Computer Lab V: Trading Agents

Bastiaan Daems & Matthijs Dorst

Contents

Introduction

The goal of this exercise is the implementation of a small agent communication language. To this end a simple economic landscape is created with traders, retailers and producers. Traders move around between retailers and producers, buying cheap stock and trying to sell for more.

Instead of directly accessing each other's members the agents are forced to use a simple language containing several different message types to propose sales, ask for prices, reject offers etc.

For a full elaboration of the implementation please refer to the attached documentation file.

Assignment 2: Model discussion

Code refactoring

In the original model all classes were listed under the same, default package. Thus, our starting point was to refractor that into several packages each designed with a specific purpose in mind.

Package: agent

The agent package contains the (now abstract) Agent base class. All agents implement this class which provides generic functionality such as moving, keeping track of a location, listing other agents in range and various set- and get-methods.

New in this class is a random movement method which can be used by stuck agents. Whenever an agent fails to succeed in moving to this preferred location, a random movement is executed to prevent gridlock's from occurring.

The Producer class extends this Agent class. New in the Producer is the ability to handle a sale (which tells a buyer that the sale is agreed upon) and an extended message handling method to allow for both price requests as well as price agreement messages.

The Retailer class is augmented with an ability to decide on a purchase. Whenever an agent proposes to sell his stock to the retailer, the Retailer compares the proposed price with its own sell index for that product and either replies with an acceptance or rejection.

In addition the Retailer now has the ability to negotiate a sale by asking the agent how much he charges for his stock.

The Trader class is implemented with an ability to propose a sale (to a retailer), complete a sale to a retailer, an ability to decide whether or not to purchase a certain product (for the price set by a producer), handle the failure of a sale (try again till either the price drops too low or the retailer accepts) and lastly complete a purchase from a producer.

Package: communication

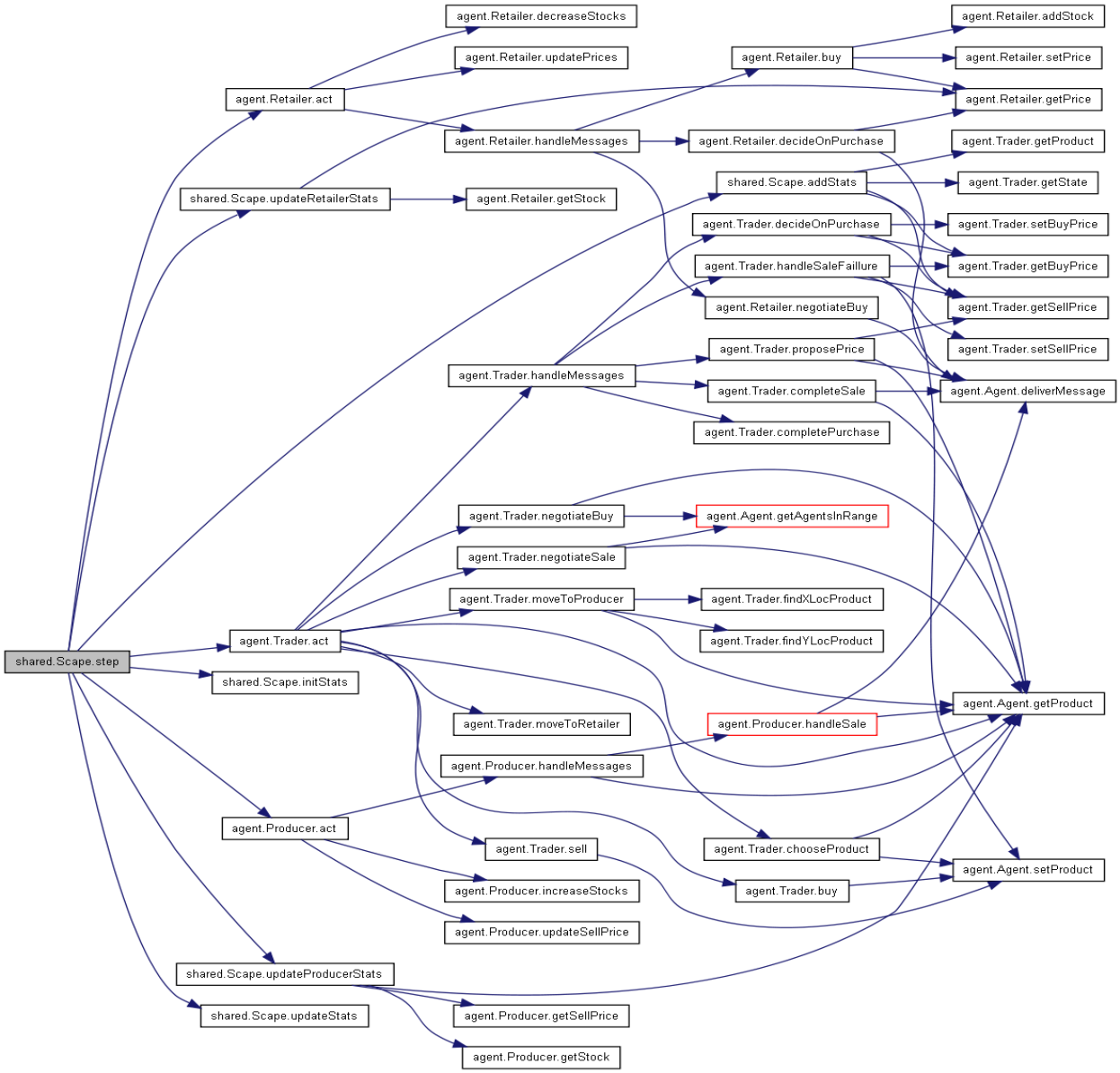
The communication package handles communication between agents. It consists of both a Message and a Protocol class. The Message class is used for all messages send between agents. Each message

contains a certain (content) type, as defined in Protocol. This lets agents know what the purpose of each message is.

Additionally a Message contains a reference to its original sender (useful if you want to reply to the original guy!) and optionally a "what" and "number" – mainly used to denote the type of product and the price for that product.

Package: shared

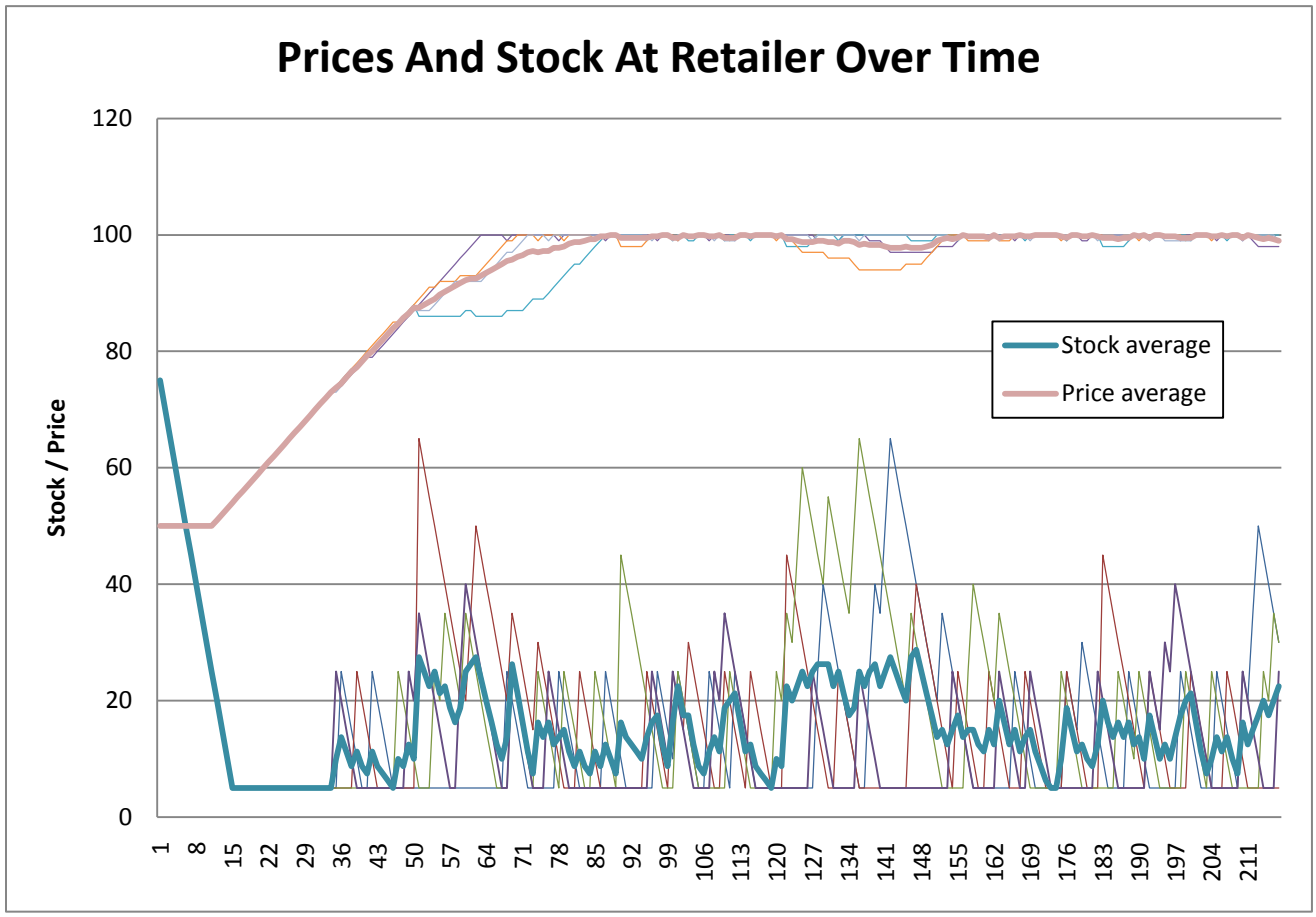
While little modified, this shared package contains the classes responsible for setting up our environment (Scape, Site) and the GUI (ButtonPanel, MainPanel). As such, they do not belong to the other packages but are a shared resource instead. Interesting to note is the call graph for Scape: since this acts as a overall landscape handler, it illustrates the general information flow and interaction between agents quite graphically:



Assignment 3: Simulation Runs

Listing of price & stock at retailer

Consider the prices & amount in stock for all products over the course of several hundred iterations:



We see a steady increase in product prices at the retailer for the first 80 ~ 90 iterations, which accounts for a lack of new products being brought in (thus creating a greater demand). Once agents start delivering and the price reaches its zenith it stabilizes around 100 units.

Such stabilization is more difficult to find for stock: it varies greatly between 5 and 30 units. While a single agent can restock 20 units of one product, for a stable stock the number of trading units must be at least 1 each round. Since there are 40 agents and a round-trip from producer to retailer costs around 30 steps in the most optimal condition it comes as no surprise that this equilibrium state is barely maintained with the default number of agents present.

Assignment 4: Gridlock Prevention

While the ability to detect stales is useful, we argue that switching goals in such a situation is economically unfeasible. It requires both the agent to drop his stock, as well as for the agent to choose a different product to purchase without additional information about prices. In effect, this means the same agents could bump into each other, decide to dump their cargo, go back, refill at their original producer and on their way to a retailer bump into each other again. While extreme, such a scenario means agents would also need a way to randomly decrease the likelihood they will buy from their last producer again, if going to that producer results in gridlock.

A far more effective and economically feasible solution therefore is suggested: instead of 'dumping cargo' and switching goals, the agents simply take a step aside when they find their way blocked by another agent. This step is in essence random and will take the agent anywhere he can go, provided the target location is not blocked either.

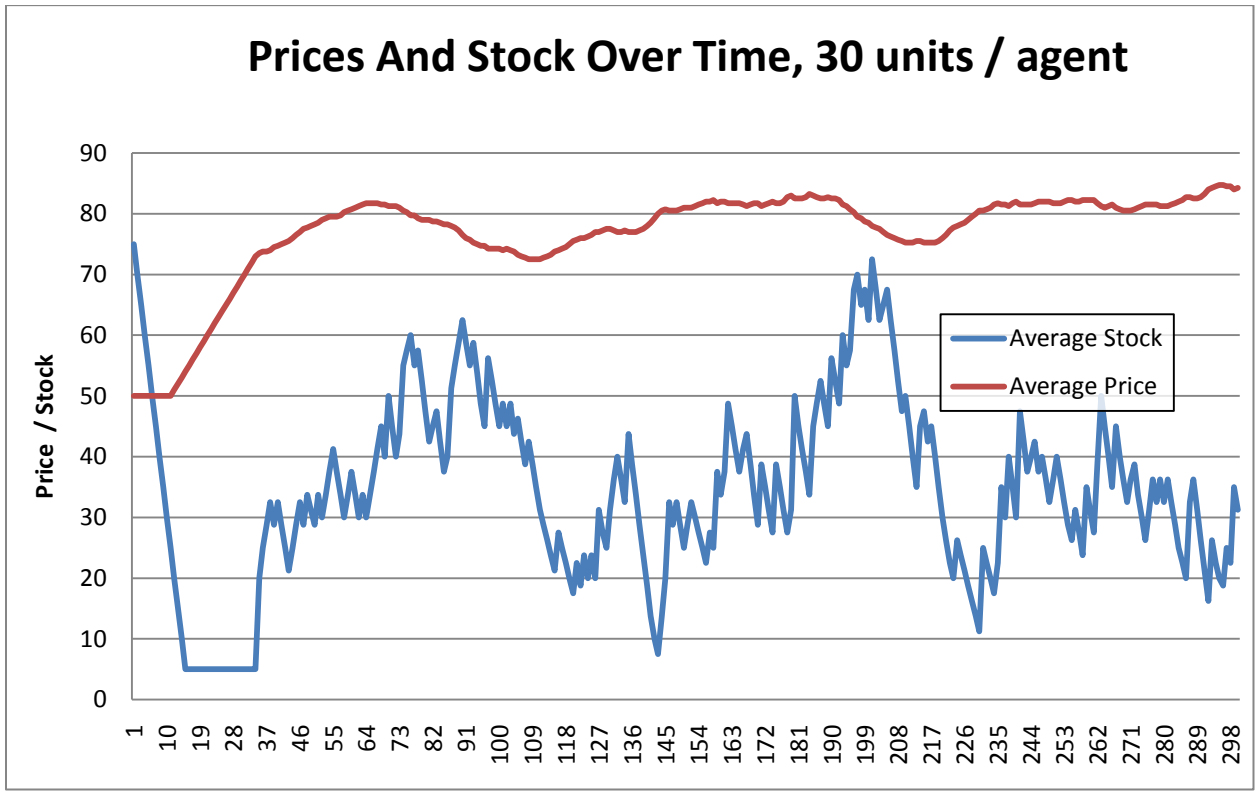
Furthermore this solution does not require the agent to dump cargo, nor to remember where he has been, nor to adjust his prices and settings to prevent making the same mistake again. It is elegant, simplistic, and extremely effective – gridlocks were reduced by 100% after implementation.

Assignment 5: Sale Variables

Increased Agent Capacity

As we saw in exercise 3, the default number of agents was barely able to keep the retailer supplied. To this end we propose to increase the quantity bought and sold by each trader from 20 to 30. This should prevent the retailer from running out of stock and as a consequence from maximizing his sale prices (no more 'infinite' demand).

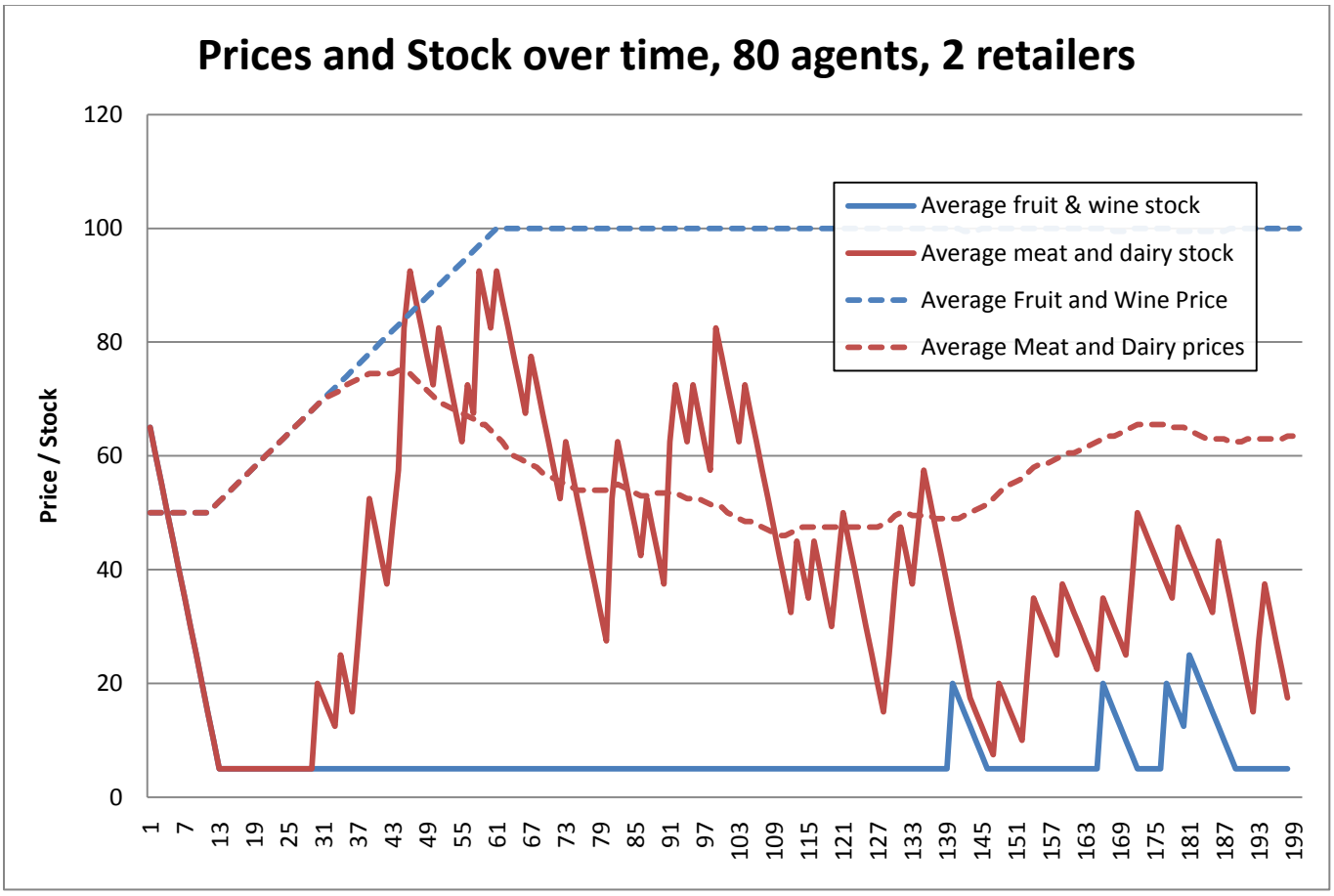
For reasons of clarity, only average prices and stock is listed:



As we can see, the average price at the retailer does not max out any longer, nor does he run out of stock easily. Interesting to see as well is the slight drop in price right after stock surges around iterations ~100 and ~190. While an optimal situation would consist of a roughly half stocked store (so there is still room for more produce, but he won't run out quick either) and we see no indication of this yet, it is a step in the right direction and at least our economic landscape is now dynamic and somewhat realistic.

Multiple Retailers

Interesting is also the situation where we add a second retailer and instruct agents to move to the retailer closest to them. Retailers were positioned in the middle of the map height-wise, with one retailer at 25% of the map width and another at 75% map-width. We did this and let the simulation run for 200 iterations, obtaining the following result for the right-most retailer:



Consider this the situation where a shop is positioned further away from a resource than another shop (as for example is the case in isolated countries or islands): sometimes an agent will find it worth the effort to go the long way around, but in general the shop will remain under stocked on those specific products, driving up prices. We can clearly see a few restocking's, but overall only meat and dairy is sufficiently supplied to meet demands. In fact, with the same average amount of agents per retailer there is considerable more stock for the 'close by' products than for those same products in a single-retailer situation.

Note that we did not implement a price estimate per retailer for agents – their price guess is (incorrectly) considered generic, which we would expect is more or less true for a real world situation but of course in this simulation holds no grounds.