily solved with local greedy heuristics.

Self-Organizing Maps Algorithm

A Self-Organizing Map (SOM) (Kohonen 2001) is a type of artificial neural network that is trained using unsupervised learning to produce a low-dimensional (typically two dimensional), discretized representation of the input space of the training samples, called a map. We use the formula:

$$Wn(t + 1) = Wn(t) + \Theta\ (t)\ .\ \alpha(t)\ .\ (I(t) - Wn(t))$$

where $\alpha(t)$ is a monotonically decreasing learning coefficient and $I(t)$ is the input vector. The neighborhood function $\Theta(t)$ depends on the lattice distance between the winner neuron and neuron v.

Learning Automata Algorithm

The Learning Automata (Narendra and Thathachar, 1989) is a type of reinforcement learning algorithm that uses two very simple rules:

$$P\ (s,a) = P(s,a) + \alpha\ .\ (1 - P(s,a)) \qquad (1)$$
$$P\ (s,b) = (1 - \alpha)\ .\ P\ (s,b) \qquad (2)$$

In these equations (α) is a small learning factor. If case of success, rule (1) is used to increase the probability of action (a) in state (s), while rule (2) is used to decrease the probability of the rest of actions.

Q-Learning Algorithm

The Q-Learning algorithm is another type of reinforcement learning algorithm (Kaelbling et al. 1996), which works with the well-known formula:

$$Q(s,a) = Q(s,a) + \alpha\ (R(s) + \gamma \max_b Q(s',b) - Q(s,a))$$

Where (s) is the previous state, (s') the next state, (a) the action previous chosen, (α) the learning factor, (R(s)) the reinforcement, (γ) the discount factor, Q(s,a) is the value of Q for the present state and action chosen, and Q(s',b) for the next state (s') and action b.

SIMULATION RESULTS

In this section we present our simulation conditions and results. The simulations have been performed on a superserver with 8 processors Intel(R) Xeon(R) CPU X5460 at 3.16GHz and with 6 MB of cache memory per processor taking less than an hour for the whole set. We use a 65% of bidireccional roads and we generate a new city for every challenge.

For implementing the learning techniques described in the previous section we took several decisions. First, police cars and the thief car take decisions about what road to take only at every intersection. This reduces the number of states and speed up the simulations.

The thief has two different learning systems. The first is used to go from its particular location to the hideout when there is no police car at sight. The other learning system is used in the chase to escape from the police. The police car only have one learning system which is used to go from its present location to a destination, for instance, to pursue the thief during the chase.

For the LA and QL techniques, when going to a destination, the input for every learning system considers the possible configurations of the road intersection (16 – 1 = 15 posibilities, as one has all roads blocked) depending if a certain road is blocked or not. We also consider the 8 possible locations of the destination point around the car. Therefore we have 8 x 15 = 120 input states for LA and QL.

In the thief case, using LA and during the chase, we consider only what is the closest police car. Therefore we have 4 inputs x 15 possible road intersections = 60 input states. For the QL case, we consider the discrete distance in blocks (0, 1 or 2) of every police car when they are closer than two blocks. Then we have 81 inputs x 15 road intersections = 1215 states. Finally, the SOM neural network is only used by the thief during the chase. First we identify the type of intersection we have (15 possibilities) and for each one we set a SOM with 4 real valued inputs (the 4 directions) with a number describing the exact distance to a police car (if anyone if less than 2 blocks, or zero if there is no one). We used a lattice of 16 x 16 = 256 neurons. Therefore, we have 256 x 15 = 3840 neurons with 4 inputs each one. The output of every neuron is one of the four possible roads to take and it is based in a probability distribution of those possible exits, and trained as the LA case.

The learning parameters we used in the simulations are: α=0.1 for the LA, QL and the probability distribution of every neuron in the SOM. Besides, in QL we set R(s)=±0.25 and γ=0.2. Finally, for the SOM case we have $□(t) = 1\ /\ (1 + 0.01*t)$ and $\Theta\ (t) = □(t) = 1\ /\ (1 + 0.01*t)$, where t is the learning iteration. Of course, as there are several SOMs, each one has its own $□(t)$ and $□(t)$ parameters.

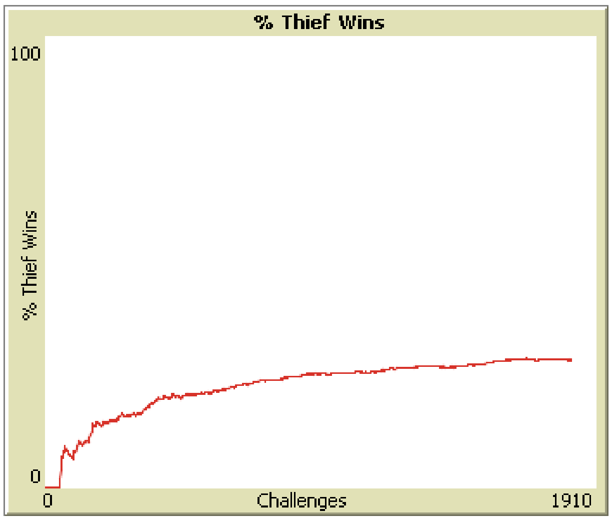


Figure 4: % of thief victories using LA (2 police cars)

Figure 4 shows the evolution of the absolute percentage of thief victories along 1800 challenges in a city of 5x5 blocks; with the thief using the LA technique. We see how it starts low and it grows as soon as the automata learns the best options in every situation. After 1000 challenges, this percentage remains stable around 28%.

Now we compare the results using the learning techniques in a run. A run (set of challenges) stops when the average standard deviation of the thief wins in last 500 challenges is lower than a 3%, provided than at least 1000 challenges have taken place.

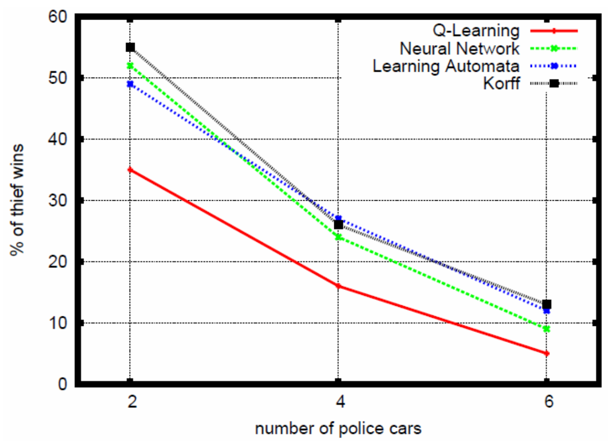


Figure 5: 10x10 map results with Police using Korf

Figure 5 shows the results obtained with a city of 10x10 blocks. We set the Korf's algorithm for the police cars and then we change the algorithms for the thief car. Korf's algorithm does not learn and may be used to compare thief success when using the learning techniques. As we may see in figure 5, the best results are obtained in average by the LA algorithm, followed by the Korf, SOM and the QL algorithms. This is even more interesting considering that the LA algorithm only uses less states and considers if there are police cars in a direction, but without determining their precise distance. We are still considering this curious result as we see how sometimes an excess of information can be a disadvange, and simple solutions are good enough for complex problems; (Korf, 1992; Gershenson, 2004) exemplify it. As it was expected (figure 5) the percentage of thief victories decreases when the number of police cars increases.

We present in table 1 a comparison among the 3 learning strategies. LA is clearly the winner in all cases, and on the other side, QL obtains the worst results. We point out that SOM is only used by the thief.

Table 1: % of thief victories among the learning strategies: LA, QL and SOM (used only by the thief)

Thief Algorithm	Police Algorithm	Police Cars	% Thief Wins
LA	QL	2	81,4
SOM	QL	2	76,6
SOM	LA	2	59,4
QL	LA	2	48,2
LA	QL	4	77,2
SOM	QL	4	46,8
SOM	LA	4	34,2
QL	LA	4	20,4
LA	QL	6	60,2
SOM	QL	6	29
SOM	LA	6	18,2
QL	LA	6	10,4

CONCLUSIONS AND FUTURE WORK

In this paper we have presented SinCity, a simulating testbed that has been created to obtain a highly flexible and efficient testbed for MAS. SimCity has been developed in NetLogo and it can be considered as a more complex version of the predator-prey pursuit problem. In our case, we model a police/thief pursuit in an urban grid environment where other elements (cars, traffic lights, etc.) may interact with the agents in the simulation. SinCity allows to model, in a graphical friendly environment, different learning strategies for both, the police or the thief.

The main contributions of the paper are a graphic city simulator in NetLogo (distributed as open source), which serves as a testbed for simulating multi-agent learning strategies. We also present the results of the simulations performed.

As future work we plan to extend the model of the traffic in the city and to allow more complex interactions among the normal traffic and the police-thief cars. We also plan to implement more complex learning techniques and consider evolutionary programming techniques for generating the agents decision rules.

REFERENCES

Benda M., Jagannathan, V. and Dodhiawalla, R. 1985. On Optimal Cooperation of Knowledge Sources, *Technical Report BCS-G2010-28*, Boeing AI Center.

Chainbi, W., Hanachi, C. and Sibertin-Blanc, C. 1996. The Multi-agent Prey-Predator problem: A Petri net solution. In *Proceedings of the IMACS-IEEE-SMC conference on Computational Engineering in Systems Application (CESA'96)*, Lille, France, 692–697.

Gershenson, C. 2004. Self-organizing traffic lights. *Complex Systems* 16, No 29.

Haynes T. and Sen. S. 1995. Evolving behavioral strategies in predators and prey. In Sandip Sen, editor, *IJCAI-95 Workshop on Adaptation and Learning in Multi-agent Systems*, Montreal, Quebec, Canada, 20-25 Morgan Kaufmann, 32–37.

Jim, K. and Giles, C.L. 2000. Talking helps: Evolving communicating agents for the predator-prey pursuit problem. *Artificial Life* 6, No 3, 237–254

Kaelbling, L.P., Littman, M.L., Moore A.W. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* 4, 237–285

Katayama, K. Koshiishi, T. and Narihisa, H. 2005. Reinforcement learning agents with primary knowledge designed by analytic hierarchy process. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing* , New York, NY, USA, 14–21

Kohonen, T. 2001. Self-Organizing Maps. Third, extended edition. *Springer*.

Korf, R.E. 1992. A simple solution to pursuit games. In *Proceedings of the 11th International Workshop on Distributed Artificial Intelligence*. Glen Arbor, MI.

Narendra, K., and Thathachar, M. 1989. Learning Automata: An Introduction. *Prentice-Hall*, Englewood Cliffs, NJ.

RoboCup 2006. Robot Soccer World Cup X. Lakemeyer, G., Sklar, E., Sorrenti, D.G., Takahashi, T. (Eds.). Vol. *Springer-Verlag* 4434.

Wiering, M., Vreeken, J., van Veenen, J., and Koopman, A. 2004. Simulation and optimization of traffic in a city. *IEEE Intelligent Vehicles Symposium*, 453–458.

Wilensky, U. 2005. NetLogo Traffic Grid model. Web site: http://ccl.northwestern.edu/netlogo/models/TrafficGrid

Wilensky, U. 2007. NetLogo: Center for connected learning and computer-based modeling, Northwestern University. Evanston, IL. Web site: http://ccl.northwestern.edu/netlogo

Wooldridge, M. and Jennings, N. R. 1995. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10, No. 2, 115–152.

Wooldridge, M. 2002. An Introduction to multi-agent systems. John Wiley and Sons, Ltd., London.

AUTHOR BIOGRAPHIES

ANA M. PELETEIRO-RAMALLO is presently finishing her M.Sc. Thesis in Telecommunication Engineering at the University of Vigo, Spain. She currently has a grant to serve as assistant researcher in the Department of Telematic Engineering in the University of Vigo, Spain. Her research interests include autonomous agents and multi-agent systems. Her e-mail is: `Ana.Peleteiro@det.uvigo.es`

JUAN C. BURGUILLO-RIAL received the M.Sc. degree in Telecommunication Engineering in 1995, and the Ph.D. degree in Telematics (cum laude) in 2001; both at the University of Vigo, Spain. He is currently an associate professor at the Department of Telematic Engineering at the same university. He has participated in several R&D projects in the areas of Telecommunications and Software Engineering, and has published more than one hundred papers in journals and conference proceedings. His research interests include autonomous agents and multi-agent systems, distributed optimization and telematic services. His e-mail address is: `J.C.Burguillo@det.uvigo.es` and his Web-page is at: `http://www.det.uvigo.es/~jrial`

PEDRO RODRÍGUEZ-HERNÁNDEZ received the M.Sc. degree in 1989 from Polytechnic University of Madrid, Spain, received the PhD degree (cum laude) from University of Vigo, Spain, in 1998. He is currently an associate professor at the Department of Telematic Engineering at University of Vigo. He has written several papers in international journals and conferences. He has participated in diverse R&D projects in the areas of Telecommunications and Software Engineering. His research interests include intelligent agents, distributed optimization, real-time and embedded systems. His e-mail address is: `pedro@det.uvigo.es` and his Web-page: `http://www-gti.det.uvigo.es/~pedro`

ENRIQUE COSTA-MONTENEGRO received the M.Sc. degree in 2000 and the Ph.D. degree (cum laude) in 2007 from the University of Vigo, Spain; both in Telecommunication Engineering. He is assistant professor at the Department of Telematic Engineering, at the University of Vigo, Spain. His research interests include wireless networks, car to car communication technologies, multi-agent systems and peer-to-peer systems. He is author of several articles in international journals and conferences and has participated in several R&D projects in these areas. His e-mail address is: `kike@det.uvigo.es` and his Web-page: `http://www-gti.det.uvigo.es/~kike`