# The Agents in an Agent-based Economic Simulation Model

Brian M. Slator and Golam Farooque
Computer Science Department
North Dakota State University
Fargo, ND 58105

## Abstract

The NDSU Retailing game is an interactive multi-media, multi-player, educational, and simulation-based economic game implemented in LambdaMoo, which is an object-oriented multi-user Domain (MUD). The primary simulation methodology of the Retailing game is agent-based modeling. Agent-based modeling involves specifying how individual agents (such as people, nations, or organizations) interact with each other and with their environment. Computer simulation is then used to discover the emergent properties of the model, and thereby gain insights into dynamic processes. This Agent-based approach is capable of revealing consequences through simulation that cannot be deduced with standard mathematical techniques [1]. Like induction, the main method of finding these consequences (and perhaps new insights) is through analysis of a set of data - in this case data generated by running the computer simulation. The goal, in these cases, is to discover new principles about the dynamics of complex systems, especially complex adaptive systems, which are typical of social processes. The over-arching goal of the Retailing Game is to construct a plausible economic simulation in order to create an authentic learn-by-doing environment for microeconomics education.

**Key Words**: Agent-Based Systems, Computers in Education, Intelligent Systems, Multimedia Applications

## Introduction

Economic models are typically built on the idea of demand and supply functions where economic entities maximize their welfare when the market achieves equilibrium. These models, however, often either ignore how these entities evaluate information, form expectations, evolve strategies, and execute their plans: and ignore the role of learning in decision making. Models in neoclassical economics are traditionally based on aggregation of behavior or use a representative entity. The mainstream Walrasian tradition usually focuses on entities that are perfectly rational and maximize expected utility.

Agent-based economics models, on the other hand, promises to model rational agents as heterogeneous individuals with divergent theories. They allow us to relax assumptions about perfect rationality, rational expectations, and perceptual maximization of expected utility. To date agent-based models have been used in the social sciences to explore patterns of spatial segregation, the evolution of cooperation, the emergence of money, cultural evolution, market processes, electoral politics, state formation, and group stability.

In this paper, we describe the NDSU Retailing game in contrast with other agent-based models, concentrating on the role of the Customer agents in implementing the simulation. The goal of the Retailing Game is to teach a wide set of skills associated with running a retail business by allowing the microeconomics' student to run a simulated store in a simulated economy [2]. The NDSU Retailing game is a multi-user, educational, economic game implemented in LambdaMoo, which is an object-oriented Multi-user Domain (MUD).

The objective of this paper is to examine the NDSU Retailing game, implemented in the mythical synthetic environment called Dollar Bay, in the context of an agent-based economic simulation model. There are three types of software agents in the Retailing Game

1) atmosphere agents, lending local color, and a measure of authenticity, to the environment. These agents are largely designed for their entertainment value. In Dollar Bay these include a Fire Inspector, a Juggler, a Beat Cop, and so forth
2) infrastructure agents, contributing in a meaningful way to the "play" of the game. These agents are essential to the pedagogical goals of the educational environment. In Dollar Bay these include the Customers that effect economic demand, and, to a lesser extent, the Employees that control the day-to-day workings of each synthetic retail establishment, and the agents who supply wholesale goods, direct advertising services, and so forth.
3) intelligent tutoring agents that monitor, mentor, and remediate the human learners in the performance of their roles. The tutoring agents are being developed as sub-topic experts who have access to problem solving experiences, context sensitive help and advice, conceptual and procedural tutorials, stories of success and failure within their particular sub-topic, and conversational networks for learner interaction.

Although the literature on agent-based models has been increasing very rapidly, there are very limited numbers of studies on agent-based economic simulation models [3,4,5]. [3] implements two case studies on agent-based economic models using Swarm. Swarm is a collection of software libraries, written in Objective-C. [4] develops an evolutionary trade network game (TNG) that combines evolutionary game play with endogenous partner selection. The TNG is implemented in C++. [5] builds an economic society of agents in which buyers and sellers compete with each other and try to increase their total values and total profits, respectively.

The agent-based economic models are different from one another in a variety of ways. To some extent, the Retailing game possesses some characteristics of other existing agent-based economic models, in particular the Swarm model. The players of the Retailing game are classes of consumers and firms. The Retailing game's players are allowed to compete with each other openly. They set a competitive price, hire the appropriate staff, and sell appealing products. The players advertise in a cost efficient way in order to reach the most customers. The two agent-based economics case studies implemented in Swarm also assumes that the economic agents are the firms, consumers, and labors. Swarm assumes an oligopolistic competition with differentiated products, the consumers have a preferred product, and each firm produces and sells only one product.

**Playing the Retailing Game**

The players in the Retailing game are assigned a location, and they must decide what to sell, how much to stock, who to buy from, and what price to set. The players also decide the level of services they offer. The decisions of all players are sent to a main server. The task of the server is to compute each Customer agent's shopping strategy based on all players decisions and the interests, wants, and location of consumers in Dollar Bay. The Retailing game provides the players the financial tools to calculate their inventory, assets, liability, expenditures, and profits. The players are also able to research their competitors' prices, inventories, and staffing decisions (but not profits or other presumably "secret" information). Further, each player has access to local newspapers, radio, and a wholesale directory to get general information about Dollar Bay's business environment, other competitors' advertisements, and to contact suppliers. By contrast, the economic models implemented in Swarm assume that the firm chooses to produce a product at a particular location, but may choose to change to a different location. Like the Retailing game, the economic agents in Swarm make self-interested decisions based on available information in the environment.

The Retailing game consists of interface objects including a map of Dollar Bay, representation of population and products, product class definitions, and product models. The Retailing game divides the Dollar Bay into neighborhoods and models the entire population of Dollar Bay into 20 distinct psychographic groups based on age, income, life narrative, interests, values, and lifestyles. Unlike the Retailing game, the Swarm economic models can be implemented at any place, and there is no clustering of customers. Swarm's constructed classes are: ConsumerClass, FirmsClass, OfferSpace (which represents all offer and sales data in the product space), MarketSpace (which records all actual transactions between buyers and sellers), and Creator (control exit and entry). Player creation of such classes are absent in the Retailing game.

The Retailing game uses product classes and models as its level of representation. A model represents a particular good for sale, and a product class is used to describe the market for an entire class of goods. All models in the same product class compete with each other while no models in different product classes compete. In Swarm, some product classes compete with each other.

In the Retailing game, a product class contains all information on each consumer's attraction for a particular product. The representation of any particular product is composed of average potential demand (APD), a percentage number for each cluster group's relative level of interest compared to the overall all potential demand (PD), a dollar amount for unit search cost (USC) for each cluster group, a unit transportation cost for each cluster group, unit service benefit (USB) and unit quality benefit (UQB) the cluster group receives from the store and/or product, all possible features of the product, price sensitivity (PS) of cluster groups, and a maximum dollar amount (MAX) and a minimum dollar amount (MIN). Based on the information stored in the product class representation, the Customer agents decide the amount of a particular product, at what price, from which stores they would buy. The individual purchasing decisions of these agents implement the economic activity of Dollar Bay.

The city of Dollar Bay is simulated by building a graphical user-interface onto a MOO (MUD, Object-Oriented, where MUD stands for Multi-User Domain or Dungeon). The basic components are "rooms" with "exits", "containers" and "agents" or "players". MUDs support the object management and inter-agent messaging that is required for multi-agent games and at the same time provides a programming language for writing simulation and customizing the MUD.

**Infrastructure Agents in the Retailing Game**

In this approach to simulating an economic environment, it is important to represent continuous demand for products. This is simulated by implementing the notion of a "virtual week" where customer agents are given a shopping list representing a week's worth of demand. Therefore, a "week" is defined as the amount of time it takes for all customer agents to exhaust their shopping lists. At the end of each week, new shopping lists are created and, more important to the players, new "attractiveness" lists are compiled. Then the customer agents are reset, and a new virtual week is begun. As a result, to understand the economic simulation, it is important to understand what happens in the "weekend calculation" which occurs at the end of each virtual week.

**Algorithm: The "Weekend" Calculation**

The purpose of the weekend calculation is to charge players for their weekly expenses and recalculate the customer agent "motivations". This calculation, which takes only a few minutes to execute, is composed of the following steps:

1) A batchmode switch is set, which effectively shuts down the many various software agents who run during the course of a virtual week; the turn counter is updated; and the season variable is updated, as necessary. If it is the first "week" of a new season (or new product types have been added to the simulation), then the Customer agents re-initialize their maximum product type demand values

   **Algorithm: Maximum Product Type Demand**

   Maximum demand for each product type (there are currently 46 product types defined) is calculated for each customer agent (psychographic representative) on a yearly basis. This is calculation is based on
   a) cluster group population in a neighborhood
   b) cluster group "interest" in a product (APD)
   c) neighborhood and product features that affect demand, and

d) a "fixed" demand multiplier intended to reflect average national demand for the product type (a figure taken from national sales revenue data).

The figure calculated for each product type (i.e. hardware for pets) indicates how many units of each type will be purchased by the cluster group in one year. This figure is divided by 52 to return a weekly demand figure.

The weekly demand figure on each customer agent is later copied and used as their "shopping list". Items are removed from the list as they are purchased (or abandoned) by the customer agent.

2) The Discount Warehouse does its weekly activities: ordering and advertising (which varies by virtual "season").

3) Weekly accounting and debiting is processed for player's Leases, Staffing, Liabilities (i.e. loans), Advertising, and Purchases.

**Algorithm: Weekly accounting and debiting**

a) Lease charges are assessed on each store for a week's rent. Leases are automatically and semi-randomly arranged when players first enter the game. Player's are only allowed a single location. If the store has insufficient funds to pay rent, an eviction function is invoked, and the player goes into virtual receivership.

b) Staffing charges are assessed on each store for a week's wages. Staff is acquired with a hiring interface. If there are insufficient funds, the store's staffing is reduced by one level. However, store staffing can never be reduced to zero, as this would prevent all future sales, which depend on employee agents.

c) Liabilities charges are assessed on each store for the weekly loan repayment amount. Loans are arranged through interactions with the banker agent.

d) Advertising charges are assessed on each store. Advertising is purchased on a continuing basis with the advertising interface. If a player cannot afford to pay, their continuing ad is deleted. Note: it is also possible to order "leaflet style" advertising through visiting the Quicky Ad agent, but these are one-time advertising expenses that are immediately billed (and cannot be purchased without sufficient funds).

e) Weekly product ordering is calculated. If the player has insufficient funds, the player gets no new merchandise.

4) Customer agent motivations are determined for the coming week based on both fixed and dynamic factors.

**Algorithm: Customer Agent Motivations**

a) Fixed attractiveness factors are calculated for each company: public image (a value calculated on successful versus unsuccessful customer agent "visits" to the store), and longevity benefit (0.1 "points" added for every week in business, up to a maximum of 20 points)

b) The "attractiveness" of each store, to each customer agent (representing a psychographic group), is calculated for every neighborhood. This is another complex formula involving longevity, public image, advertising "benefit", and travel cost.

The formula for calculating a store's attractiveness is

attractiveness = SV + abs(SV) * longevity / 100;

where SV = ad benefit - transportation costs + public image

ad benefit = the combined effect of each ad's media, tone, size and feature effects

transportation cost = 2 * distance * travel constant

The result is a list of {store, score} pairs, sorted and stored on each customer agent. This list is used by customer agents to choose which stores they will visit first.

5) The customer agents are reset and their "maximum demands" (which are periodically recalculated using the function described above) are copied over into their shopping list

6) The batchmode switch is cleared, indicating the weekend calculations are at an end.

**Process Control in the Economic Simulation**

It is interesting to note that process flow in the Retailing Game's economic simulation is strictly agent-based. There is no infinite loop anywhere in the simulation, rather, there is an infinite sequence of forking processes.

In LambdaMOO, as in many operating environments, there is a control structure named 'fork()' which has the effect of spawning a new process and returning control to the calling environment. Thus, a forking routine will launch a process that runs in parallel with itself. In the Retailing Game, two major events occur: 1) one of the last steps in the "weekend" calculations is to fork() the dispatching sequence of the customer agents, and 2) one of the final steps of the last customer agent (there are currently 49 customer agents in the simulation) is to fork() a call that invokes the weekend calculations. In this way, the fork() operation provides a straightforward way to maintain a continuous simulation.

**Customer Agent Behavior**

The customer agents in the economic simulation are what drive the action. The simulation randomly chooses customer agents, and the customer agents randomly choose one item from their shopping lists. This product becomes the object of their search. The customer agents first scan their list of "attractive" stores in order, looking for one that sells products of the type needed. Then the customer agents visit these candidate stores in turn, attempting to buy the item.

It is interesting to note that customer agents visit stores without knowing in advance that the store actually carries the product they seek. This is because stores advertise and are cataloged according to their product lines, e.g.. hardware or pets, but customer agents are searching for specific products, e.g.. rock saws or fish. When a customer agent visits a store, they know they want a product, but not specifically which model, and they know the player carries that product line in their store, but not which particular products, and without advance knowledge of whether the store has the items on hand or is sold out.

This approach has two obvious benefits. First, the simulation is plausible in that customer agents must search for products, operate without perfect knowledge, and can quite easily go home empty handed. Second, players are permitted to improvise and adapt in that if a succession of customer agents visits in search of fish, and the player sells no fish, they can quickly order some to take advantage of the perceived demand. This footwork on the part of the player is imperceptible to the customer agents who are merely searching for products, so when a player suddenly orders fish, the customer agents are perfectly willing to buy them, even though the player only acquired them moments before.

**Algorithm: Shopping**

The customer agents are dispatched randomly.

1) The dispatch function simply calls the shop function, except for about 1% of the time, when it randomly calls a return product function
2) The shop function chooses the next item to shop for, by randomly selecting a product from the shopping list then
3) The customer agent chooses the "best" store to shop in, by cycling down the sorted "attractiveness" list of stores searching for the next one that has the chosen product type in it's catalog, then
4) The customer agent physically moves to the store, and attempts to shop there
5) The shopping attempt function implements an agent-to-agent interaction between the customer agent and the store's employee. This interaction can culminate in one of several possible outcomes
   a) The customer agent first checks if there is an employee on duty. If not, the customer agent consults its own "larceny index" and decides whether to simply steal something from the store. This only happens rarely.
   b) Assuming shoplifting is rejected, the customer agent will simply leave the store if there is no employee present
   c) If an employee is present, the customer agent will first check if the employee is busy. If busy, the customer agent will again briefly consider larceny, but rarely will attempt it
      i) If the customer agent attempts shoplifting while the employee is present, there is a chance the employee will detect the theft. If so, the customer agent might be ejected (depending on individual employee characteristics) and a report might be made to the police (the Beat Cop, mentioned above)
      ii) In the usual case, the customer agent will wait for a busy employee from some period, depending on factors that include the customer agent's errand factor and impatience index.
      iii) If the employee is not busy, or the customer agent's patience is rewarded, the customer agent will request a product from the employee. From this point on, the customer agent/employee interaction is played out
   d) When the customer agent requests a product, the employee agent scans the store's inventory, looking for specific models of that product type. The employee will then typically present the models, in descending order by price, for the customer agent to consider.
   e) The customer agents consider purchases as follows:
      i) First, the customer agent calculates whether they can afford the proffered model, by referencing its own psychographic group as defined on the object for that product line. On this basis the customer agent determines whether the price of the model is too expensive, or in its range.
      ii) Then the customer agent calculates whether to buy the model or not. This decision is calculated as a linearly increasing likelihood, also based on the minimum and maximum affordability values. The probability of the customer agent purchasing the model is calculated by
      prob = 100 * (max - price) / (max - min)
         however, the decision to buy is still randomized by
         yes if random(100) < prob, otherwise no, which simply says that the closer you are to the minimum affordability, the more likely you are to buy because minimum affordability is the price at which everyone (in the particular psychographic group) can afford the purchase.
      iii) Finally, however the customer agent/employee interaction proceeds, there is a random chance the customer agent will attempt to play the store's fortune machine, if one exists.
   f) If the item is successfully purchased, the customer agent removes the item from its shopping list and returns home. If the item is not found in the first store, the customer agent makes several attempts then abandons the search, removing the item from its shopping list and returning home.
   g) As described above, when the last customer agent has exhausted their shopping list, the week is ended, which invokes the "weekend" calculation and the process starts all over again.

**Conclusion**

This paper presents an agent-based economic simulation implemented in LambdaMoo. The object-oriented simulation method offers many possibilities for extending and manipulating the existing economic model. The Agent-based approach promises to model rational agents as heterogeneous individuals with divergent theories. They allow us to relax assumptions about perfect rationality, rational expectations, and perceptual maximization of expected utility.

**References**

[1] Axelrod, Robert (Winter 1996), "Education and Training"- a course offered at the University of Michigan.
[2] Hooker, B. and B. Slator, B (1996) A Model of Consumer Decision Making for a Mud-based Game. ITS'96 Workshop on Simulation-Based Learning Technology. Montreal, June.
[3] Stefanson, Benedikt (1997), Swarm: An Object Oriented Simulation Platform: Applied to Markets and Organizations in Angeline, Reynolds, McDonnell, Eberhart, (eds), Evolutionary Programming VI, Lecture Notes in Computer Science, Vol. 1213, Springer-Verlag, New York.
[4] Tesfatsion, Leigh (1996), A Trade Network Game With Endogenous Partner Selection. Proceedings of the Fifth Annual Conference on Evolutionary Programming, June.

[5] Vidal, James M. and Edmund H. Durfee, (1996), "Building Agent Models in Economics Societies of Agents." http: //ai.eecs.umich.edu/people/jmvidal/papers/amw/amw.html