UNIVERSITÀ DEGLI STUDI DI PERUGIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

INTELLIGENT SYSTEMS

A.A. 2019/2020

PROGETTO FINALE
SVILUPPATO UTILIZZANDO LA PIATTAFORMA NETLOGO

Studente: Manuel Severi

MAT: 327372

INDICE

I	Introduzione	3
II	NetLogo e Agent Based Model	3
1	Testo del progetto	5
2	Descrizione del simulatore	7
	2.1 Presentazione interfaccia	7
	2.2 sezione di sinistra: parametri	10
	2.3 guida ad un primo utilizzo e specifica simulazione	11
3	Codice sorgente	12
	3.1 introduzione del codice	12
	3.2 definizione variabili	12
	3.3 definizione procedure	13
	3.4 commento del codice	21
4	Conclusione	22

I | INTRODUZIONE

Il progetto che ho richiesto di sviluppare voleva avere come nodo centrale un modello basato sugli agenti tramite la piattaforma NetLogo. Nella seguente relazione verrà presentato il progetto seguendo le linee accordate con il professor Alfredo Milani. Il progetto completo è consultabile sul mio GitHub personale al seguente link.
(https://github.com/manusevo/NetLogoDiffusioneInfo).

II | NETLOGO E AGENT-BASED MODEL

Figure I: logo NetLogo

NetLogo è un ambiente di modellazione programmabile per la simulazione di fenomeni naturali e sociali. È stato ideato da *Uri Wilensky* nel 1999 e da allora è stato in continuo sviluppo presso Center for Connected Learning and Computer-Based Modeling.
NetLogo è particolarmente adatto per modellare sistemi complessi che si sviluppano nel tempo (all'interno del ambente viene definito tick). I programmatori possono dare istruzioni a centinaia di agenti che operano in modo indipendente. Ciò rende possibile esplorare la connessione tra il comportamento di micro livello degli individui e gli schemi di livello macro che emergono dalla loro interazione.
NetLogo è open source e multipiattaforma scritto principalmente in Java ed è pensato per dei agent-based modeling (modello basato su agenti).

Un modello basato su agenti è una classe di modelli computazionali per simulare le azioni e le interazioni di agenti autonomi al fine di valutare i loro effetti sul sistema nel suo complesso. Combina elementi di teoria dei giochi, sistemi complessi, sociologia computazionale e programmazione evolutiva. I modelli basati su agenti sono una sorta di modello su

microscala che simula le operazioni e le interazioni simultanee di più agenti nel tentativo di ricreare e prevedere la comparsa di fenomeni complessi.

La maggior parte dei modelli basati su agenti sono composti da:

1) numerosi agenti specificati a varie scale (tipicamente indicati come granularità degli agenti);

2) euristica decisionale;

3) regole di apprendimento o processi adattativi;

4) una topologia di interazione;

5) un ambiente. Gli ABM sono tipicamente implementati come simulazioni al computer, sia come software personalizzato, e questo software può essere quindi utilizzato per testare come i cambiamenti nei comportamenti individuali influenzeranno il comportamento generale emergente del sistema.

L'idea di modellazione basata su agenti è stata sviluppata come un concetto relativamente semplice alla fine degli anni Quaranta. Poiché richiede procedure ad alta intensità di calcolo, non si è diffuso fino agli anni '90

NetLogo ha un'ampia documentazione e tutorial. Inoltre viene fornito con la Libreria dei modelli, una vasta raccolta di simulazioni già scritte che possono essere utilizzate e modificate. Queste simulazioni riguardano aree di contenuto nelle scienze naturali e sociali, tra cui biologia e medicina, fisica e chimica, matematica e informatica, economia e psicologia sociale. Sono disponibili diversi programmi di ricerca basati su modelli che utilizzano NetLogo e altri sono in fase di sviluppo.

1 TESTO DEL PROGETTO

Il testo per svolgere il seguente progetto che mi è stato assegnato è il seguente:

il progetto sarà simile, ma non esattamente a quello della trasmissione di un virus:

consiglio di consultare le applicazioni NetLogo VIRUS e FLOCKING nelle librerie fornite con l'applicazione (le trova nel menu File>Models Library>Biology)

tra i parametri generali del sistema che deve implementare vi dovranno essere:

-numero totale di agenti: gli agenti saranno generati in posizioni casuali nel rettangolo a disposizione

-percentuale di agenti con informazione, colorati ad esempio di ROSSO,

-tempo di memorizzazione: dopo quanti istanti di tempo un agente rosso dimentica l'informazione e torna BLU

-velocità: ritmo a cui un agente si sposta casualmente nello spazio di un passo (vedi FLOCKING)

-raggio di visione, per ciascun agente indica l'area attorno a sé in cui "vede" la presenza di altri agenti (vedi FLOCKING)

-probabilità trasmissione: un numero tra 0 e 1es. 0.30

FUNZIONAMENTO

ad ogni istante un agente si sposta a caso a seconda della velocità stabilità (che può essere zero)

se è un agente ROSSO guarda tutti gli agenti intorno a sé entro il RAGGIO DI VISIONE e per

ciascun agente in raggio di visione, estrae un numero random tra 0 e 1, se è tra 0 e la probabilità di trasmissione cioè maggiore della probabilità di

trasmissione, colora di ROSSO il nuovo agente e azzera il contatore di memorizzazione

incrementa il tempo di memorizzazione dell'agente se questo supera il parametro corrispondente torna BLU”

Come richiesto dal professore per svolgere il seguente progetto ho usato la libreria di NetLogo Flocking, sviluppata da Uri Wilensky, coperta dalla licenza Creative Commons Attribution-NonCommercial-ShareAlike 3.0, la libreria in questione è usata per controllare il movimento degli agenti.

2 DESCRIZIONE DEL SIMULATORE

Nella seguente sezione verranno spiegate le varie funzionalità degli elementi grafici presenti nel simulatore stesso, inoltre verrà fornita una guida sull'utilizzo del simulatore per effettuare una simulazione impostando i parametri desiderati.

2.1 PRESENTAZIONE INTERFACIA

L'interfaccia presentata nel seguente simulatore è stata divisa in tre sezioni per facilitare l'esperienza utente nel setting e nella visualizzazione della simulazione:

- La sezione a sinistra (come visibile nella figura sottostante numero 2) presenta i bottoni e gli slider per la specifica dei parametri essenziali nella simulazione;

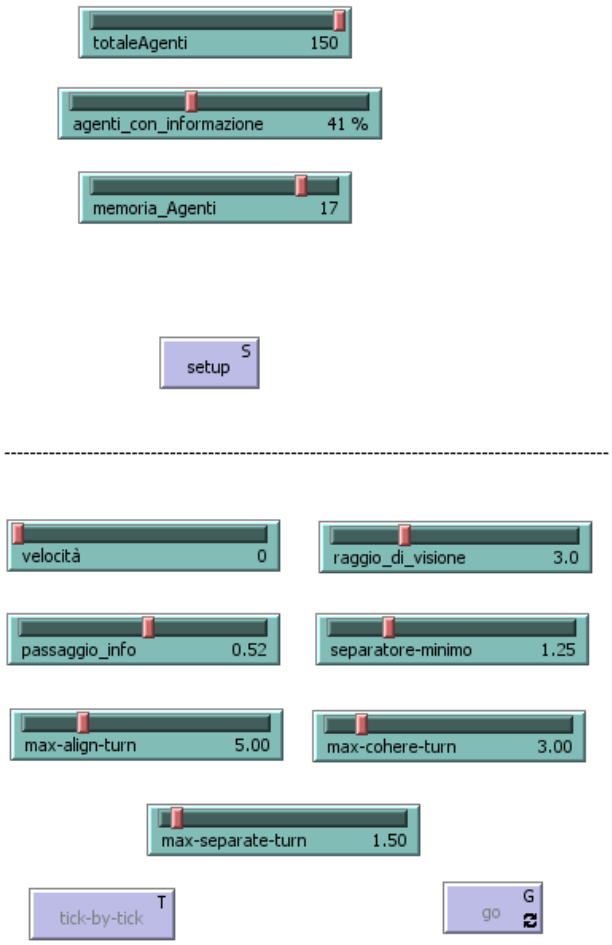

Figura II: sezione dei parametri

Come si può notare questa sezione viene poi “divisa” in due parti. La prima contiene le informazioni sul numero degli agenti, la percentuale di persone informate e quanta memoria ogni agente avrà (parametro che è possibile modificare anche durante l’esecuzione). Mentre la parte sottostante sono tutti i parametri che fanno parte del Flocking, inclusa la velocità che nel progetto di Wilensky è di default a 5 mentre in questo progetto è regolabile da 0 (gli agenti rimangono fermi) fino a 20.
Infine sono presenti i pulsanti che permettono di avviare il simulatore in maniera continua (pulsante go) oppure tick per tick (pulsante tick-by-tick) che si attivano solo una volta premuto setup che oltre a spawnare le persone avvia il conteggio dei tick

- Nella sezione centrale è presente un riquadro con il quale si può avere una visione in tempo reale della simulazione specificata dai parametri settati nella sezione appena presentata.

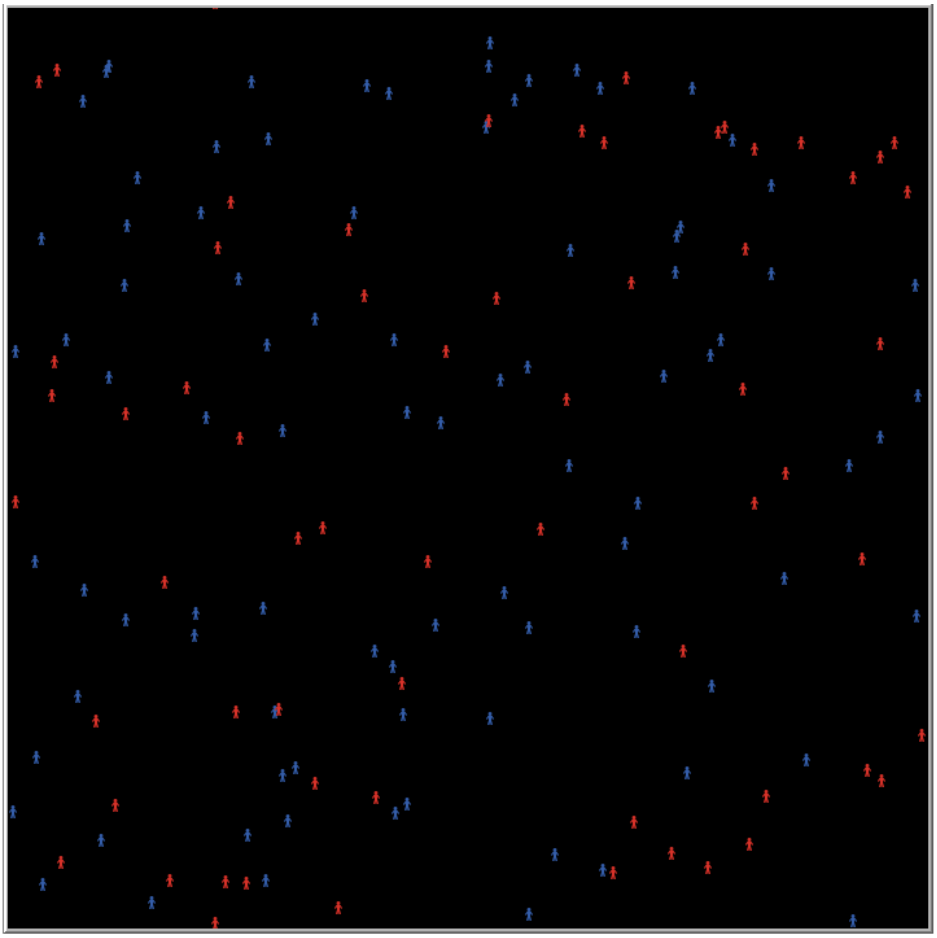

Figura III: Word view

- Nell'ultima sezione, quella di destra, troviamo i bottoni per seguire un determinato agente (che si attivano solo dopo aver cliccato setup), spawnare un nuovo agente informato (utile nel caso tutti gli agenti diventino disinformati) e i dati sul numero di perdite di memoria e di agenti informati. Inoltre vengono creati in tempo reale due grafici; nel primo si mette in relazione il numero di agenti informati con quelli senza informazione mentre nel secondo viene mostrato l'andamento delle perdite di memoria, naturalmente, fortemente connesso al valore della memoria degli agenti.

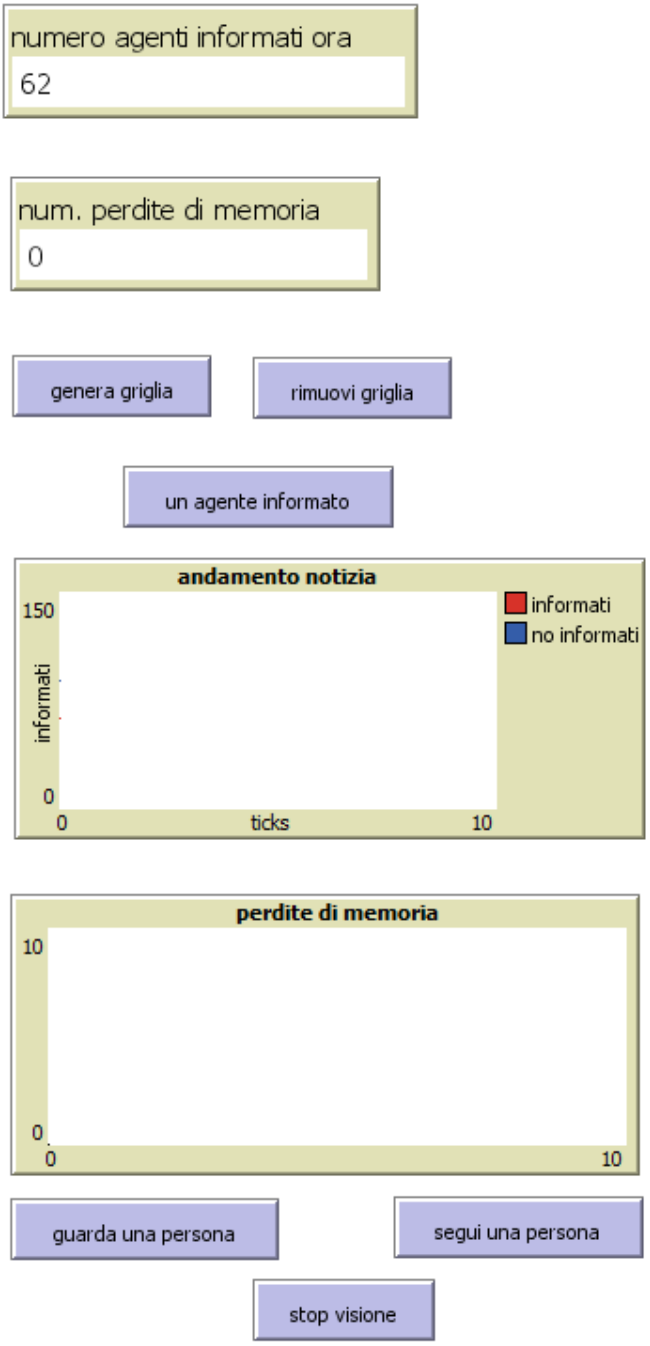

Figura IIIV: Sezione di destra

2.2 SEZIONE DI SINISTRA: PARAMETRI

Partendo dall'alto il primo slider "*totaleAgenti*" imposta il parametro che permette di creare il numero desiderato di agenti (numero agenti = agenti con informazione + agenti senza informazione).
a seguire troviamo lo slider "*agenti_con_informazione"* che indica la percentuale degli agenti che devono possedere l'informazione al momento che viene generato il mondo. Troviamo ora lo slider "*memoria_Agenti"* dove è possibile, in ogni momento della simulazione, indicare la memoria che avrà ogni singolo agente quando riceve l'informazione. Una volta impostati i parametri sopracitati si può passare cliccare su "*setup"* che si occupa di generare il mondo.

Nella seconda sottosezione troviamo tutti i parametri utili a Flocking, che possono essere modificati in ogni momento dell'esecuzione, nello specifico:

- Velocità → aggiunta per determinare la velocità con cui si spostano gli agenti, ricordo che se impostata a 0 gli agenti rimangono praticamente sempre nella stessa posizione.
- Raggio_di_visione → che ci permette di indicare quanto è ampia la vista di un agente, maggiore è il raggio di visione, maggiore sarà la possibilità di entrare in contatto con altri agenti.
- Passaggio_info → la possibilità che ha un agente informato che entrato in contatto con un agente disinformato di passargli l'informazione. Nel caso di un valore uguale a 0 l'informazione non viene scambiata, contrariamente con valore uguale a 1 l'informazione sarà sicuramente condivisa.
- Separatore-minimo, Max-align-turn, Max-cohere-turn, Max-separate-turn → tutti valori utili per il FLOCKING

Infine troviamo i due bottoni che avviano la simulazione e sono "*tick-by-tick"* e "*go"*. Entrambi richiamano la stessa funzione, ovvero, GO ma mentre nel primo caso viene eseguita una sola volta, nel secondo caso viene ripetuta fino a nuovo click grazie al parametro settato a vero sul bottone di nome Forever.

2.3 GUIDA AD UN PRIMO UTILIZZO E SPECIFICA SIMULAZIONE

All’avvio del simulatore, non è possibile premere tutti i pulsanti ma bisogna premere, come già detto in precedenza, il bottone “*setup*” per inizzializzare il contatore dei tick di clock così da fargli assumere il valore 0.
All’avvio della simulazione, gli elementi presenti sullo schermo interagiscono tra di loro, mostrando queste interazioni nel riquadro centrale.
Una volta che sono presenti tutti gli elementi della simulazione, è possibile premere il bottone “*go*” per far partire la simulazione e osservare il comportamento degli agenti nel mondo. In ogni caso, è sempre possibile stoppare la simulazione, ripremendo il medesimo pulsante e modificare in tempo reale alcuni parametri come detto nelle pagine precedenti.

3 CODICE SORGENTE

Nella seguente sezione vengono spiegate le variabili, gli oggetti e le funzioni definite nel codice proposto come soluzione al problema d'esame.

3.1 INTRODUZIONE DEL CODICE

I primi oggetti che vengono definiti sono le turles, ovvero oggetti in grado di muoversi, cambiare colore, forma e altre proprietà. Sono inoltre in grado di immagazzinare informazioni sotto forma di variabili proprietarie. Ogni agente generico che viene generato appartiene alla classe turtle, ma è possibile definire degli agenti differenti appartenenti a "sottoclassi" dette breed che permettono di differenziare gli agenti, facendogli possedere variabili proprie e comportamenti che si differenziano in base alle scelte del programmatore. All'interno del codice vengono definite sia il nome della breed (al plurale), sia quello del generico agente che ne fa parte (al singolare).

```
1.  breed [agenti agente]
```

N.B. è importante specificare come, se un agente fa parte di una breed, esso viene comunque considerato anche un turtle.

3.2 DEFINIZIONE VARIABILI

In NetLogo, principalmente, sono presenti tre tipologie di variabili:

- **Variabili globali**: esitono in esemplari unici e sono visibili ovunque nel codice. Devono essere dichiarate fuori da ogni procedura e aggiornate con il comando "*set nome_variabile possibile_valore_default"* oppure direttamente dall'interfaccia, come abbiamo visto, attraverso slider, switch o chooser
- **Variabili proprietarie**: possono genericamente appartenere a tutte le turles (indicate con *tutles-own*), patches (indicate con *patches-own*) oppure ad una breed specifica (indicate con *nome_breed-own*), le

variabili che ho appena indicato sono replicate per ogni agente di cui appartengono.

- **Variabili locali**: sono dichiarate con *"let nome_variabile valore"* all'interno di una routine e possono essere utilizzate solo in quella specifica routine.

A seguire vengono mostrate le variabili globali e proprietarie che sono state definite nella soluzione, che devono essere inserite prima delle procedure.

```
1.  agenti-own[
2.   flockmates
3.   nearest-neighbor
4.   memoria
5.   informato? ;; booleana per sapere se agetne è informato oppure no
6. ]
7. globals
8. [
9.  agent_informati
10.   Smemorati
11.  ]
```

3.3 DEFINIZIONE PROCEDURE

All'interno del linguaggio di programmazione NetLogo ci sono due tipologie di procedure:

- **Subroutine:** sono richiamate da altre subroutine o direttamente dall'interfaccia per mezzo della pressione di un button. Possono ricevere valori in ingresso, ma non restituiscono alcun valore in uscita. Iniziano sempre con la parola chiave to e termina con end
- **Reporter:** sono richiamati da altri reporter o subroutine. Possono ricevere valori in ingresso e restituiscono sempre un valore in uscita, attraverso il comando report. Iniziano sempre con la parola chiave to-report e termina con end.

3.3.1 SETUP

La funzione "*setup"* richiamata dal medesimo bottone si occupa di ripulire tutta la cache del programma, resettare i tick e creare i nuovi agenti per avviare una nuova simulazione.

```
1. to setup
2.   clear-all
3.   reset-ticks
4.   creo_Agenti
5. end
```

come si evince dal codice appena mostrato, viene richiamato anche la funzione **creo_Agenti**, che si occupa della creazione degli agenti all'interno del mondo.

```
1. to creo_Agenti
2.
3.   let con_Agenti (totaleAgenti * agenti_con_informazione) / 100
   ;;creazione numero agenti con informazionni
4.   create-agenti round (totaleAgenti - con_Agenti)[ ;;creazione agenti
   senza informazioni
5.     set shape "person"
6.     set color blue
7.     setxy random-xcor random-ycor
8.     set memoria memoria_Agenti
9.     set informato? false
10.    ]
11.    create-agenti round con_Agenti[ ;;creazione agenti con informazioni
12.      set shape "person"
13.      set color red
14.      setxy random-xcor random-ycor
15.      set memoria memoria_Agenti
16.      set informato? true
17.    ]
18.    set agent_informati round(con_Agenti)
19.    set smemorati 0
20.  End
```

La prima cosa che viene fatta all'interno di questa funzione è calcolare il numero esatto di agenti che nascono con l'informazione. Il codice crea inizialmente gli agenti senza informazione andandogli ad attribuire una forma (*shape "person"*), un colore (*color blue*), una posizione casuale all'interno del mondo (*setxy random-xcor random-ycor*), settargli una memoria in base al valore dello slider presente nell'interfaccia e assegnare alla variabile booleana *informato?* il valore false così da indicare, appunto, che questo agente non ha l'informazione.
Si prosegue creando gli agenti che conoscono l'informazione che hanno le

stesse caratteristiche degli agenti appena descritti tranne che per il colore, vengono indicati con il rosso e naturalmente la variabile booleana ha il valore true.
Infine si vanno a inizializzare due variabili.

3.3.2 FUNZIONE GO

La funzione go, che possiamo definire come l'anima del programma, richiama le procedure del Flocking per il movimento e le procedure per diminuire memoria e passare l'informazione ai vicini.

```
1. to go
2.   ask agenti [ flock ]
3.   repeat velocità [ ask agenti [ fd 0.2 ] display ]
4.   diminuire_memoria
5.   ask agenti with [informato?][passa_informazione]
6.   tick
7. end
```

Inizia chiedendo a praticamente tutti gli agenti di eseguire la funzione flock e poi li sposta nel display in base alla velocità che viene presa dallo slider in tempo di esecuzione, perciò modificabile ogni qual volta l'utente voglia. Viene poi richiamata la funzione *diminuire_memoria* il cui codice sorgente è il seguente:

```
1. to diminuire_memoria
2.   ask agenti with [informato?]
3.   [
4.     ifelse memoria = 0
5.     [
6.       set color blue
7.       set informato? false
8.       set memoria memoria_Agenti
9.       set agent_informati agent_informati - 1
10.        set smemorati smemorati + 1
11.      ]
12.      [set memoria memoria - 1]
13.    ]
14.  end
```

La funzione appena presentata, a grandi linee, si occupa di controllare che tutti gli agenti che sono informati non abbiamo concluso il loro tempo di memoria e che perciò debbano tornare disinformati cambiando alcune caratteristiche dell'agente. Se invece l'agente ha ancora memoriaa disposizione questa viene ridotta ad ogni tick.

Continuando con l'esecuzione della procedura *go*, troviamo il momento in cui si va a controllare se vicino ad agenti senza informazione vi siano alcuni con informazione per provare a scambiarla richiamando la funzione *passa_informazione* il cui codice è sotto esposto.

```
1. to passa_informazione
```

```
2.    find-flockmates
3.    let num_casuale random-float  1
4.    ;;print num_casuale
5.    if any? flockmates
6.    [
7.      ask flockmates with [not informato?][
8.        if num_casuale <= passaggio_info
9.        [
10.           set informato? true
11.           set color red
12.           set agent_informati agent_informati + 1
13.         ]
14.       ]
15.     ]
16.   end
```

Inizialmente si controllano i propri vicini e si genera un numero casuale compreso tra 0 e 1 e se un vicino è informato e il numero casuale3 è minore o uguale al valore impostato come passaggio_info (che ricordo essere la probabilità che un agente informato condivida l'informazione con uno non informato) l'agente non informato diventa rosso si cambia il valore della variabile booleana e si aumenta il numero di agenti informati.

3.3.3 FLOCKING

Questo modello è la un tentativo di imitare il flocking o il movimento delgi uccelli (che è anche simile a quello dei pesci che si muovono in banchi). Infatti gli agenti all'interno di un gruppo non segue un "leader" ma bensì segue un insieme di regole che sono uguale anche per i suoi compagni. Le regole in particolare sono tre:

1) Allineamento → significa che un agente cerca di muoversi nella stessa direzione in cui si muovono altri agenti.
2) Separazione → un agente si muove cercando di non avvicinarsi troppo ad un altro agente
3) Coesione → un agente cerca di spostarsi per stare vicino agli altri fino a che altri non sono troppi vicini

Quando due agenti sono troppo vicini, la regola di "separazione" prevale sugli altri due, che vengono disattivati fino al raggiungimento della separazione minima. A differenza del Flocking normale nel mio progetto la velocità può essere impostata dall'utente modificata in corso di esecuzione. I parametri importanti per questa parte sono velocità, raggio di visione, passaggio info, separatore minimo, max align turn max cohere turn, max separate turn come è possibile vedere anche dalla parte di codice sorgente riportata.

```
1. flock  ;; turtle procedure
2.   find-flockmates
3.   if any? flockmates
4.     [ find-nearest-neighbor
5.       ifelse distance nearest-neighbor < separatore-minimo
6.         [ separate ]
7.         [ align
8.           cohere ] ]
9. end
10.
11.  to find-flockmates  ;; turtle procedure
12.    set flockmates other turtles in-radius raggio_di_visione
13.  end
14.
15.  to find-nearest-neighbor ;; turtle procedure
16.    set nearest-neighbor min-one-of flockmates [distance myself]
17.  end
18.
19.  ;;; SEPARATE
20.
21.  to separate  ;; turtle procedure
22.    turn-away ([heading] of nearest-neighbor) max-separate-turn
23.  end
```

```
24.
25.  ;;; ALIGN
26.
27.  to align  ;; turtle procedure
28.    turn-towards average-flockmate-heading max-align-turn
29.  end
30.
31.  to-report average-flockmate-heading  ;; turtle procedure
32.    ;; We can t just average the heading variables here.
33.    ;; For example, the average of 1 and 359 should be 0,
34.    ;; not 180.  So we have to use trigonometry.
35.    let x-component sum [dx] of flockmates
36.    let y-component sum [dy] of flockmates
37.    ifelse x-component = 0 and y-component = 0
38.      [ report heading ]
39.      [ report atan x-component y-component ]
40.  end
41.
42.  ;;; COHERE
43.
44.  to cohere  ;; turtle procedure
45.    turn-towards average-heading-towards-flockmates max-cohere-turn
46.  end
47.
48.  to-report average-heading-towards-flockmates  ;; turtle procedure
49.    ;; "towards myself" gives us the heading from the other turtle
50.    ;; to me, but we want the heading from me to the other turtle,
51.    ;; so we add 180
52.    let x-component mean [sin (towards myself + 180)] of flockmates
53.    let y-component mean [cos (towards myself + 180)] of flockmates
54.    ifelse x-component = 0 and y-component = 0
55.      [ report heading ]
56.      [ report atan x-component y-component ]
57.  end
58.
59.  ;;; HELPER PROCEDURES
60.
61.  to turn-towards [new-heading max-turn]  ;; turtle procedure
62.    turn-at-most (subtract-headings new-heading heading) max-turn
63.  end
64.
65.  to turn-away [new-heading max-turn]  ;; turtle procedure
66.    turn-at-most (subtract-headings heading new-heading) max-turn
67.  end
68.
69.  ;; turn right by "turn" degrees (or left if "turn" is negative),
70.  ;; but never turn more than "max-turn" degrees
71.  to turn-at-most [turn max-turn]  ;; turtle procedure
72.    ifelse abs turn > max-turn
73.      [ ifelse turn > 0
74.          [ rt max-turn ]
75.          [ lt max-turn ] ]
76.      [ rt turn ]
77.  end
```

3.3.4 CODICE DELLA PARTE GRAFICA

Per riuscire a creare in tempo reale i due grafici di cui abbiamo accennato nel capitolo precedente, ho utilizzato la procedura di NetLogo plot che permette di disegnare. Nel caso specifico

```
1. plot count agenti with [not informato?]
2. plot count agenti with [informato?]
```

Che rappresenta nel grafico il numero di agenti informati e non informati in un determinato tick.

Per generare la griglia si è usata la seguente funzione

```
1. ask patches [set pcolor random 10]
```

mentre per rimuoverla e tornare allo sfondo del mondo in nero la funzione o procedura per meglio dire è la seguente

```
1. ask patches [set pcolor black]
```

Il primo contatore che mostra il numero degli agenti informati in un dato momento è una procedura molto semplice.

```
1. count agenti with [informato?]
```

Il codice per guardare una persona, seguirla e fermare la visione di un singolo agente nell'ordine è:

```
1. watch one-of turtles
```

```
1. follow one-of turtles
```

```
1. reset-perspective
```

3.4 COMMENTO DEL CODICE

Ovviamente tutte le funzioni interagiscono tra di loro durante l'esecuzione della simulazione, inoltre, si evidenzia come alcune di loro vengano richiamate a cascata, come anche visibile nei codici elencati in precedenza.

Le funzioni cercano di emulare in modo fedele il comportamento di una folla che si scambia le informazioni e che la stessa folla si muove cercando comunque di rimanere "unita".

Ovviamente, il comportamento delle persone è analogo a quello delle persone reali, cercando sempre di muoversi coerentemente con un movimento veritiero, prestando attenzione alle altre persone.

4 CONCLUSIONE

Con il seguente programma si vuole mostrare come l'interazione tra diversi individui correlato al loro movimento possa influenzare nella diffusione di un'informazione.
Si evidenzia dal tipo di movimento come le persone cercano di rimanere, per quanto possibile, in gruppo senza sovrapporsi e come questo favorisca la diffusione delle informazioni all'interno di un gruppo. Eseguendo diverse simulazioni si può notare come quando anche pochi elementi di un gruppo sono informati, generalmente, tutti gli altri membri del gruppo di informano oppure, nel caso in cui l'informazione abbia scarsa "trasmissibilità", tutti gli agenti della simulazione diventano disinformati.