

# CSE 471/571 Semester Report: Agent 20: Living, Surviving and Thriving in a Simulated World

Will Ryan

December 18, 2003

## Abstract

Running simulations of agents that can only interact with the world, essentially ignoring other agents, cannot possibly lead to effective testing of these agents. In fact, unless agents use other agents as a source of learning, then it may be very difficult for agents to find an equilibrium in the environment such that they can survive and possibly be around to create another generation of agents. It is up to the total community of agents to share information about the environment with all agents in it and ensure its survival. A community-based learning approach is necessary for agents to truly find an equilibrium point with the environment.

## 1 Introduction

The problem that was being studied in this project was how to allow agents to learn in a simulated environment. The obvious first way is to just interact with the environment, look at how the interactions benefited or harmed the agent, and allow the agent to make a judgment about such an action again in the future. Another method is to tell the agent to do some action and tell them what the expected result of such an action will be. Finally, allow agents to interact with environment but also allow them to interact with other agents and teach each other things that could benefit the other agents.

For this project, the last method was what was implemented. It gave the most opportunity for an agent to learn about the environment. The reason for this is because even though an agent may never have experienced a situation itself, it can still learn that certain actions are really good in some situations and others are really bad. This ensures that a community, and thus each of its individual agents, can learn more quickly to avoid doing things that put the life of its agents in jeopardy.

The rest of the paper will contain the following: a detailed problem description, details on how the project was implemented, possible experiments to be performed on the system, and a conclusion section that will also give a list of improvements for the project. The problem description will be exactly how the problem is defined, including the world, items in the world, and the agents. It will also talk about the knowledge base of each of the agents and two possible ways it can be

implemented. The implementation section will discuss in detail the model of community-based learning and also the implementation of the knowledge base using the Belief, Desire, and Intention architecture. It will also give an explanation on why this solution works. The last two parts will be about possible experiments on the system that could be performed to test the affectiveness of the system versus agents that only learn through another means of learning and observations about the system so far. The conclusion will include improvements that could allow us to make our agents perform even better.

## **2 Problem description**

For the project, everything that would define the problem was built from scratch, so that everything that was defined in the system; the world, items, and agents; could be changed to allow full control over it. There were 4 main parts that governed this problem. The first was the environment itself with which the agents interacted. The environment needed to allow the agents to move around autonomously in the world and interact with agents and items that are located in it. The next factor was the items. Each item needed to affect different needs (or possibly the same needs but by differing amounts), so that the agent could learn that having or doing actions with different items will affect them differently. The items would need to have a class type of some sort, in order for the agent to make general statements about items of each class. For simplicity sake, we would also need to make sure that the items would get replenished in the world from time to time. This was to prevent items for not being available after the first couple of rounds in the simulation.

The third problem had to do with the agents. The agents needed to have needs that would determine how satisfied they were. The agent would always need to be looking to satisfy one of its needs, even if they were all relatively high (satisfied). Eventually, the agent would start to get older and would need to shift the focus of what its needs were. The system that would be implemented for the agent would need to take care of these things as well as internally (such that the agent would not be aware of it) take care of its own biological changes (i.e. after it eats an apple, change its needs and health accordingly, or after time, lowering all of its needs overall). In addition to all that, the agent would also need to have an interface to the knowledge base and interpret what it is telling it to do.

### **2.1 Knowledge Base**

The last problem had to do with the knowledge base. We needed a knowledge base that could let the agent formulate what action it should take in a given situation. It would need to allow for possible activities, but to also notify the agent which of them would be the best to take. The system would also need to allow an agent to learn, both from experience and from other agents. Since the agent is working in a partially observable environment, it would also need to have absolutely no knowledge about the items in the world or the world itself a priori. It would need to keep track of what items (and other agents for the same reason) before it can make statements about them. That would constitute actually knowing of the item or other agent that you talk about in knowledge base

statement. The last thing that we would need the knowledge base to do is to create a system by which an agent can give value to certain items that they know about.

There are several ways in which all of this could have been implemented. The two best are to either use a Belief, Desire, and Intention architecture to represent choices for actions in the knowledge base. The second could be to use a first order logic system with unification. Both offer several benefits to allow the agent to be more expressive of how it reasons.

The first proposed idea would be to use a Belief, Desire, and Intention architecture to represent the statements in the knowledge base. It would need to be modified to fit our system from what it typically is used for. First of all, for simplicity, we will ignore the idea of planning for an agent. The system will only be concerned with single actions to satisfy the current need of the agent. Next, we need to redefine what each of the parts of the architecture mean in our system. The belief part of it would mean anything that the agent can take as given (such as what it can see or what it has in its pocket). This means that belief is anything that the agent can tell from the environment. The desire is the need that needs to be satisfied. Finally, the intention is what action (and if applicable what object to use for the action) can be performed to satisfy this need given the current belief about the environment. Using this system, a constraint satisfaction system can be put in place to compare the agent's current need and the current environment conditions in order to find several knowledge base statements that are similar fit these conditions.

The second proposed option is to have a first order logic system that will have a unification subsystem. This system would allow us to define actions in the knowledge base rather than have them ungrounded. We could then classify objects using characteristics that are defined in the knowledge base. The idea would be, then, that as an agent learns an action, it could try to classify it using some of the definitions that have been provided to it already and could try to establish which action will evaluate to true, we could then return all such statements to the system querying the knowledge base.

The benefits of the first idea are that it would be much more simple to implement and allow us to associate weights to each of the statements based on the current situation. The problem is that this system may cause a lack of expression that would cause us to write many more lines in the knowledge base than we need (ultimately causing a slower performance overall). The first order logic system would be much more expressive but it would be much more difficult to implement from scratch.

## **3 Learning Through a Community Effort**

### **3.1 Description of Solution**

For the majority of the project, we implemented everything separately and brought it together keeping track of what between each of the subsystems, which each group member created, needed. Towards the end of the project, we finally brought everything together. This was to allow each

group member to focus on one specific aspect of the project and understand, once the simulation was run and the code needed to be tweaked, why something may not have been working right.

To begin the project, one group member created a world by using a matrix of vectors. This matrix would represent a grid environment for the agents to wander around in and the vectors would be a listing of what was in each square in the grid. This made it easy to represent where everything was in the world without letting it become too crowded such that new items and agents could be placed in the world.

The next part of the project was to create a need system for the agents. Each need could be satisfied by various actions using various objects in the environment. The needs that were defined for the simulation were nutrition, energy, cleanliness, need to waste, need to reproduce, need to have knowledge, need to have objects, need for others to have knowledge, and need for others to have objects. These actions were enough to define the agent and give it an idea of how it should behave. We defined also several actions that could be used to satisfy these needs. The actions that were defined are eat, rest, waste, reproduce, take, drop, exchange, give, teach, teachme, and searchfor. Some actions may do different things based on who or what the agent interacted with.

The next focus of the project was on the agent and its biology. The agent needed to define certain internal substructures that would take care of its internal workings. We defined a concept of health to make sure that the agent will eventually die if it is not satisfying its needs. This was a nice control in the system to give incentive to the agents to satisfy their needs. The system would also control the needs of an agent such that over time all the needs will decrease and thus become more important for an agent to satisfy them.

## **3.2 Implementing the Knowledge Base**

The last thing the group focused on was the knowledge base. Early on the decision was made to let them learn through their interactions with the environment and also to allow them to interact with other agents in the environment. Due to that fact and that the agents needed to interact with objects they don't really know about, the decision was made to use the BDI architecture. This system would be easier to implement for learning and for make decisions about what action to take in a given circumstance. The first order logic system would be too difficult to implement. Furthermore, given the issue of unification in the knowledge base would cause for dangerously unstable agents. For instance, one agent may easily be able to unify for some variable in some statement that it checks against the knowledge base. Now suppose that it teaches another agent part of what it knows. There is no way to tell that the unification will work correctly because it may refer to some item that is defined in the first agent but not the second. When the second agent tries to check some statement against its knowledge base, it may break.

To avoid possible conflicts, the BDI architecture was used. A statement in the knowledge base was defined by "have X & see X & need : Y  $\zeta$  action X" where X is some object (or possibly another agent) and Y is the most pressing need of the agent. When we initialized a new knowledge base, we filled it with several of these statements that were templates of what they should do if

they don't know anything about the world around them. They have the word "object" to refer to some unknown object in the world or "other" to refer to another agent ("self" referred to the agent himself.) As the agent started to interact with different things and learning about them, it could add more specific statements with the actual name of the object or agent id in a knowledge statement. This meant that an agent would also need to keep track of what objects and agents it knows of. This was accomplished using two vectors: one for items and one for agents.

The knowledge base was broken into three main classes. The KnowledgeBase class is the main class that manages all the basic functionality of the knowledge base. The actual Knowledge Base itself was contained in this object as a vector. Within that vector, Knowledge Statements were stored that held each individual statement. The KnowledgeStatement class was created to store these statements. For searching the knowledge base to choose an action, rather than returning one possible action, a list of actions was returned just in case there were multiple things that could satisfy the need. Each statement could also be assigned different weights when searching that could be used to find the best ones between them. The Search class was created to handle the weight and results of the search; a vector in KnowledgeBase was created containing several Search objects.

### **3.2.1 Knowledge Base class**

Though there were many functions that this class implemented, only a few of them need to be pointed out because they are not very intuitive. There are three basic search functions. exactSearchKB is used to find a statement exactly in the knowledge base (this is only used when adding new statements to prevent multiple copies and also to figure out what other agents can be taught). fuzzySearchKB is used to check the constraints of a situation to find appropriate matches. It will find statements that have at least one matching constraint (and return a percentage) and deal with the current need. At last, it will populate the search results with the possible actions to do. search is used to find the value of an object; it finds all statements that are applicable for a current need or action combination. From there the search results can tell whatever other code that is querying the system whatever it needs to from those statements only.

scanKB is a function that can be used to setup the current situation. It will take in an object of what the agent can see, an object of what the agent has, and the current need. This list is given disregarding the fact that some items and agents, the knowledge base may not be familiar yet. So we write the statement such that those items will come up as "object" or those agents will come up as "other". This will force the templated versions to take more precedence. It will then pass these constraints onto the knowledge base itself.

The learning function will take an object of the status of all the agents needs before an action and the status of all its needs after an action and whatever the action performed was. The first thing it will try to do is figure out whether the action was good for it or not on its primary need. Once it figures that out, it will then infer several statements that it can add to the knowledge base (based on what the action was). If the action was bad, we put in the knowledge base to do the action under those circumstances, otherwise we tell the agent not to do that action (using the " " symbol). Finally, it adds any other statements about other needs that it can infer from the change

in the status of its needs.

### **3.2.2 Knowledge Statement class**

This class, as explained before, is to store individual statements in the database. It is set up to do some basic searching for the value of a statement or against its general constraints. The class is setup to hold a vector that breaks up the String that represents the statement to allow an easier search. It also has an importance and taste variable. The importance variable holds how much the action affected the agent after doing it (how did it affect all of its needs). The taste variable, on the other hand, has more to do with the actual items used. So that the best item has the best possible taste, even though it may change over time (if he finds a better item for instance). These two variables help to define the value of information for each statement and allow the agents to determine is most important to take and if teaching, which action is most important to teach.

### **3.2.3 Search class**

An object of this class is created when searching the knowledge base. There are 5 parts to this class of objects. The first is the percent that is how close the conditions in the current environment match this selected statement in the knowledge base. The index is used to find what knowledge base statement was this referring to. The intention is a string that tells what action and object pair the agent should try according to this statement. The importance and taste factors have already been explained in the Knowledge Statement class subsection. A vector is filled with several of these objects and sorted first by percent, then by taste, then by importance, for the agent to try these actions.

## **3.3 Why learning from other agents works**

This approach worked because we have agents that will walk around the world and experience different things while satisfying their needs. After a while they start to know exactly what items work the best for different actions and will automatically do that when they have such a need again. When a new agent comes along and has not had as much experience as this agent, then they can learn this and start to understand more about the world a lot quicker. Another example is when you have two agents that meet each other, each having totally different experiences, they can trade information and almost specialize so that the community as a whole is better off.

The idea is to allow the community as a whole to improve so that each of the individual agents will be able to gain as much knowledge as they can as quickly as possible. This allows each of the agents a better chance at surviving (preventing health from reaching 0) and thriving (keeping need levels as high as possible which means that the needs are all satisfied). If you do that then there will be a greater probability that that generation of agents will survive and be able to create a new generation of agents that can then have a chance to survive.

## 4 Experimentation and Observations

No real experimentation was done on the system. There were, however, noticeable improvements that were observed in the simulation as the project started to wrap up. There was a drastic difference in how long agents would survive before we allowed them to teach each other. The average life expectancy was approximately 30 years in the simulator with many agents dying at age 15 or below (thus not ever being able to reproduce) before we got the teach or teachme action working. After we got it working, agents would live to be about 80 years old and very few would die earlier than about 60.

Another observation was that the more they learned the more natural their exchanges would be. For instance, agents would not be able to tell the difference between what they are trading with another agent. Even worse, they would not understand the real value of an object if they new of it already. They would just trade based on their initial experience with it such that if a better object was dirty and they lost cleanliness when they picked it up, they would trade for a worse object because it might be better for one action but not anything else.

Some suggested experiments for this system would be the following. We have agents in the world that are more or less willing to share information than others and see how well information spreads. We can see how the agents find an equilibrium point with an environment by looking at the growth of the agents versus the growth of items. We could look at the difference between the bad items that are never taken and the good items that are never taken and watch the balance between them change as a community learns more. Finally, we could put two individuals in the environment, one that learns and the other that tries to learn but does not; we could see which actually stands a better chance at survival and by how much.

## 5 Conclusion

In our project, the aim was to show how much better a community that shares information about the environment is able to adapt to it than one that was not. The goal is to show that this is an important way that agents can learn about foreign partially observable environments. Given that an agent can do some observation by itself and, in addition, learn more from other agents around it, it stands a much better chance of learning about its environment in detail and doing so much more quickly.

There have been several revisions to the project that have been discussed. The first is to add a system by which the agents understand position and keep track of an internal map of the environment; this would allow us to less randomly create environments and put some objects always in certain areas so that the agents could learn to go there if it needs that object. Another possibility would be to allow for more complex actions with objects, ones that may even change the objects or the environment itself. This would allow for a more general learning that would increase the agents ability to satisfy its needs. One other idea is to give the agents personality that will influence how they behave in the environment.