Learning a representation of a believable virtual character’s environment with an imitation algorithm.

Fabien Tencé, Cédric Buche, Olivier Marc, Pierre de Loor

To cite this version:

Fabien Tencé, Cédric Buche, Olivier Marc, Pierre de Loor. Learning a representation of a believable virtual character’s environment with an imitation algorithm.. First annual Pan-Arabic International Conference on Intelligent Games and Simulation (GAMEON-ARABIA’10), Dec 2009, Egypt. pp.141-145. hal-00784296

HAL Id: hal-00784296
https://hal.archives-ouvertes.fr/hal-00784296
Submitted on 4 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
LEARNING A REPRESENTATION OF A BELIEVABLE VIRTUAL CHARACTER’S ENVIRONMENT WITH AN IMITATION ALGORITHM

Fabien Tencé*,**, Cédric Buche*, Pierre De Loor* and Oliver Marc**

* UEB – ENIB – LISyC
** Virtualys
Brest – France
{tence,buche,deloor}@enib.fr, olivier.marc@virtualys.com

ABSTRACT

Improving the believability of the characters populating a virtual environment is one way to make users feel in the simulation. The model developed in (Le Hy et al. 2004) seems to be a good base to generate believable behaviours: it is close to what gives quite good results in the industry but also uses probabilities and a learning algorithm which could make the illusion of believability last longer. This article first describes how this model works. As it uses a manually-defined representation of the character’s environment, improvements can be done to that representation to enhance its autonomy and believability. We propose to use an other model named growing neural gas to learn by imitation the representation of the environment. The way this model was implemented and the evaluation of the quality of the learned representations are then detailed. Ideas about improvements for the growing neural gas to give more information to Le Hy’s model are given in the conclusion.

KEYWORDS

Autonomy, believability, behaviours, imitation learning, topology.

INTRODUCTION

Simulation in virtual environment make the user feel like being in the environment and allows him to act in it. To be complete, entities in these environments must have a believable behaviour (Bates 1994). To generate those behaviours, techniques from classic artificial intelligence are mainly based on behaviour rules defined a priori by designers. For the models to perceive the environment, designers have to create a representation of the environment which is also often defined a priori.

This manual work cannot be done for every new character and environment. We propose that our entities will be able to learn, for them to be autonomous and have believable behaviours. This learning will be unsupervised and online: the entity will learn while it acts. This will both remove the burden of parametrizing the models controlling the characters and increase believability by having real-time evolution of the behaviour.

To be able to achieve the best believability, we want the agents to do like human-controlled virtual characters. Indeed, there are no better example of what a believable behaviour is than a human behaviour itself. It is this kind of learning, by example (Del Bimbo A., Vicario E. 1995) or by imitation (Gorman and Humphrys 2007; Bauckhage et al. 2007) we want to use to model believable and autonomous characters.

The way the characters act and learn depends heavily on the kind of virtual environment they are in. We share issues with the video games industry because the game designers want the players to be immersed in the simulation too. They are trying to be as close as possible from reality, making rich and complex environments. Researchers can avoid some technical difficulties (rendering, physics, networking, etc.) by using such games. They can then focus on the making of the entities they want to study (Cavazza et al. 2003; Mac Namee 2004). Furthermore, video games being made for human beings, they offer a real challenge for the entities to be believable. An other advantage is that they offer experts of these environments, human players, which can give pertinent criticism on the entities’ behaviours (Silverman et al. 2006).

This article first presents a Bayesian model developed in (Le Hy et al. 2004) which can be a good base for controlling a believable character. The representation of the environment must fit into the model and must be learned by imitation for both autonomy and believability purposes. A graph model named growing neural gas is then introduced to learn by imitation such representation. After that, the characteristics and qualities of the learned representations is assessed by different measures. To conclude, some enhancements are proposed for the growing neural gas to give more information to the agent’s model.
A PROBABILISTIC MODEL FOR BELIEVABLE AUTONOMOUS CHARACTERS

Models which use probabilities have two advantages. First they add some unpredictability to the behaviour which can improve the believability. Second, when using imitation learning, probabilities can express uncertainty about what the demonstrator really intended to do, making the learning more robust.

A Bayesian model (Le Hy et al. 2004) has been developed for characters in video games. The advantage of this model is that it is quite easy to modify so that the character act as wanted. It is also possible to learn the parameters of the model by imitation. The believability of the behaviours has not been carefully evaluated but some preliminary tests show that it is comparable to models from industry.

We believe that choosing a model, more complex than what is used in industry but less complex than what is usually developed in research is a good idea to generate believable behaviours. With that approach, we can try to have more complex behaviours than what is done in video games but still being able to understand and modify the internal parameters to achieve the best believability.

In Le Hy’s model, the agent has sensors named $S_0, ..., S_n = S$ which give information on internal and environment’s state like for example the character’s inventory and the position of another character. To act, the agent has motor commands named $M_0, ..., M_p = M$ which can be rotation, jump commands, etc.. To direct the character’s behaviour, the notion of task has been introduced, the variable is named $T$ which can have different values like searching for an object or fleeing.

The value of $T$ is chosen knowing the value of the sensors, following the probability $P(T|S)$, and knowing the previous task, following the probability $P(T^n|T^{n-1})$. Thus the value of $T$ is chosen following the probability $P(T^n|S^nT^{n-1})$, the task model, computed using the two previous probabilities. As $S$ is the conjunction of $n$ variables, to reduce the complexity Le Hy introduce the notion of inverse programming: $P(T|S)$ is computed using $P(S_i|T)$ (and not $P(T|S_i)$) as they are supposed to be independent, which is a strong hypothesis.

Once the value of $T$ is chosen randomly following the probability $P(T^n|S^nT^{n-1})$, the model must decide which motor command should be activated. The value of each motor command is chosen following the probability $P(M_i|S)$, the motor model. It is equivalent to having $P(M_i|S)$ for each value of $T$. Again, to reduce the complexity, Le Hy introduce the notion of fusion by

\[
P_n(T^n|T^{n-1}) = \frac{1}{Z} \left( P_{n-1}(T^n|T^{n-1}) + \Delta P_{n-1}(T^n|T^{n-1}) \right)
\]

enhanced coherence. Each command is computed separately then they are combined using using the formula $P(M_i|S) = \frac{1}{Z} \prod_j P(M_i|SjC)$ ($1/Z$ is a normalization factor). $C$ is how the $P(M_i|SjC)$ command should be taken into account. If the command must be followed, the order is prescriptive. If the command must not be followed, the order is proscriptive. If the command have not to be followed, the order is not taken into account. For this 3 commands, only the probability $P(M_i|SjC)$ is needed because we have the following equality:

\[
P(M_i|SjC) = \begin{cases} P(M_i|Sj) & \text{if prescriptive order} \\ 1 - P(M_i|Sj) & \text{if proscriptive order} \\ \text{Constant} & \text{if order not taken into account} \end{cases}
\]

Thus the model is composed of three types of parameters whose relation is summarized in figure 1:

- $P(T^n|T^{n-1})$
- $P(S_i|T)$
- $P(M_i|SjT)$

Those parameters can be specified manually or learned by imitation. Results seems to be better in term of believability and performance with learned parameters (Le Hy et al. 2004). The imitation is done by observation of the virtual representation of the player, his avatar. By monitoring at each time step the value for $S$ and $M$ it is possible to update the value of the parameters. The learning algorithm developed by Le Hy is based on (Florez-Larrahondo 2005) but only updates the task model, the parameters $P(T^n|T^{n-1})$ and $P(S_i|T)$.

This algorithm, a modified version of the incremental Baum-Welch, updates at each time step $n$ the parameters with the following formula:

\[
P_n(T^n|T^{n-1}) = \frac{1}{Z} \left( P_{n-1}(T^n|T^{n-1}) + \Delta P_{n-1}(T^n|T^{n-1}) \right)
\]

\[
C = \prod_j P(M_i|SjC)
\]

\[\text{if prescriptive order}\]

\[\text{if proscriptive order}\]

\[\text{if order not taken into account}\]

Figure 1: Summary of the influences between model’s variables (Le Hy et al. 2004).
The principle of the GNG is to modify its graph, adding or removing nodes and edges and changing the nodes’ position for each input of the demonstrator’s position. For each input the closest and the second closest nodes are picked. An edge is created between those nodes and the closest node’s error is increased. Then the closest nodes and its neighbours are attracted toward the input. All the closest node’s edges’ age is increased by 1 and too old edges are deleted. Each Δ input a node is inserted between the node with the maximum error and its neighbours having the maximum error. At the end of an iteration, each node’s error is decreased by a small amount.

The version we use is a bit modified to give better results for our needs as shown by figure 2. Instead of inserting a new node each λ input, a node is inserted when a node’s error is superior to a parameter MAX_ERROR. As each node’s error is reduced by a small amount ERROR_DECAY for each input, the modified GNG algorithm does not need a stopping criterion. Indeed, if there are many nodes which represent well the environment, the error added for the input will be small and for a set of inputs, the total added error will be distributed among several nodes. The decreasing of error will avoid new nodes to be added to the GNG resulting in a stable state. However if the player which serves as a example, the demonstrator, goes to a place in the environment he has never gone before, the added error will be enough to counter the decay of the error, resulting in new nodes to be created.

This algorithm has 5 parameters which influence the density of nodes, the quality of the representation, the adaptivity and the time to converge:

- The attraction applied to first toward \((x, y, z)\)
- The attraction applied to first's neighbours toward \((x, y, z)\)
- The nodes’ error decay, ERROR_DECAY
- The nodes’ maximum error, MAX_ERROR
- The edges’ maximum age, MAX_AGE

The nodes learned by this model can be used directly by Le Hy’s model. However, the information given by the edges cannot be used as it denotes only proximity
nodes ← {} 
edges ← {} 

while demonstrator plays do 
  (x,y,z) ← demonstrator’s position 
  if |nodes| = 0 or 1 then 
    nodes ← nodes ∪ {(x,y,z,error=0)} 
  end if 
  if |nodes| = 2 then 
    edges ← {(nodes,age=0)} 
  end if 
  first ← closest((x,y,z),nodes) 
  second ← secondClosest((x,y,z),nodes) 
  edge ← edges ∪ {{first,second},age=0)} 
  first.error+=||(x,y,z)-first|| 
  ∀ edge ∈ first’s edges, edge.age++ 
  Delete edges older than MAX_AGE 
  Attract neighbours(first) toward (x,y,z) 
  ∀ node ∈ nodes, node.error-=ERROR_DECAY 
  if first.error > MAX_ERROR then 
    maxErrNei ← maxErrorNeighbour(first) 
    newNode ← between(first,maxErrNei) 
    first.error/=2, maxErrNei.error/=2 
    newError ← first.error+maxErrNei.error 
    nodes ← nodes ∪ {(newNode,newError)} 
  end if 
end while 

Figure 2: Algorithm used to learn the topology of the environment represented by a growing neural gas.

and not a path between them: nodes can be close but there may be an obstacle between them. To evaluate the quality of the representation, find good parameters, we implemented the GNG.

EVALUATION

We used the game Unreal Tournament 2004 because it features quite complex environments and because human players can control avatars in the game so the GNG can learn for them. We have to choose the parameters in an empirical way because we cannot find them analytically nor use an optimization algorithm. Indeed, our goal is believability and it can be only measured with human judge. This kind of evaluation is not suitable for optimization. The best parameters we found are:

- Attraction force applied to first is 0.03 times the vector \((x, y, z) - first\)
- Attraction force applied to first’s neighbours is 0.0006 times the vector \((x, y, z) - second\)
- Nodes’ error decay is 10
- Nodes’ maximum error is 20000
- Edges’ maximum age is 75

To compare with other environments, the position in Unreal Tournament is given in Unreal units (1 meter is roughly equal to 50 Unreal units) and all the parameters are based on a demonstrator’s position in Unreal units.

With those parameters we trained 2 GNG on 2 different maps. The first one is a simple map, called Training Day, it is small and flat which is interesting to visualize the data in 2 dimensions. The second one, called Mixer, is much bigger and complex with stairs, elevators and slopes which is interesting to see if the GNG behave well in 3 dimensions. The results is given in figure 3 for the simple map and in figure 4 for the complex map.

Figure 3: Result of a growing neural gas learned from a player for a simple map, top view.

Figure 4: Result of a growing neural gas learned from a player for a complex map.
To study the quality of the learned topology, we first chose to compare the GNG’s nodes with the navigation point placed manually by the map creators. Of course, we do not want the GNG to fit exactly those points but it gives a first evaluation of the learned representation. In our case we have those navigation points but our goal is that they are no longer necessary for a character to move in a new environment. Figure 5 shows both the navigation points and the GNG’s nodes. As we can see, the two representations look alike which indicates that the model is very effective in learning the shape of the map. However, there are zones where the GNG’s nodes are more concentrated than the navigation points and other where they are less concentrated. We cannot tell now if it is a good behaviour or not as we should evaluate an agent using this representation to see if it navigate well. Even in the less concentrated zones, the nodes are always close enough to be seen from one to another, so it should not be a problem.

As the attraction applied to the nodes for each input is constant, the GNG is not converging to a stable state. This is a wanted behaviour, allowing the GNG to adapt to a variation in the use of the map: if the professor suddenly uses a part of the map which he/she has not explored yet, the GNG will be able to learn this new part even if the GNG has been learning for a long time. We do want, however, the GNG to learn quickly the topology and to keep a good representation of the world over the time.

To study the time evolution of the GNG’s characteristics, we introduce a distance measure: the sum of the distance between each navigation point and its closest node. We also study the evolution of the number of nodes because we do not want the GNG to grow indefinitely. Figure 6 shows this two measures for the simple and the complex maps. For the simple map, the GNG reached its maximum number of node and minimum error in approximately 5 minutes of real-time simulation. For the complex map, it takes more time, about 25 minutes, but results at 12 minutes are quite good. Those results show that it is possible to have an agent learn during the play.

The GNG can handle inputs from multiple professor. Figure 7 shows the distance and number of node for a GNG trained on 1 professor and for a GNG trained on 4 professors. The learning with 4 professors is, as expected, faster: about 3 minutes for the distance to stabilize instead of 5 minutes for 1 professor. It is interesting to note that the learning is not 4 times faster but the gain is still important. Learning with multiple professors seems to give a GNG with less variation during the learning. The gain have however a small drawback: the number of nodes is a little superior for multiple professors. It may be due to the fact that professors are scattered in the environment instead of a unique professor following a path.

It is interesting to compare two learned GNG on the same demonstrator in the same environment and conditions but for 2 different simulations. The goal is to see if the two representations fit. Figure 8 shows that the resulting GNG are a bit different. The first GNG
The last evaluation assesses the impact of the frequency at which the demonstrator’s position is given to the GNG. For the previous experiments, the frequency was set at 10Hz. Figure 9 shows the differences for 1, 10 and 100 Hz. Results indicates that 1Hz give comparable results to 10Hz but it takes much longer to give a good representation. At 100Hz, the GNG reaches a stable state as fast as at 10Hz but the resulting GNG has much more nodes resulting in a lower error. Using a high frequency is therefore not useful because the number of nodes can be increased changing the MAX_ERROR and ERROR_DECAY parameters without using computing power.

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Time</th>
<th>Number of nodes</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Hz</td>
<td>1h30</td>
<td>22</td>
<td>3800 UU</td>
</tr>
<tr>
<td>10Hz</td>
<td>5min</td>
<td>24</td>
<td>3300 UU</td>
</tr>
<tr>
<td>100Hz</td>
<td>5min</td>
<td>39</td>
<td>2300 UU</td>
</tr>
</tbody>
</table>

Figure 9: Comparison of growing neural gas’ characteristics learned at different frequencies on a simple map (Training Day). Time, number of nodes and error are given when the growing neural gas reach a stable state. UU stands for Unreal units.

The growing neural gas proves to be very efficient at learning the topology of an environment by imitation. The learning is quite fast, up to 25 minutes, and is done during the demonstration of the professor. The model does not converge to a unique and stable solution but gives similar solutions for different runs. It can adapt quickly to changes in the use of the environment, the agent being capable of evolving while playing. However the information extracted from the GNG is only based on the position of the nodes. It should be possible to learn much more.

**IMPROVEMENTS**

The biggest difference between the GNG we implemented and navigation graphs coded manually is that graphs give also information on the accessibility of a node from another. Edges in a GNG gives only an information on proximity but there can be a obstacle between two nodes joined by an edge. However in the experiments the edges seldom crossed a wall so they could be used for paths. An idea could be to store the previously activated node and create an edge to the current activated node. Like the GNG edges we should make the edge age and disappear if they are too old. Whether those edges should replace the GNG edges could be an interesting experiment to set up.

To share more information with the behaviour model, we can introduce the table $P(M|TN)$, giving the best actions to do, knowing the task $T$ when the character
is near node $N$. The agent could then retrieve those probabilities and mix them with $P(M|ST)$. This process is quite similar to the process of tagging: designers often annotate navigation points with information such as “jump spot” or “covering”. The learning of those $P(M|T)$ probabilities should be done during the Baum-Welch algorithm to have an approximation of the $P(T)$ probabilities.

Another interesting information to learn is the area covered by each node. If the professor pass exactly at the position of the node or if there is a big variation in the distance to the node is an important information. The nodes’ error give a bit of information about this distance, however with the error decay, this information is lost over time. This information is quite difficult to learn because each winner node moves so it does not represent the same area. It could be possible to update the influence radius of winner and its neighbours according to their current radius and the distance to the example. The exact formula is yet to be found.

CONCLUSION

Virtual environments, like for example video games, need believable characters for users to feel in the environment. Le Hy’s model seems to be a good base to control those characters, focusing on believability and trying to produce quite complex behaviours. To improve the agent’s behaviour, we decided to use a growing neural gas to learn by imitation the topology of the environment. We believe that it will make the agent use the environment in a more human-like fashion. It also removes the burden from the maps designers of placing manually the navigation graph.

Our first evaluations tend to show that the growing neural gas give a good representation of the environment. The learning is fast, with one professor it takes up to 25 minutes to learn a representation of the whole environment. As it is possible to learn with several professors, learning can be done very quickly. Although different runs gives different results, the representations are very similar.

With this ability to learn the environment, the agent can be placed in any simulation without a priori knowledge and still be able to move by imitating human users. As the learning is quite fast, users could perceive the evolution in the way the agent acts and thus believing it can be human. The growing neural gas gives autonomy and believability to the model.

The next step is to put more information in the growing neural gas, learning which node is accessible from each node and mixing motor probabilities with the one computed by Le Hy’s model. To see if this work really gives results we will have to test the difference in the behaviour between an agent using the navigation points and agent using the growing neural gas.

REFERENCES


