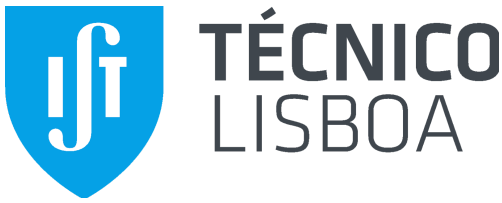


Autonomous Agents and Multiagent Systems

2nd Semester - 2015/2016

Agents In Wolf Pack

Group 01 - Alameda

Gonçalo Castilho - 75305

Tiago Gomes - 76079

Rui Silva - 77213

Abstract

In the areas of multi agent system we tried to solve several problems from the beginning without using previous knowledge. But with the complexity rising exponentially this became impossible, we could not solve real world problems, only small and simple ones.

One of this problems is the Pursuit Problem. This one was introduced by Benda et al. [1] and it has been studied due to its simplicity to understand, and difficult to solve with a high success.

In this project there were applied several solutions to test different implementations, to get better results. It started with the most simplistic architecture, **Reactive**, this one only relies on is sensors and actuators and has no memory. After that, we implemented the **Deliberative** approach, in this case the agents had memory and communicated with each other, and in each step they created a plan to achieve their goals. At last, we implemented the **Learning**, this one consisted in using several test cases until all agents learned the best method. This one uses the **Q-learning** algorithm to save which are the best choices, and also uses the **e-greedy** to select its action.

This last approach was used to solve this problem, similar to the study [2]. After implementing every architectures, they were compared between them to see which are more efficient and more quick to get the sheep in lesser time. Besides this they were tested in different examples to show which was the best in various scenarios to allow understand in what conditions the architectures best adapt.

Keywords: Reactive, Deliberative, Learning, Q-Learning, e-greedy.

INDEX

1. Introduction

This project appears due to the necessity to apply the general knowledge acquired in the classes of Autonomous Agents and Multiagent Systems.

The goal of this project is to study how different agent-architectures, namely reactive, deliberative and reinforcement learning, influence the behavior of the agents. We use the Pursuit Problem as the domain and NetLogo as the implementation platform to simulate the behaviors of the agents with the different approaches.

The approach adopted for the realization of this objectives, started by choosing what were the agents and what they represented, followed by the environment, the objectives of the agents and how they relate.

This report is divided in six sections, where we start by explaining the scenario to describe the environment, the actuators, sensors, actions and features of the agents. After that, a section dedicated to the different architectures and several results to explain why and what are the results obtained. In the end we explain our results in general and future modifications that could have been implemented.

2. The Scenario

The agents we implemented consist in four wolves with the role of hunters, that will chase and capture another agent(sheep) designed as prey. The environment chosen is a field represented by a two-dimensional torus, implemented as a grid whose edges are connected.

In this case our hunters will be represented by four wolves and their objective is to surround the prey which is the sheep, who is alone in this world. Each wolf has a field of vision with which they can seek the sheep which is defined by the expression $2d + 1 < n$. Besides this limitation each agent only can move in four or eight directions (depending on the simulation configuration) and only one step at the time.

The environment is inaccessible because there is no agent that can get the full state of the environment, non-deterministic, since each action might have more than one effect, dynamic, because every agent moves at the same instant of time, discrete due to existing a finite number of possible actions and perceptions, and episodic, given that the agent's decisions will affect the decisions the other agents will take and change the outcome.

3. Agent Architectures / Algorithms

3.1 Reactive Architecture

To develop the agents using a reactive architecture we defined the following set of perceptions and actions:

Perceptions:

- See sheep in its depth of field
- Is next to sheep

Actions:

- Move in sheep's direction
- Move randomly
- Stop

The movement of each is wolf is made in two possible ways, either horizontally and vertically only, or with the possibility of diagonal movement added to the first. Contrary to the wolves possible movement pattern the sheep can only move horizontally and vertically. Since the environment is non-deterministic each of these actions may not result in a movement in the desired direction, as collisions are not permitted and are corrected by returning the agent's previous positions

Each wolf decides the action it will take by checking its perceptions in the following order:

1) Verifies if it is next to the sheep, in one of the 4 adjacent positions to the sheep, if so, then the wolf chooses the action **stop**.

2) Verifies if the wolf can see the sheep in it's depth of field radius, if so, it moves in the direction of the sheep.

3) If the wolf is not standing next to the sheep, nor can it see it, then the wolf moves to a random adjacent position, with 75% probability, and does not move, with 25% probability

Each of the agents in this environment possesses two internal variables, **prev-xcor** and **prev-ycor** respective to its previous position in the X and Y axis of the environment. These variables exist only to simplify the correction of the collisions occurring. The collisions are corrected after all the agents execute their actions and are corrected by replacing each agent's position with its position in the previous time-step.

3.2 Deliberative / BDI Architecture

The Deliberative's perceptions will change due to the addition of communication between wolves. Because of this the Deliberative's perception are different in relation to the reactive architecture. So in this case each wolf will seek for other wolves within is depth of field and then transmit the sheep's position(if known). This way the wolves as a pack have an expanded depth of field relative to each one alone.

In this architecture the Wolves set of beliefs is composed only by its knowledge of the sheep's position.

The desires each wolf can have in this architecture are as follows:

- **Stop;**
- **Pursuit;**
- **Search;**

Each wolf can only have one of these desires at the same time being that it verifies its options in the order in which the desires were enumerated.

If the wolf is next to the sheep, it's desire is **stop** , as remaining still is a favorable action to capturing the sheep.

If the wolf is not next to the sheep, but it knows the sheep's position, either by it being in it's depth of field or by communicating with other wolves, then, it's desire is **Pursuit** where the wolf wants to chase down the sheep and move to a capturing position.

If the wolf has no knowledge of the sheep's position, it's desire is to **Search**, where it will execute random actions in search for the sheep or a wolf which knows the sheep's location.

For simplicity reasons we decided to implement the wolves' intentions as being a position to where it intends to go. Depending on its intention this position has to be calculated and shared with the wolves it can see, as to avoid having wolves with the same position as their intentions. This approach contributes to avoiding competition and promoting cooperation and coordination. This sharing is done by each wolf communicating to its visible wolves, its intention and also his knowledge of the others wolves intentions, making the wolves use that knowledge to filter the positions that will not be explored and therefore, avoiding unnecessary collisions when converging in the sheep's position. Although this approach works very well with reducing the number of collisions, it still does not avoid collisions at all when wolves are chasing the sheep and coming from the same side of it, due to sharing common trajectory plans.

Each wolf in its plan possesses the set of positions it has to traverse in order to reach its intended position. These set of positions constitute its trajectory and in order to calculate this trajectory we used an implementation of the algorithm A* using the wolf's position as the starting node, and its intention as the goal node. To avoid considering positions occupied by other agents, these positions had a very high cost relative to free positions in order to avoid collisions.

With this plan the wolf executes the action that permits it to move to the first position in the plan.

Since the environment is dynamic each wolf has to deliberate its intentions and review its plan in each time-step. Due to this constraint each wolf is an open-minded agent, as its desires and intentions have to be reviewed and filtered at the beginning of each time-step, and therefore a plan can never be executed to the end (except for the case in which the plan just has one position in the trajectory). This is negative in terms of path-planning as a complete path has to be calculated at each step and only one action at most will be executed for that plan.

A possible change would be to correct the path-planning, as instead of using the A* algorithm there could be used an algorithm which calculated the nearest, free neighbor position to the sheep and using that position as the plan at each time-step, as to avoid unnecessary calculations when formulating the plan for this kind of agent.

3.3 Learning Component

In this project a third architecture was implemented in which we used reinforcement learning to teach the wolves how to capture the sheep. To implement this architecture we used the Q-learning algorithm with the **ε**-greedy or the greatest-mass algorithm in order to select the action to execute in each state.

For this environment each state had to be modeled as the relative positions of each agent(except itself) to each other agent. This has a nuisance which consists in having a data set for the Q-values of the learner of ((2d+ 1)^2 + 1)^4 , in which d is the wolf's depth of field. This is enormous even with a small depth of field. To overcome this problem we used 3 different learning modules for each sheep-wolf pair state. Each state consisted of the relative position of the sheep and one of the wolves. Each learner would therefore have different Q-values for each state-action pair and the next action would be selected by applying the action selection algorithm for each action in a given state.

The update made to our Q-values using the Q-learning is made by giving reward to the wolves in the following situations:

-If sheep is captured, reward 1 to all wolves

-If sheep is not captured, reward -0.1 to all wolves

We also considered another approach, in which we rewarded each wolf for being next to the sheep in order to "motivate" it to not perform actions that would move the wolf away from the sheep. This resulted in some exploitation from the wolves, in which they would learn that it's better to get near and then move away from the sheep in order to maximize the total reward over a set of time-steps, than to simply try to converge to the sheep's position and stay next to the sheep.

```
ask sheep [
  sheep-loop
]

ask wolves [
  wolf-think-act-loop
]

ask wolves [
  wolf-learning-loop
]
```

Fig 3.4 Code example for the learning architecture

To make sure that each wolf received reward when the pack captured the sheep we separated the action loop from the reward loop, in a manner that, each one of the wolves observed the same state of the world as the others. The ***wolf-think-act-loop*** is where each wolf selects an action using the knowledge acquired with Q-learning algorithm and then executes the selected action. The ***wolf-learning-loop*** is where the wolves update their state, as well as their Q-values using the Q-learning algorithm's update rule.

4. Comparative Study

When testing with a reactive sheep, in which it evaded the wolves by moving in the opposite direction to the average position of its visible wolves, we observed that with the wolves reactive architecture the sheep in some cases could not be captured whenever its depth of field was higher than 1. This phenomenon occurred due to the sheep trying to flee to an occupied position in the following situation(fig.4.1):

Fig. 4.1 movement example

In this situation, the wolf on the top tries to move closer to the sheep in an angle approximate to where the sheep is, and since this angle coincides with an occupied position, the wolf's action will always fail. The sheep will try to move away from the wolves

to the opposite average direction of the wolves, which in this case, is the position to its left, where there is a wolf, and therefore its action will also fail.

Using a depth of field smaller than 2 will result in the sheep never considering moving to a position occupied by a wolf as a possible action.

Our simulations consisted in testing the several architectures elaborated, with constant values to certain variables to verify the main differences about them. In these variables we tried different values and compared them to get different results.

For every architecture it was tested the following variables with different values:

- Wolf Depth Field:
  - This variable represents the range the wolf can see, it was tested with the values of 3 and the maximum possible value for a map size, in combination with the other variables.
- Number of Episodes:
  - This variable represents the maximum number of episodes per simulation. Each episode represents the number of time-steps that will take for the wolves to catch the sheep.
- Size of the Map:
  - The size of the map must be n by n having a minimum size of 3 by 3 to be possible to appear all agents, due to this case being very simple the agents were tested in 5 by 5, 10 by 10 and 20 by 20.
- Diagonal Movement:
  - This variable represents a switch for the possible movements of the wolf. When it's false the predator can only move in the orthogonal axes. In the case of being true, diagonal movements will also be possible as well.
- Sheep Reactive:
  - This variable is also a switch, in this case it changes the architecture of the sheep. It can be either have a random behavior or a reactive behavior similar to the implementation of the reactive wolves's architecture.
- Sheep Movement Probability:
  - This variable represents the probability of the sheep choosing an action in a given step. It was initialized with a value of 25% of movement for the tests.

Below we will show the different results we got to the different architectures and the comparative results between them:

- Diagonal + Orthogonal Movement VS Orthogonal Movement:

While testing either of the architectures, but using a variable type of movement for the wolves, we could verify that for the most part, having diagonal movement resulted in a boost of performance to the wolves time performance per episode, since because of this the wolves could, in the reactive architecture converge faster while chasing the sheep and in the other architectures the wolves enlarged role of actions, gave them the possibility to converge to the sheep's position not only faster but through more time efficient paths.

These results can be backed by the data gathered and shown in the graphics of figures 1 and 3, 4 and 5, 7 and 8, and ,10 and 11.

- Reactive Sheep VS Random Sheep:

One approach we considered interesting to test with each architecture was to add some reactivity to the sheep. In this approach, as was already referred previously, the sheep could react to the wolves threat and move away from them. This resulted in a bigger challenge for the wolves and this can be verified in an increase of time-step average per episode(see figures 1, 2 4, 6, 7, 9, 10 and 12), something that happened no matter the size of the map or the wolves depth of field.

Using this approach we also verified that whenever we used a random sheep with a depth of field bigger or equal to 2, that it became impossible to capture it in a toroidal environment if it's movement probability was of a 100%. Due to this fact we understood that the sheep's behavior in terms of "speed" was as big as a factor as its type of behavior(reactive vs random).

- Different size maps and Depth of Fields:

For the given data, we can see in either architecture the average-time-steps increase several times proportionally to the increase of the size of the map or the decrease of depth of field. This is due to the fact that the wolves are very dispersed in a map and is more difficult to encounter the sheep, so the bigger the map the harder it is to find the sheep. We can observe in the figures 7, 10 and 16 this significant increase.

In the case of the reactive architecture, they do not communicate so it becomes even more difficult than in BDI architecture as shown in the figures 1 and 4.

All the previous cases occur with the same values only changing the size of the map, nevertheless if the value of the depth of field is the maximum possible value, the average-time-steps remain constant for the BDI architectures, due to the existence of communication between the wolves, and only increases the time-steps due to the sheep being more faraway. These can be seen in the figures 13, 15, 17.

There is an additional case in which the variance of the size of the map does not really matter, because if the depth of field of the wolf is equal to one, it cannot perceive the sheep so it will only stop due to the probability and not by seeing the sheep. With this depth of field neither the reactive, or the BDI, or the Reinforcement Learning will be equivalent to make the wolves random.

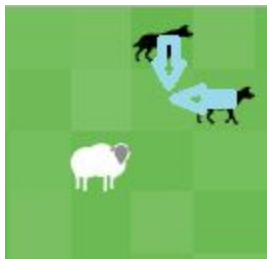
- Sheep's movement probability("speed"):

While testing the several architectures with different values of the sheep's acting probability, we tried with following values, 25%, 50% and 75%.

For the reactive architecture the wolves will increase the time steps to catch the sheep. This situation occurs in the case of the BDI as well. For the Learning agents they take more

time-steps to understand how to catch it, but after that the wolves catch it even faster than the cases before, this is shown in the figures 18,19,20,21,22,23 and 24.

The increase of steps in the reactive is due to the wolves using only the manhattan distance, that is there can be cases where the wolves will be stuck trying to catch the sheep if it moves always in the same direction. For the BDI he only tries to move to a free position near the sheep, nevertheless, if the sheep moves they must recalculate the routes to catch the sheep and they need more steps to catch it. For last, in the Reinforcement Learning after they learn, each wolf will focus in a capture position and this way they will not interfere with the other wolves and catch the sheep more efficiently.



- Number of episodes:

The number of episodes is only relevant for the Reinforcement Learning, this is due to the fact the BDI and the Reactive do not use previous knowledge. So for the Learning agents this number can change the results obtained. This value is particularly relevant for the ε-greedy action-selection algorithm since the epsilon is directly computed from this value. This happens because there is a learning curve, that can change with the number of episodes, so for the agents to learn they must execute several steps until they understood which is the best path to achieve the objectives, after that number of episodes, this value has little relevance.

5. Conclusions

We set out to compare reactive, deliberative and learning agent-architectures, using the Pursuit Problem as our model. We built the environment, all three architectures and an interface, that allowed us to easily parameterize the simulations, on top of NetLogo. We made use of the interface and tested several hypotheses under all architectures. Through that we were able to replicate the proposed paper's findings.

We compared the different architectures in terms of performance, ease of implementation, degree of behavior control and understanding and scalability to the general problem. The reactive architecture was the easiest to implement and the easiest to control its generated behavior but had the worst performance. The deliberative architecture was harder to implement, since it required us to come up with a strategy to solve the problem, easy to control its generated behavior and had good performance. The learning architecture was the hardest to implement, due to its inherent complexity, has the best performance after a period of learning, the hardest to control the generated behavior and it was the most general solution.

Since the wolves have limited vision and have to cooperate to achieve their goal and it was expected that the purely reactive architecture was going to have difficulties in solving the problem. To overcome these restrictions, we moved on to a deliberative architecture that leveraged memory and communication to overcome the limited vision and cooperation problems, which clearly increased the performance of the wolf pack. At last we tried learning that showed us how it was possible to have coherent strategy without communication by having functionality-specialization in fixing the individual capture position.

6. References

[1] M. Benda, V. Jagannathan, and R. Dodhiawala. “On optimal cooperation of knowledge sources - an empirical investigation”. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington, July 1986

[2] N. Ono and K. Fukumoto. “Multi-agent reinforcement learning: A modular approach”. In Second International Conference on Multiagent Systems, pp. 252-258, Kyoto, 1996.

[3] *Instituto Superior Técnico Agentes Autónomos e Sistemas Multi-Agente, Official webpage*. Retrieved from: http://tecnico.ulisboa.pt/disciplinas/AASMA-2. Last accessed May 2016.

7. Annexes

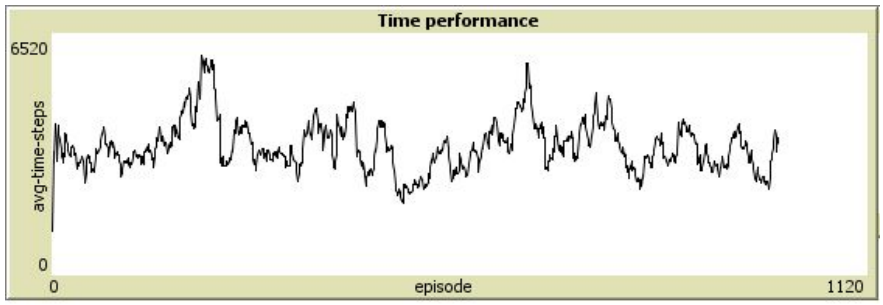


Fig.1 : Time performance for 10 by 10 map with reactive architecture

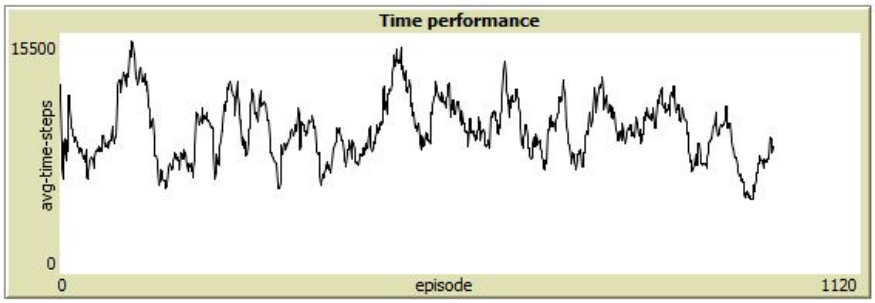


Fig.2 : Time performance for 10 by 10 map with reactive architecture and reactive sheep behavior

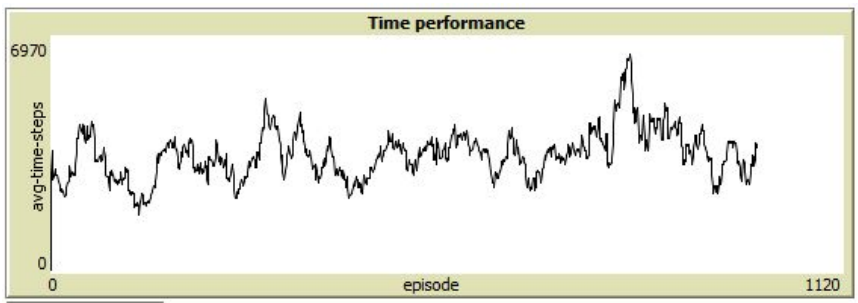


Fig.3: Time performance for 10 by 10 map with reactive architecture and Diagonal movement

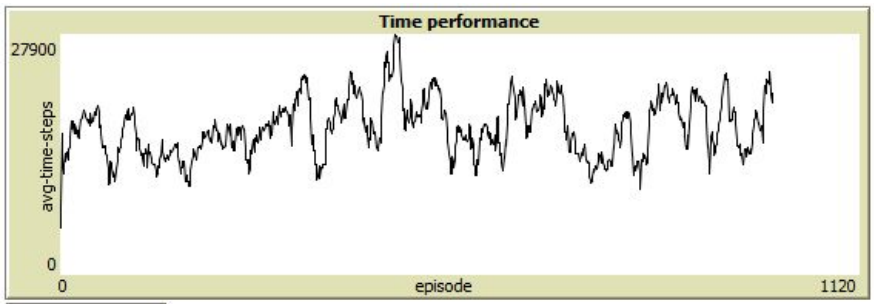


Fig.4: Time performance for 20 by 20 map with reactive architecture

Fig.5: Time performance for 20 by 20 map with reactive architecture and Diagonal movement

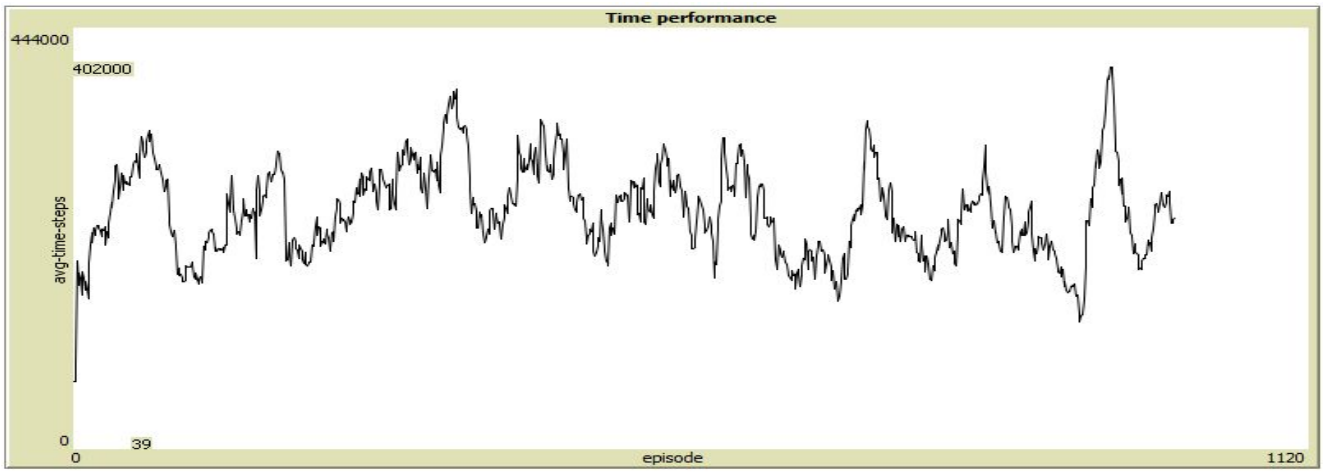


Fig.6: Time performance for 20 by 20 map with reactive architecture and reactive sheep behavior

Fig.7: Time performance for 10 by 10 map with BDI architecture

Fig.8: Time performance for 10 by 10 map with BDI architecture and Diagonal movement

Fig.9: Time performance for 10 by 10 map with BDI architecture and reactive sheep behavior

Fig.10: Time performance for 20 by 20 map with BDI architecture

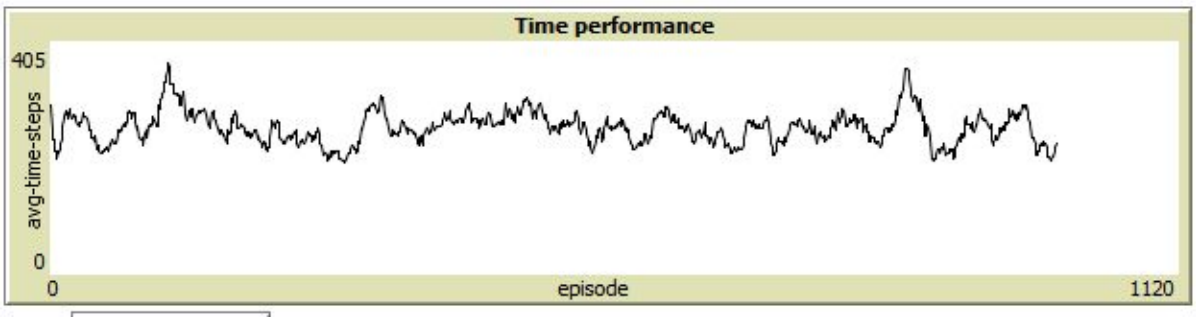


Fig.11: Time performance for 10 by 10 map with BDI architecture and Diagonal movement

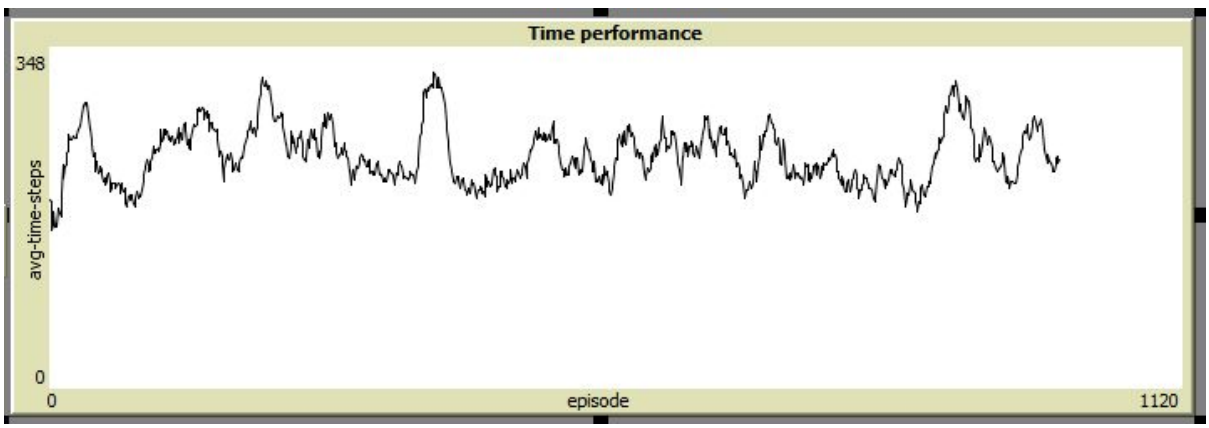


Fig.12: Time performance for 10 by 10 map with BDI architecture and reactive sheep behavior

Fig.13: Time performance for 10 by 10 map with BDI architecture with maximum depth of field

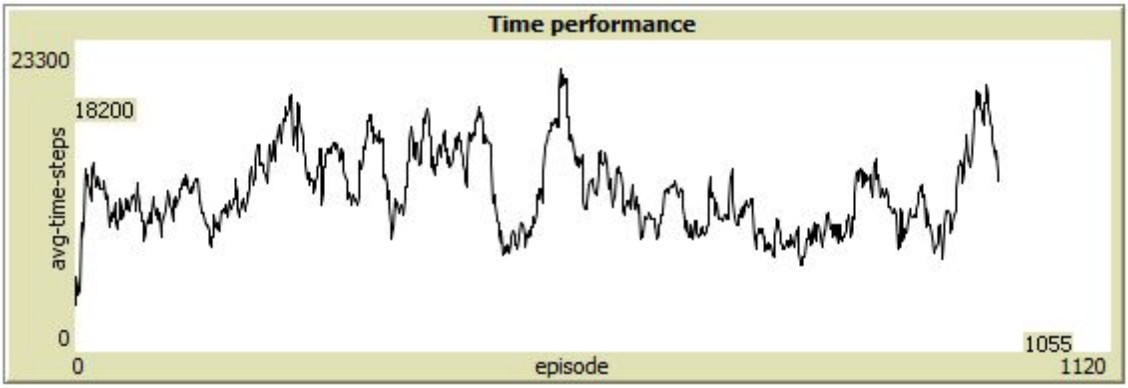
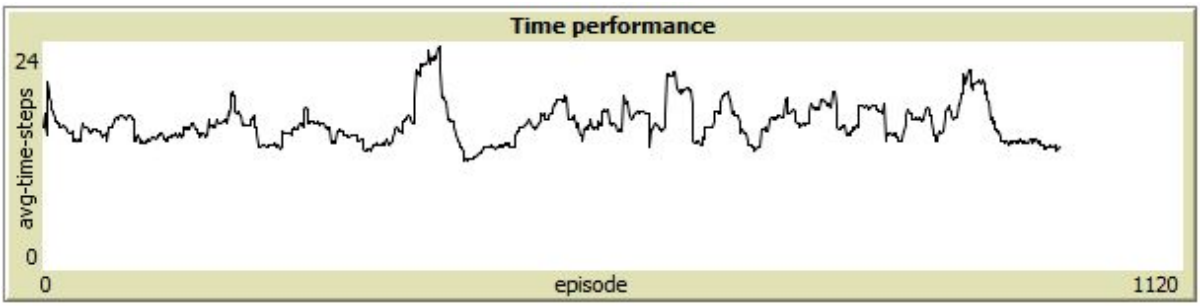


Fig.14: Time performance for 10 by 10 map with reactive architecture with maximum depth of



field

Fig.15: Time performance for 20 by 20 map with BDI architecture with maximum depth of field

Fig.16: Time performance for 30 by 30 map with BDI architecture

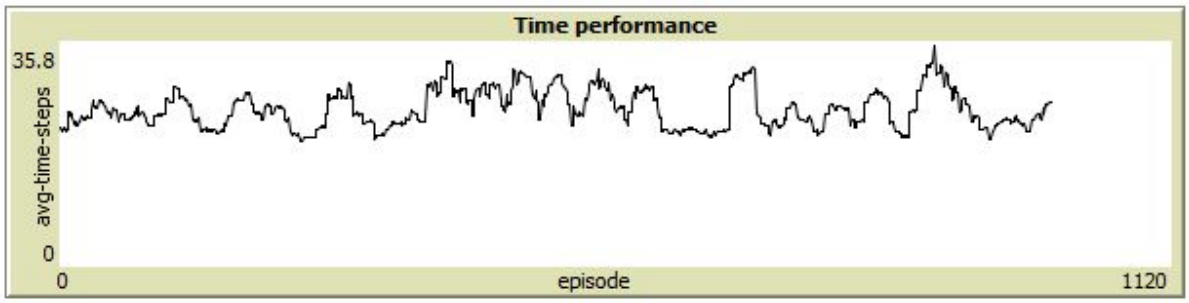


Fig.17: Time performance for 30 by 30 map with BDI architecture with maximum depth of field

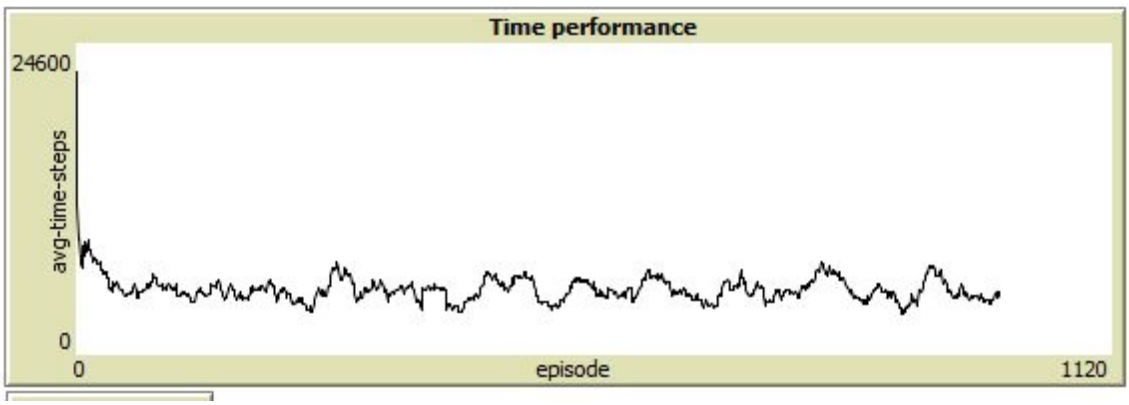


Fig.18: Time performance for 10 by 10 map with reactive architecture with 50% Sheep movement probability

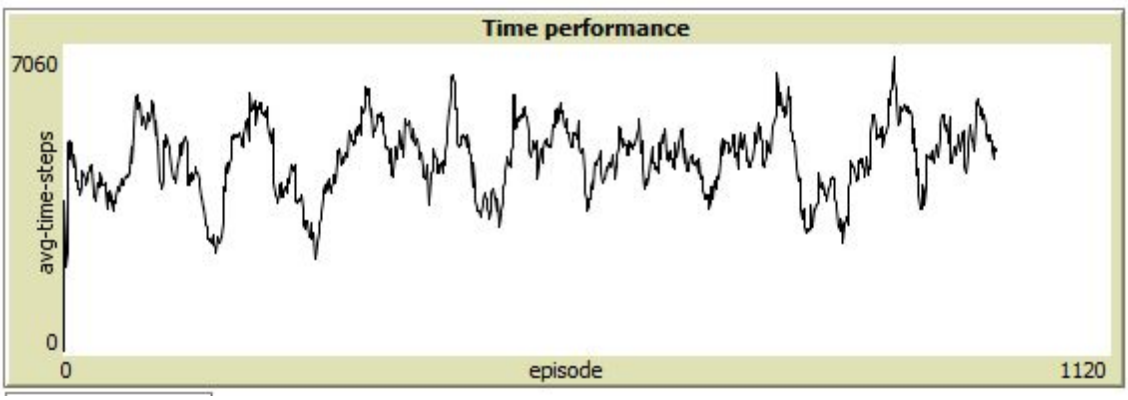


Fig.19: Time performance for 10 by 10 map with reactive architecture with 50% Sheep movement probability

Fig.20: Time performance for 10 by 10 map with BDI architecture with 50% Sheep movement probability

Fig.21: Time performance for 10 by 10 map with BDI architecture with 75% Sheep movement probability

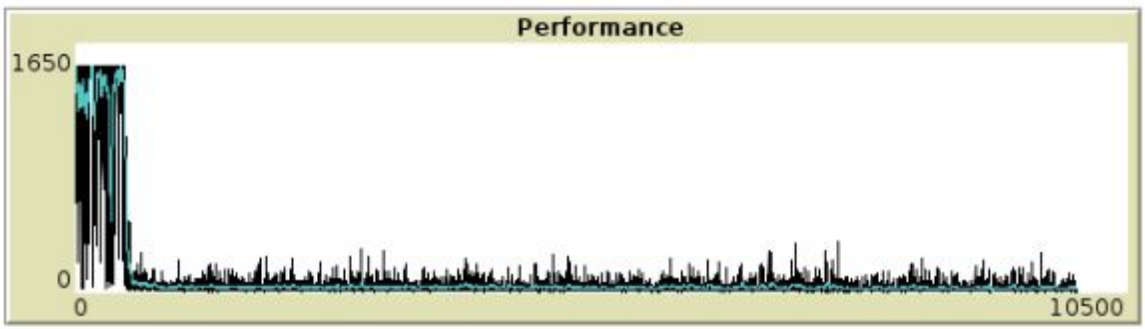


Fig.22: Time performance for 10 by 10 map with Learning architecture with 25% Sheep movement

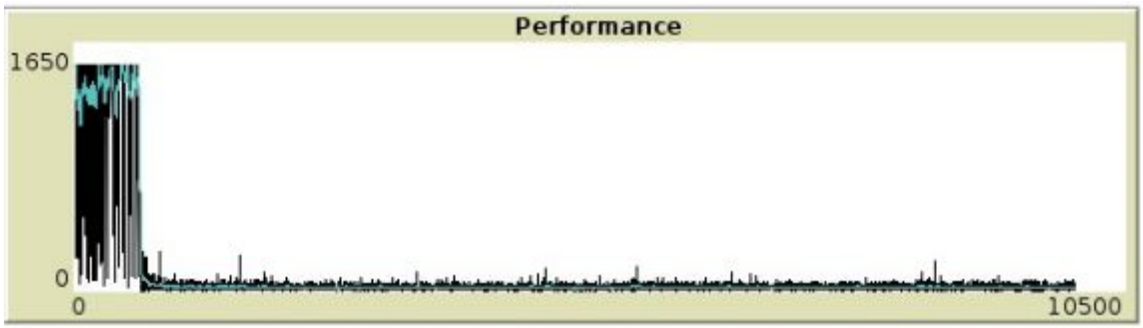


Fig.23: Time performance for 10 by 10 map with Learning architecture with 50% Sheep movement

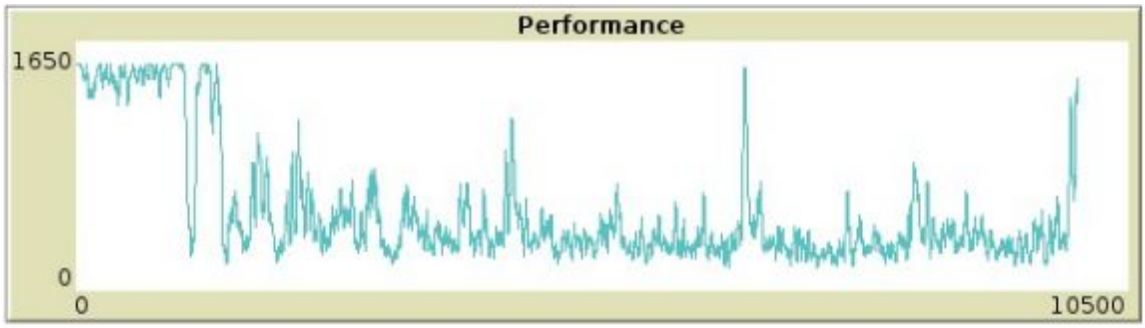


Fig.24: Time performance for 20 by 20 map with Learning architecture with 25% Sheep movement