# Knowledge & Agent Systems
# Assignment 4
Deadline: October 6th 22:00

## Report requirements

- Hand in your updated *kas_assignment4.nlogo* code, and a PDF report with your answers to the questions below.
- Include your student numbers and names in the code and the report itself.
- If you have to write code, also add the small snippets of code to the report. You can use the *listings* package for this. In addition, explain the design choices of your code and how it solves the associated problem.

## Introduction to the assignment

The goal of this exercise is the implementation of a small agent communication language. The setting is a simple economic environment, in which prices are dynamically established by negotiated buying and selling. The starting point is a world of patches [$31 \times 31$]. The center patch is the locus of the retailer agent that wants to buy four kinds of products: dairy, fruit, meat, and wine. The corners of the environment are the locations of production, each represented by an agent. The rest of the world is inhabited by a fixed number of trading agents.

**Producers** Production representatives sell their product to the traders. They have a price for their product and produce a certain amount (variable `producersProduction`) per tick. Their stock is maximally 100 units. Prices depend on stocks and sales. When their stock drops below 25 (variable `lowerSL`), they increase their price by 0.1 price unit. When it rises above 75 (variable `upperSL`), they decrease their price by 0.1 price unit. In this way, their price strategy aims at a stock between 25 and 75. The idea is that a high stock level is a sign that the current price level is too high for the market, and that a low stock level is a sign of a price that is too low. In the simulation, the producers are represented by colored patches in the corners. Their colour depicts the product they are selling.

**Retailer** The retailer buys products from the traders, hence also has a price for each type of product. Stocks for each product drop by a certain amount (variable `stockDecreaseRetailer`) per tick, as a representation of the retailer's selling to customers (not modelled further in the simulation). When his stocks are above 75 (variable `upperSL`), he reduces his price by 0.1 price unit. When his stocks drop below 25 (variable `lowerSL`), he increases his price by 0.1 price unit. In the simulation, the retailer is the black patch in the center.

**Traders** The traders move around in the world according to their current plan. Traders buy products at the producer locations in fixed quantities (variable `saleQuantity`) and bring them to the retailer in order to try to sell them at a profit. Traders do not have access to the actual prices of the producers and retailer. Instead, each trader estimates the buying and selling prices based on its previous interactions.

The traders have seven internal states:

- `CHOOSE_PRODUCT`

  The trader decides which product to buy, based on the maximal difference between his expected buy and sell price.
- `MOVE_TO_PRODUCER`

  The trader moves towards the producer.
- `NEGOTIATE_BUY`

  The trader's negotiation with a producer must be handled.
- `BUY_FROM_PRODUCER`

  The trader buys a product from a producer.
- `MOVE_TO_RETAILER`

  The trader moves towards the retailer location.
- `NEGOTIATE_SALE`

  The trader's negotiation with the retailer must be handled.
- `SELL_TO_RETAILER`

  The trader sells a product to the retailer.

Trading agents have the colour of the product they are trading in. The trader can interact with a producer or retailer once they are close enough (variable `interactionRange`).

**Trader-Producer exchange** If a trading agent meets a producer, the trader should initialize a negotiation by sending a message in which he asks for the producer's price. When the trader's current buying price estimate is equal or higher (i.e., when he has overestimated the buying price) and the offered price is below their selling price estimate (i.e., when he thinks he can make money), the trader accepts. Note that a transaction also fails if a producer does not have enough stock.
In case of a successful transaction, the trader uses the agreed price as the new buying price estimate. When the exchange is not successful, the trader increases its buying price estimate for the product by 1 price unit and chooses a new buying goal.

**Trader-Retailer exchange** When a trader arrives at the retailer's location, the trader initializes another negotiation. The trader accepts the retailer's price, if it is equal or above its estimated selling price.
If the transaction is successful, the trader sets its selling price estimate to the agreed price. When a trader cannot sell the product he is carrying to the retailer, he discards it and reduces his selling price estimate for the product by 1.

What the agents do not yet have, is a way of communicating which each other, and hence, no way of negotiating sales. If you run the simulation as provided, each trader will choose a buy goal, move to the appropriate producer, and then stop. The idea is to have agents send messages to each other to exchange information about their identity, products, buy and sell prices and so forth. For this purpose, every agent has a `messageQueue`, which all other agents can 'deliver' messages to (procedure `sendMessage`). Every agent's act cycle starts by dealing with its received messages (procedure `handleMessages`). These procedures have not yet been implemented.

## Question 1

Complete the code that allows for communication. You will need to implement the communication using messages, as per the introduction. Incoming messages should trigger appropriate changes in the traders' states. All information exchange should occur through messages.

**Note:** it is normal that the system is not stable (i.e., prices plummeting/skyrocketing indefinitely) after implementing the code. Making it stable is asked for in Question 6.
**Tip:** Consider the following code snippet to see how you can use the procedures `createMessage` and `sendMessage` while iterating over the messages inside the `messageQueue` (that is, when handling messages).

```
foreach messageQueue [ [message] ->
  let messageContent table:get message "content"
  let messageSenderID table:get message "senderID"
  let messageProduct table:get message "product"
  let messageNumber table:get message "number"

  if messageContent = "foo" [
    let messageToSend createMessage "bar" who messageProduct 42
    ; reply to the turtle who sent a message
    sendMessage (turtle messageSenderID) messageToSend
  ]
]
; Do not forget to empty the message queue!
set messageQueue []
```

The procedure `createMessage` expects four parameters. The `content` parameter should represent the 'subject line' of the message, which helps the recipient know what is being discussed. The `senderID` parameter should represent the ID of the sender, which helps the recipient know whom to reply to. When a turtle sends a message, the (built-in) `who` turtle variable can be used to retrieve their own ID (see `https://ccl.northwestern.edu/netlogo/docs/dictionary.html#who`). The `prd` parameter should represent the product that is being discussed. The `number` parameter should be used to discuss the price.
Finally, note that the `receiver` parameter inside `sendMessage` should be a turtle, so we use the `turtle` procedure to retrieve the turtle associated with the `messageSenderID` (see `https://ccl.northwestern.edu/netlogo/docs/dictionary.html#turtle`).

## Question 2

Describe and motivate the design decisions in your implementation. This should include a description of all implemented procedures.

## Question 3

Run the simulation with `producersProduction` set to 3.0, `stockDecreaseRetailer` set to 1.0, `numberTraders` set to 30, and `saleQuantity` set to 15. What happens to prices and stocks? Describe the simulation's overall behaviour.

## Question 4

Currently, a trader is choosing the product for which they estimate the highest profit. Instead, implement a probabilistic selection of producer, such that agents select what producer to visit based on the expected profit probabilistically rather than a take-the-best approach. Note that every product should have a non-zero probability of being chosen, even if the expected profit is negative. Compare the simulation's overall behaviour with the results from Question 3. What are the most notable changes?
**Tip:** the `rnd` extension might be helpful (see `https://ccl.northwestern.edu/netlogo/docs/rnd.html`)

## Question 5

Keep the `chooseTheBest` switch set to `off`. Experiment with the `producersProduction`, `stockDecreaseRetailer`, `numberTraders`, and `saleQuantity` parameters. For each parameter individually, describe and explain their effect on the producers' and the retailer's prices.

## Question 6

Adjust the variables `producersProduction`, `stockDecreaseRetailer`, `numberTraders` and `saleQuantity` so that a stable, dynamic environment is established. Stable means that prices should not plummet or skyrocket indefinitely. You are free to adjust the agent mechanisms themselves if necessary. In that case, describe the changes in your report, and create a switch that can enable/disable your new mechanism. Innovation and experimentation will be rewarded.