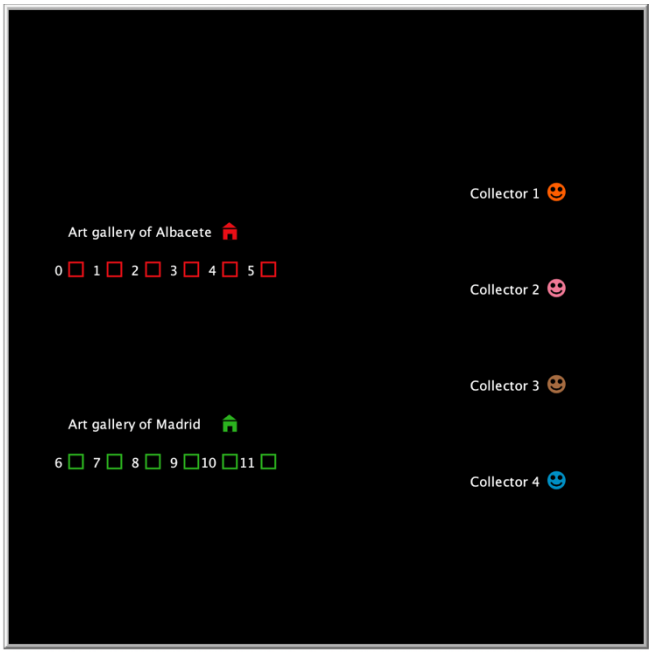
Market in NetLogo

Art galleries, paintings, and collectors



Jordi Bonet Valiente
Clara Rivadulla Duró

Index

INTRODUCTION

In this final delivery, we've decided to model a MAS Market which consists mainly of the transaction between art galleries and collectors, where the specific products that are sold and bought are paintings. Communication is done through message passing (which we saw in the last practice).

Although its process has involved the definition of interaction protocols among agents and the design of the interface in NetLogo, the following document explains in detail how the project was coded, and which were the experiments we did to test it.

CODE ANALYSIS

We've declared the following global and agents' (turtles) variables:

```
globals [
  number-of-galleries
  ;;number-of-collectors
  number-of-paintings
  paintings
  authors
  paintings-sold-to-process
  buyers
]

turtles-own [
  ;; Communication between agents
  current-messages ;; List of current messages
  next-messages    ;; List of messages for the next iteration
  ;; Properties costumers (collectors) and providers (galleries)
  own-paintings
  own-authors
  number-own-paintings
  price-paintings
  name
  type-entity
  money
  initial-position
  ;; list preferences collectors
  preferences-attributes
  preferences-values
  preferences-paintings
  preference-price-max
  ;; advertisement
  advertisement-to-sent  ;; text ad
  boolean-ad-sent        ;; boolean to kwow is the ad was sent to all the collectors
  number-ad-receivers    ;; the current number of collector that receive the advertisement
  ads-received           ;; save in a collector's list the gallery's name that has already sent an ad to him
  ad-attributes          ;; list of attributes of the ad's content
  ad-values              ;; list of the attributes values
  ad-painting-title      ;; name of ad's painting
  ;; Properties paintings
  own-price
  sold
]
```

Like we can see, we've declared as global variables which ones that needs to be access to all agents (all the paintings, his authors, his buyers). The attribute

paintings-sold-to-process is a list for update the market's interface when a painting be sold.

By the other way, we've the turtles' attributes for galleries, collectors and paintings (the paintings aren't agents, only is a turtle to see them in the market interface).

In the setup function, we setup the paintings, the galleries and the collectors.

```
to setup
  clear-all
  reset-ticks
  setup-paintings
  setup-galleries
  setup-collectors
end
```

In the paintings' setup, we set all paintings attributes and load them in the market's interface. We set them considering that always have the same two art galleries.

```
to setup-paintings
  set number-of-paintings 12
  set paintings (list "The Tree of Life, Stoclet Frieze" "Jimson Weed" "Oriental Poppies" "Dream Caused
  set authors (list "Gustav Klimt" "Georgia O'Keeffe" "Salvador Dalí" "Frida Kahlo" "Hieronymus Bosch")
  set paintings-sold-to-process []
  set buyers []
  set-default-shape turtles "square 2"
  let x-cordinates 20
  let y-cordinates 3
  let i 0
  foreach paintings [
    create-turtles 1 [
      set label word i "      "
      if i <= 5 [
        set color red
      ]
      if i > 5 [
        set color green
      ]
      set name item i paintings
      set sold false
      set type-entity "Painting"
      set own-price 100
      setxy x-cordinates y-cordinates
      if i != 5 [
        set x-cordinates (x-cordinates + 2)
      ]
      if i = 5 [
        set x-cordinates 20
        set y-cordinates 26
      ]
      ;; We initialies the lists of received messages
      set next-messages []
    ]
    set i (i + 1)

  ]

end
```

In the following picture, we've the galleries' setup. Here we create two art galleries: the art gallery of Albacete and the art gallery of Madrid. We initialize their attributes and put them in market's screen.

```
to setup-galleries
  set number-of-galleries 2
  set-default-shape turtles "house"
  let x-cordinates 28
  let y-cordinates 5
  create-turtles 1 [
    set name "Art gallery of Albacete"
    set type-entity "Gallery"
    ;; setup paintings
    set number-own-paintings 6
    set own-paintings (list "The Tree of Life, Stoclet Frieze" "Jimson Weed" "Oriental Poppies" "Dream Caused by the Flight of a Bee Around a Pomegranate a Second Before Awakening" "The Elephants" "The Two Fridas")
    set price-paintings (list 100 70 73 68 60 89)
    set own-authors (list "Gustav Klimt" "Georgia O'Keeffe" "Georgia O'Keeffe" "Salvador Dalí" "Salvador Dalí" "Frida Kahlo")
    ;; view
    set color red
    set label word name "       "
    setxy x-cordinates y-cordinates
    set y-cordinates (y-cordinates - 10)
    ;; money
    set money 0
    ;; ads
    set advertisement-to-sent "'Jimson Weed' at 70 billion euros, we practically give it away."
    set boolean-ad-sent false
    set number-ad-receivers 0
    set ad-painting-title "Jimson Weed"
    set ad-attributes (list "price" "author" "title")
    set ad-values (list 70 "Georgia O'Keeffe" "Jimson Weed")
    ;; We initialies the lists of received messages
    set next-messages []
  ]
  create-turtles 1 [
    set name "Art gallery of Madrid"
    set type-entity "Gallery"
    ;; setup paintings
    set number-own-paintings 6
    set own-paintings (list "The kiss" "Adoration of the Magi" "The Garden of Earthly Delights" "Black Iris" "The Persistence of Memory" "Self-Portrait with Thorn Necklace and Hummingbird")
    set price-paintings (list 90 80 62 78 80 69)
    set own-authors (list "Gustav Klimt" "Hieronymus Bosch" "Hieronymus Bosch" "Georgia O'Keeffe" "Salvador Dalí" "Frida Kahlo")
    ;; view
    set color green
    set label word name "           "
    setxy x-cordinates y-cordinates
    set y-cordinates (y-cordinates + 5)
    ;; money
    set money 0
    ;; ads
    set advertisement-to-sent "'The kiss' at 60 billion euros, a true bargain!"
    set boolean-ad-sent false
    set number-ad-receivers 0
    set ad-painting-title "The kiss"
    set ad-attributes (list "price" "author" "title")
    set ad-values (list 60 "Gustav Klimt" "The kiss")
    ;; We initialies the lists of received messages
    set next-messages []
  ]
end
```

In the collectors' setup, we create N collectors and initialize their attributes. As we can see, the collector's preferences, which are his money and his preference author, are random.

```
to setup-collectors
  set number-of-collectors 4
  let cont 1
  set-default-shape turtles "face happy"
  let x-cordinates 45
  let y-cordinates 7
  create-turtles number-of-collectors [
    set name (word "Collector " cont )
    set type-entity "Collector"
    ;; setup paintings
    set number-own-paintings 0
    set own-paintings []
    ;; view
    set label word name "       "
    setxy x-cordinates y-cordinates
    set y-cordinates (y-cordinates - 5)
    set initial-position (list x-cordinates y-cordinates)
    set cont (cont + 1)
    set color one-of base-colors
    ;; money
    set money random 200
    if money < 100 [ set money (money + 100)]
    ;; set preferences
    set preferences-attributes (list "price" "author")
    let price-preference random 100
    if price-preference < 50 [ set price-preference (price-preference + 50)]
    let i random 5
    let author-preference item i authors
    set preferences-values (list price-preference author-preference "")
    print(word "Preferences " name ": " preferences-attributes preferences-values)
    set preference-price-max 80
    ;; We initialies the lists of received messages
    set next-messages []
    ;; for ads
    set ads-received []
  ]
end
```

The 'go' function is like the last delivery. First, we swap messages and process them. Then send advertisements and see if a painting was sold to update it (process-paintings-sold).

```
to go
  swap-messages            ;; We activate the messages sent in the previous iteration
  process-messages         ;; We process the messages
  send-ads                 ;; Galeries send ads to collectors
  process-paintings-sold   ;; Function to set as sold the paintings sold
  tick
end

to send-ads
  ask turtles [
    if type-entity = "Gallery" and boolean-ad-sent = false
    [
      ;; send message type ad
      let current-gallery self
      let current-gallery-name name
      ;; receiver = all agents with role "Collector"
      let possible-collector one-of turtles
      if possible-collector != nobody [
        ;; Send "advertisement" message to all the collectors
        if [ type-entity ] of possible-collector = "Collector"
        [
          let collector possible-collector
          ;; if this gallery hasn't already sent an ad to his collector
          if not member? current-gallery-name [ ads-received ] of collector[
            ;; print (word "Self: " self " Gallery: " current-gallery " Collector: " collector)
            ;; print (word [ ads-received ] of collector " Collector: " collector not member? name [ ads-received ] of collector)
            send-message current-gallery "AD" advertisement-to-sent collector [ ad-values ] of current-gallery
            ;; append gallery's name in collectors list
            ask collector [
              ;; print (word "ADD " ads-received " Collector: " collector)
              set ads-received lput current-gallery-name ads-received ;; insert-item 1 [ ads-received ] of possible-collector name
              ;; print (word "ADDED " ads-received " Collector: " collector)
            ]
            ;; increase the counter of ad's receivers
            set number-ad-receivers (number-ad-receivers + 1)
          ]

        ]
      ]
      ;; if all the collectors receive the ad
      if number-ad-receivers >= number-of-collectors [
        ;; set the advertisement as sent
        set boolean-ad-sent true
      ]
    ]
  ]

end

to swap-messages ;; all the next-messages become current-messages and we have the next-messages entry empty
  ask turtles [
    if type-entity = "Gallery" or type-entity = "Collector"
    [
      set current-messages next-messages
      set next-messages []
    ]
  ]
end
```

In 'send-ads', we send an advertisement to a collector, checking if the collector hasn't already received an ad of this gallery. When the gallery has sent each collector (when the number of ad receivers is equal to the number of collectors) the gallery doesn't send more advertisements.

The next functions are very similar to the functions of message passing in the last delivery. The difference is that we add a list of values to pass with the messages that are necessary to an operation when we process it. Only the agents, a gallery or a collector can swap messages because the paintings aren't real agents. The different kinds of messages are *AD*, *INFORM*, *REQUEST*, *RESPONSE*, *BUY*, *SELL* and *SOLD*, with a function similar like the illocutionary particles that we saw in theory.

```
to process-messages ;; We process each message separately, for convenience, here we already divide the parts of each message (items)
  ask turtles [
    if type-entity = "Gallery" or type-entity = "Collector"
    [
      ;; print (word self type-entity current-messages)
      foreach current-messages [ message ->
        process-message (item 0 message) (item 1 message) (item 2 message) (item 3 message) (item 4 message);; Cada mensaje es una lista [emisor tipo mensaje receptor list-values]
      ]
    ]
  ]
end

to send-message [recipient kind message receiver list-values]
  ;; We add the message to the message queue of the receiving agent
  ;; (it is added to next-messages so that the receiver does not see it until the next iteration)
  ask recipient [
    ;; We put [sender, kind-of-message, message, receiver]
    set next-messages lput (list recipient kind message receiver list-values) next-messages
  ]
end


to process-message [sender kind message receiver list-values]
  if kind = "AD" [
    process-ad sender message receiver list-values
  ]
  if kind = "INFORM" [
    process-inform sender message receiver list-values
  ]
  if kind = "REQUEST" [
    process-request sender message receiver list-values
  ]
  if kind = "RESPONSE" [
    process-response sender message receiver list-values
  ]
  if kind = "BUY" [
    process-buy sender message receiver list-values
  ]
  if kind = "SELL" [
    process-sell sender message receiver list-values
  ]
  if kind = "SOLD" [
    process-sold sender message receiver list-values
  ]
end
```

When a collector, the receiver, process an advertisement message, first chech if the attributes are compatible (the same), and, if is the case, check if the author that is for his interest (index 1 of the preferences' list) is the same that the advertisement. If is true, sends a message confirming to the gallery that is interested. If is false, sends one saying that it's not interesting.

```
to process-ad [sender message receiver list-values]
  print (word [ name ] of sender " -> AD: " message " with values " list-values " to " [ name ] of receiver " at " ticks " ticks")
  ask receiver[
    ;; receiver (a collector) process the ad
    ;; return if is interested in the advertisement
    ;; first check if both of them have the same attributes
    let counter 0
    let compatible true
    foreach preferences-attributes [
      if (item counter preferences-attributes) != (item counter [ ad-attributes ] of sender) [ set compatible false ]
      set counter (counter + 1)
    ]
    ;; if the ad and the collector are compatible
    if compatible [
      ;; we see if the author is interesting for the costumer
      ifelse (item 1 preferences-values) = (item 1 [ ad-values ] of sender)
      [
        send-message self "INFORM" "I'm interested, his/her art is amazing." sender list-values
      ]
      [
        send-message self "INFORM" "I'm not interested, thanks." sender list-values
      ]
    ]
  ]
end
```

When the gallery, the receiver, receives the feedback of the collector, if the collector is interested, tries to sell him the painting. Else, asks for collector's author preference. If the message of inform is that doesn't have paintings of this artist, doesn't do nothing and the communication finish.

```
to process-inform [sender message receiver list-values]
  print (word [ name ] of sender " -> INFORM: " message " with values " list-values " to " [ name ] of receiver " at " ticks " ticks")
  ask receiver[
    ;; print (type-entity)
    ifelse type-entity = "Gallery" [
      if message != "Sorry, we don't have any paintings by this artist." [
        ;; if is interested
        ifelse message = "I'm interested, his/her art is amazing." [
          let sell-message (word "I have the painting '" item 2 list-values "' at " item 0 list-values " billion euros.")
          send-message self "SELL" sell-message sender list-values
        ]
        [ ;; if is not interested asks for costumer's preferences
          ;; print(word name " asks for Collector's preferences.")
          send-message self "REQUEST" "Can you tell me what you are looking for?" sender list-values
        ]
      ]
    ]
    [ ;; BUG: is a Collector (must be a Gallery)
      ;; change xy collector's coordinates
      setxy (item 0 initial-position) (item 1 initial-position + 5)
    ]

  ]
end
```

If the gallery sends a *REQUEST* message, the collector (the receiver) sends his preference author to them.

When the gallery processes the *RESPONSE* message, if has the author tries to sell a painting to the collector. Else, only informs that doesn' have paintings by this artist.

```
to process-request [sender message receiver list-values]
  print (word [ name ] of sender " -> REQUEST: " message " with values " list-values " to " [ name ] of receiver " at " ticks " ticks")
  ask receiver[
    print preferences-values
    let preference-author item 1 preferences-values
    let response-message (word "I'm looking for " preference-author "'s paintings.")
    ;; set preferences-values (replace-item 1 list-values preference-author)
    send-message self "RESPONSE" response-message sender preference-author
  ]
end

to process-response [sender message receiver preference-author]
  print (word [ name ] of sender " -> RESPONSE: " message " with author " preference-author " to " [ name ] of receiver " at " ticks " ticks")
  ask receiver[
    ;; The gallery has paintings of this author
    ifelse member? preference-author own-authors [
      ;; Get the title and price
      let position-painting position preference-author own-authors
      let price-painting item position-painting price-paintings
      let title-painting item position-painting own-paintings
      ;; Create a list of values
      let list-values (list price-painting preference-author title-painting)
      ;; SELL message
      let sell-message (word "I have the painting '" item 2 list-values "' at " item 0 list-values " billion euros.")
      send-message self "SELL" sell-message sender list-values
    ]
    [ ;; otherwise
      ;; sorry message (inform)
      let message-not-available "Sorry, we don't have any paintings by this artist."
      let list-values (list "" preference-author "")
      send-message self "INFORM" message-not-available sender list-values
    ]
  ]
end
```

To process a *SELL* message, the collector (receiver) compares his preference price with the price of the painting and, if is equal or less than his preference price, sends a *BUY* message. Else, the collector negotiates the price: make the medium of the price and sends a *BUY* petition with this new price. In the interface, is here when the collector goes to the gallery.

```
to process-sell [sender message receiver list-values]
  print (word [ name ] of sender " -> SELL: " message " with values " list-values " to " [ name ] of receiver " at " ticks " ticks")
  ask receiver[
    ;; print item 0 preferences-values
    ;; print item 0 [ ad-values ] of sender
    ;; print(item 0 preferences-values) >= (item 0 [ ad-values ] of sender)
    let offer-price item 0 list-values
    let offer-title item 2 list-values
    let preference-price item 0 preferences-values
    ;; change xy collector's coordinates
    setxy ([xcor] of sender - 2) ([ycor] of sender - 5)
    ;; if the preference's price of the collector is equal or lower than the offer
    ifelse preference-price >= offer-price
    [
      let buy-message (word "I want to buy the painting '" offer-title "'")
      ;; sends a buy message
      send-message self "BUY" buy-message sender list-values
    ]
    [ ;; else: negotiate the price
      print(word "NEGOTIATION --> offer: " offer-price " preference: " preference-price)

      ;; get medium price
      let negotiation-price ((preference-price + offer-price) / 2)
      print(word "            --> solution: " negotiation-price)
      ;; BUY at this price
      let buy-message (word "I want to buy the painting '" offer-title "'")
      set list-values (replace-item 0 list-values negotiation-price)
      send-message self "BUY" buy-message sender list-values
    ]
  ]
end
```

The gallery processes the *BUY* message sending a *SOLD* message if hasn't aleready sold the painting (if is the case, sends an *INFORM* message).

```
to process-buy [sender message receiver list-values]
  print (word [ name ] of sender " -> BUY: " message " with values " list-values " to " [ name ] of receiver " at " ticks " ticks")
  ask receiver[
    let title-painting item 2 list-values
    let offer-price item 0 list-values ;; get painting's price
    ;; if the painting is sold
    ifelse member? title-painting paintings-sold-to-process [
      let message-not-available "Sorry, we've already sold this painting."
      send-message self "INFORM" message-not-available sender list-values
    ]
    [ ;; otherwise
      ;; send a SOLD petition (transaction)
      send-message self "SOLD" title-painting sender offer-price
    ]
  ]
end
```

The process of a *SOLD* message consists of updating the gallery's and the collector's money, updating the collector's paintings and setting this painting as sold. In the interface, this is when the collector returns to his initial position.

```
to process-sold [sender message receiver price-painting]
  print (word [ name ] of sender " -> SOLD: " message " with values " price-painting " to " [ name ] of receiver " at " ticks " ticks")
  ask receiver[
    ;; message = title
    set buyers lput name buyers
    let initial-money money
    let finish-money money
    set paintings-sold-to-process lput message paintings-sold-to-process
    set own-paintings lput message own-paintings
    ;; The payment will consist of a small fixed part plus the 1% of the purchase value.
    let price-buyer (price-painting + (0.01 * price-painting))
    set money (money - price-painting)
    print(word "PROCESS SOLD --> " message " -> " name "'s money = " initial-money " - " price-buyer " = " finish-money " billion euros. Paitings in property = " own-paintings)
    ask sender [
      let gallery-money money
      set money (money + price-painting)
      print(word "            --> " name "'s money = " gallery-money " + " price-painting " = " money " billion euros.")
    ]
    ;; change xy collector's coordinates
    setxy (item 0 initial-position) (item 1 initial-position + 5)
  ]
end
```

Each iteration calls this function which has the responsibility to change the colour of painting if it is sold.

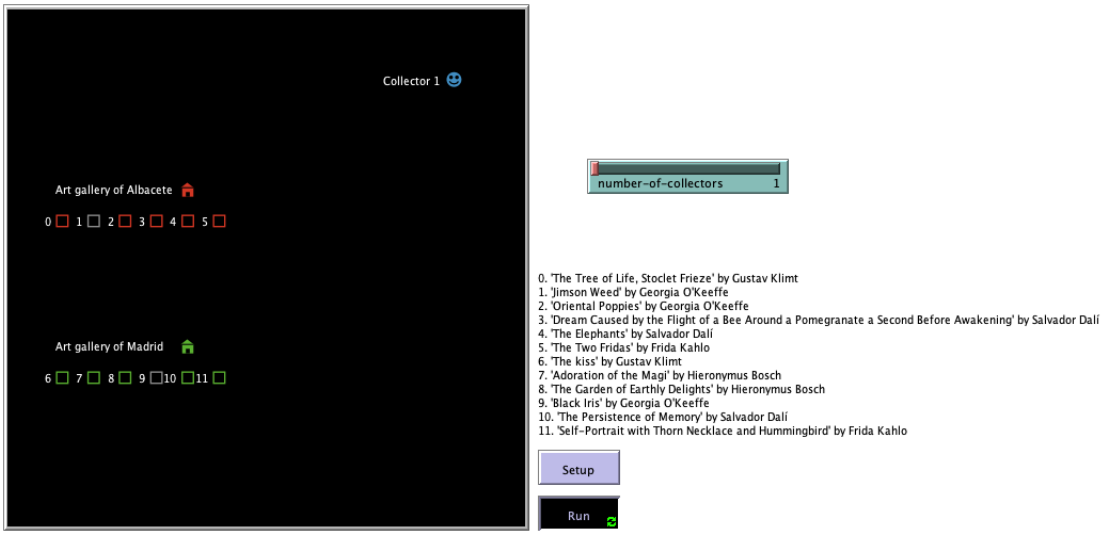
```
;; Function to set the sold paint as gray
to process-paintings-sold
  ask turtles [
    ;; we see one turtle
    let possible-painting-to-process one-of turtles
    ;; if different the nobody
    if possible-painting-to-process != nobody [
      ;; and is a painting
      if [ type-entity ] of possible-painting-to-process = "Painting"
      [
        let is-sold [ sold ] of possible-painting-to-process
        let current-painting-name [ name ] of possible-painting-to-process
        ;; if this painting is in the list to process
        if not is-sold and member? current-painting-name paintings-sold-to-process
        [
          ;; set color gray and sold
          ask possible-painting-to-process [
            set color gray
            set sold true
          ]
        ]
      ]
    ]
  ]
end
```

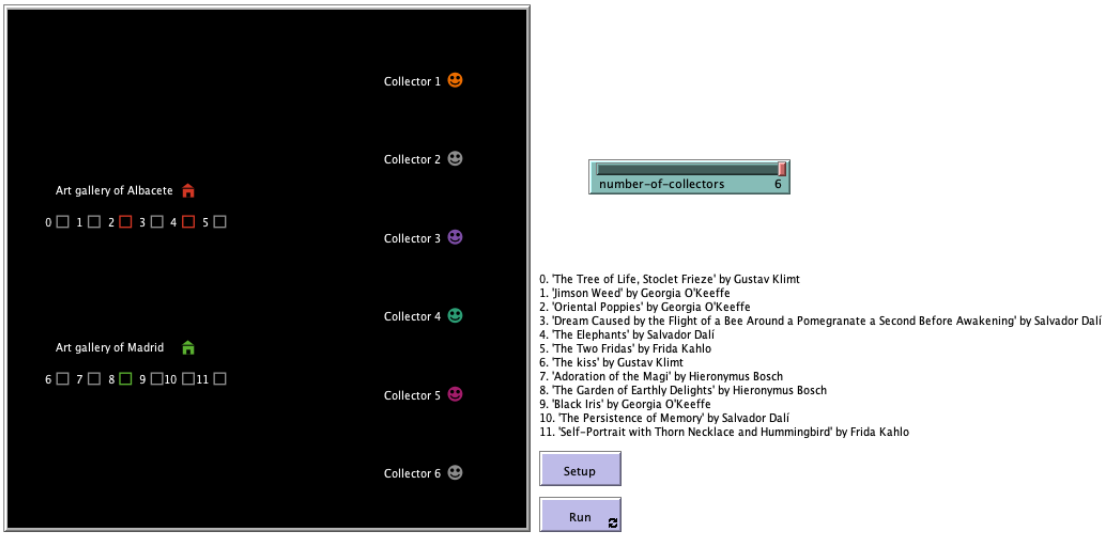
TESTS AND RESULTS

The influence of the value of the number of collectors

Each collector can get as maximum two paintings, because there are two art galleries that contact with them. More collectors mean more likely to sell paintings, which means better performance.

We can see it in the following captures:





With 6 collectors there is more paintings sold (more squares in gray).

CONCLUSIONS

The global behaviour matches with our expectations because we have designed a market between art galleries and collectors that communicates through message passing, with economic transactions when a collector buys a painting and even with the ability to negotiate according to his preferences. And all this is represented in the interface, with the limitations of NetLogo.

With these simulations we have learned the importance of reaching the maximum number of clients in a market and matching their preferences to the products you have. In the beginning, when we had not implemented yet that the art gallery asked for the collector's preferences, when the collector said he was not interested, sales were much lower.

In conclusion, the performance of our market is higher when there are more collectors.