Sistemi Autonomi
Birds of a feather flock together
Bird Flocking Simulation

Alessandro Contro, Lorenzo Monti

2 agosto 2016

Sommario

Lo scopo di questo progetto è quello di dimostrare il concetto di autonomia, di coordinazione e auto organizzazione in un sistema complesso all'interno di un MAS ed utilizzare questi ultimi nello loro diverse parti, attraverso la simulazione, al fine di testare l'efficacia e la correttezza del modello. Il tema scelto punta la sua attenzione sulla formazione degli stormi di uccelli in fase di migrazione (entità auto organizzanti) e sul concetto di prede/predatori (entrambi entità autonome). Questa simulazione si concentra sulla gestione e cura delle entità principali (due specie) e possiede aspetti sia di autonomia da un lato che di interazione con l'utente (osservatore) dall'altro. La nostra simulazione è sicuramente semplificata rispetto la complessità che possiamo trovare in realtà ma mira a seguire un modello comportamentale che emula la vita di un uccello all'interno di uno stormo. La tecnologia scelta per lo sviluppo di questo lavoro è NetLogo, un linguaggio di programmazione basato ad agenti dotato del suo IDE di sviluppo che permette anche un agile implementazione della parte grafica dell'applicazione.

Indice

1 Introduzione

Molte specie di uccelli sono gregarie e formano stormi per diversi motivi. Gli stormi (flocks) variano in dimensione e si formano in diverse stagioni. Gli uccelli volano in stormi perché stare in un gruppo molto grande porta alcuni vantaggi per la sopravvivenza di ciascun componente.

1.1 Vantaggi nell'appartenere ad uno stormo

Foraging (Ricerca di cibo) Muoversi in stormo durante la ricerca di cibo permette ad un gran numero di uccelli di nutrirsi dalla stessa fonte. Inoltre dà la possibilità a tutto lo stormo di trovare il cibo che un singolo componente ha già trovato in precedenza.

Protezione Un vasto gruppo di uccelli ha una maggiore possibilità di individuare un predatore o qualsiasi altra minaccia che un singolo componente del gruppo può avere. Inoltre, un gran numero di uccelli ragruppati hanno la possibilità di confondere o sopraffare un predatore. Infine stare in uno stormo presenta ad un predatore un gran numero di bersagli, in questo modo ogni singolo ha una probabilità inferiore di divenire la preda di qualche altro animale.

Riproduzione In uno stormo i maschi riescono a venire in contatto con molti più esemplari femminili, aumentando le possibilità di accoppiamento.

Allevare la prole Gli uccelli più giovani sono più al sicuro grazie alla protezione offerta dallo stormo.

Aerodinamicità Quando gli uccelli volano in stormi, spesso si organizzano in determinate formazioni in modo da sfruttare al meglio le condizioni del vento così da spostarsi in maniera energicamente efficiente.

Calore Nei climi più freddi lo stormo ha il beneficio del calore condiviso da ogni componente per sopravvivere.

1.2 Svantaggi

Nonostante i vantaggi siano molti, gli uccelli devono fare un tradeoff con qualche rischio quando si uniscono ad uno stormo.

Visibilità Più grande è il numero più lo stormo è visibile ai predatori nella zona.

Concorrenza Gli stormi più grandi hanno bisogno di più cibo e vi è molta più concorrenza durante la fase di accoppiamento.

Malattie Quando così tanti uccelli stanno vicini c'è un rischio molto maggiore di diffusione di malattie.

2 Goals

Si vuole simulare la formazione degli stormi di uccelli in fase di migrazione in concomitanza al concetto di prede/predatori al fine di mostrare concetti chiave visti a lezione come l'autonomia, la coordinazione e l'auto organizzazione. Il progetto verrà implementato attraverso il tool di sviluppo NetLogo, tool che è stato introdotto a lezione.
In particolare si vuole:

1. Descrivere il comportamento degli stormi di uccelli in fase di migrazione.
2. Implementare in maniera più realistica possibile quelli che sono i comportamenti individuali degli uccelli negli stormi.
3. Ricreare l'interazione tra preda/predatore al fine di sopravvivere.
4. Semplificare quanto più possibile il passaggio dalla realtà alla simulazione.

2.1 Discussione dei goal

Comportamento degli stormi Si vuole realizzare il modello e conseguente simulazione del comportamento, per quanto riguarda il movimento nello spazio (simulazione in 2D), di uccelli raggruppati in stormi. Questo comportamento è soggetto a fattori come il cambiamento climatico, ricerca di cibo, presenza di predatori, stagione di accoppiamento e si presenta maggiormente durante le fasi migratorie.

Comportamento individuale Essendo lo stormo un sistema distribuito auto-organizzante (senza capo branco), l'observer interverrà sul singolo individuo modificando i parametri che andranno ad influenzare il comportamento dello stormo stesso.

Interazione preda/predatore e sopravvivenza Nella fattispecie il nostro modello asserirà all'accoppiata Falco Pellegrino/Storno essendo il caso più esemplare di predatore rapace su stormi di uccelli.

Semplificare il passaggio alla simulazione Rendere più facile possibile il passaggio da realtà a simulazione, fornendo all'observer diversi metodi di parametrizzazione, riducendo la quantità di informazioni da derivare/inferire dalla realtà.

3 Architettura del sistema

Il modello tenuto in considerazione è sicuramente semplificato rispetto alla realtà ma serve a veicolare in modo snello la rappresentazione virtuale, realizzata mediante l'uso di entità autonome, di alcuni aspetti della vita degli uccelli negli stormi durante la migrazione in relazione con i predatori rapaci.

3.1 Struttura

All'interno del modello è stata innanzitutto creata un' area composta da Patches Netlogo e delimitata dalla finestra grafica, dove le Turtles Netlogo (entità attive) possono muoversi. Questa area inoltre offre la possibilità alle prede di nutrirsi ove le Patches Netlogo sono di colore verde.
Vi sono calati all'interno del ambiente alcuni esemplari di due specie diverse, l'una preda e l'altra predatore, ognuna con l'obiettivo di sopravvivere. L'utente rimane come Observer del sistema, ne osserva i cambiamenti e può decidere di interagire con questo mediante l'utilizzo di appositi bottoni e slider. Sia il terreno che le specie avranno delle proprietà che li descriveranno individualmente e delle interazioni che gli permetteranno di cambiare queste variabili personali.

3.2 Comportamento

Lo scopo del lavoro è di valutare una simulazione multi-livello creata per descrivere in maniera più esaustiva possibile sia lo stato individuale delle varie entità (uccelli) che popolano l'area, sia il livello macro del sistema come insieme delle sue interazioni sociali. A tal proposito siamo riusciti ad estrapolare informazioni sul comportamento degli stormi da [1, 4] che analizzeremo quì di seguito.

3.2.1 Comportamento delle prede

Le prede sono state modellate come segue:

Nascita: ogni preda "nasce" sul terreno in una certa posizione, con una certa energia e con una velocità individuale.

Si muoveranno in stormi: ogni uccello preda tenderà a muoversi in stormi sulla base di alcune semplici regole senza la presenza di un capo branco, in accordo con lo studio di [1], ed alcuni slider offerti all'Observer per poter interagire.

Cibarsi: aspetto fondamentale che attraverso l'interazione con il terreno permette agli uccelli prede di cibarsi (Patches verdi) quando sono affamati (parametrizzato) al fine aumentare la propria energia per sopravvivere.

Paura: è stato implementato il concetto di paura nei confronti dei rapaci nelle vicinanze così da poter, in accordo con l'articolo [4], schivare il pericolo deviando la propria traiettoria individuale temporaneamente.

Riproduzione: due animali delle stessa specie possono generare un figlio, l'evento prenderà la metà dell'energia del genitore che lo crea. E' stato aggiunto il parametro libido che raddoppia la probabilità di procreazione.

Morte: Nel caso in cui l'energia si esaurisca, l'uccello preda morirà immediatamente.

3.2.2 Comportamento dei predatori

Per quanto concerne il modello dei predatori invece:

Nascita: ogni predatore "nasce" sul terreno in una certa posizione, con una certa energia e con una velocità individuale.

Caccia degli uccelli preda: l'unica chanche di sopravvivenza da parte di un predatore è trovare una preda e cacciarla al fine di sostentarsi aumentando la propria energia, come per la preda anche il rapace mangia solo quando ne sente la necessità attraverso il parametro *sized-stomach-rapacious*. Nella nostra simulazione e' stato aggiunto il parametro *rapacious-gain-from-food* che indica il quantitativo di energia assunto dal rapace una volta consumata la preda.

Riproduzione: due animali delle stessa specie possono generare un figlio, l'evento prenderà la metà dell'energia del genitore che lo crea. E' stato aggiunto il parametro libido che raddoppia la probabilità di procreazione.

Morte: Nel caso in cui l'energia esaurisca l'uccello preda morirà immediatamente.

3.3 Interazione

Nel nostro sistema esistono entità mobili e immobili. Le prime possono essere rappresentate tramite agenti le seconde invece possono essere rappresentate da semplici variabili. Gli agenti sono sistemi computazionali in grado di compiere azioni autonome e di percepire eventi dall'ambiente in cui sono immersi. Per questo si adattano perfettamente al nostro problema. Sia le prede che i predatori infatti possono essere modellate come agenti (*Turtles* in *NetLogo*), in quanto possiedono le loro caratteristiche, tra le quali:

Reactiveness: dato che gli uccelli, sia prede che predatori, hanno bisogno di reagire agli stimoli dell'ambiente;

Situatedness: le azioni compiute degli uccelli dipendono anche dal contesto in cui si trovano; sulla base di ciò che si prospetta di fronte, devono tenere comportamenti diversi;

Proactiveness: non devono attendere solamente che qualcosa accada, ma prendono iniziativa per raggiungere i propri obiettivi;

Social ability: non devono agire da sole ma collaborare per raggiungere i propri obiettivi. Gli stormi ne sono sicuramente un buon esempio.

Stato: hanno bisogno di uno stato, ovvero conoscenze che vengono continuamente aggiornate, come ad esempio variabili inerenti allo stormo così da sapere quale azione intraprendere.

Dopo aver evidenziato quali sono gli agenti si è passati alla struttura dell'ambiente circostante. In *NetLogo* l'ambiente e' gestito tramite *Patches*, entità immobili che vanno a formare il background in cui le *Turtles* saranno immerse e con cui interagiranno.
Nel caso specifico avremo un terreno virtuale con cui le prede (una delle due

specie *Turtle*) potranno cibarsi solo quando si troveranno di fronte ad una *Patch* di colore verde (se è marrone non sarà disponibile il cibo). Inoltre ambedue le specie di *Turtle* saranno soggette ad un interazione con *Patches*, se questo fossero di colore nero identificherebbe la presenza di ostacoli che ogni agente dovrà necessariamente scansare.

4 Progettazione

4.1 NetLogo ed il suo IDE

NetLogo è un linguaggio di modellazione basato ad agenti progettato per la simulazione didattica di fenomi naturali e sociali. Esso fornisce al programmatore un ambiente di sviluppo corredato da un'interfaccia grafica in grado di mostrare sia i dati di elaborazione sia componenti atti all'interazione con l'utente. NetLogo applica il paradigma ad agenti dividendoli , secondo una particolare declinazione logica, in *Turtles*, *Patches* ed *Observer* (entità esterna che osserva e può influenzare il corso degli eventi sulla base dei parametri a sua disposizione). Le *Patches* sono parti del mondo nel quale vivino e interagiscono le *Turtles*. Progettate come entità attive ma immobili, possono eseguire comandi e interagire con *Turtles* e *Patches* adiacenti. Ogni *Patches* è dotata di coordinate cartesiane intere. Le *Turtles* sono agenti base dell'ambiente di simulazione e possono rappresentare qualsiasi oggetto (nel nostro caso uccelli), ogni *Turtles* possiede sia una posizione (con coordinate decimali) sia una orientamento (direzione), una forma ed eventualmente una "penna" per segnare il percorso.
Dal punto di vista tecnico *NetLogo* è in grado di supportare migliaia di agenti (32000) che vengono eseguiti in parallelo ed indipendemente su thread differenti e tutto ciò è trasparente al programmatore, questa è sicuramente uno dei punti di forza di questo IDE. *NetLogo* è quindi usato per rappresentare i cambienti di stato del sistema in funzione del tempo (rappresentato come sequenza di tick-clock). Ad ogni *Tick*, *Turtles* e *Patches* eseguono le istruzioni scritte per essi e grazie alla programmazione parallela implementata dal sistema *NetLogo* le espanderà a tutte le entità. Il linguaggio, utilizzato per impartire compiti agli agenti ed al background, offre funzioni, primitive, procedure e variabili con scope differenti (globali, breed, locali o built-in). La flessibilità di questo IDE ci permette di poter modellare simulatori di ogni genere. La lavoro svolto da noi prende in considerazione un sistema formato da un certo numero di *Patches* che simulano un'area aperta (come ad esempio un campo) e delle istanze specifiche di *Turtles* istanziate con parametri differenti; anche per l'utente è possibile interagire con la simulazione, se lo riterrà necessario, al fine di modificarne lo svolgimento.
L'interfaccia grafica editabile di NetLogo divide i suoi elementi in tre classi diverse:

Controls: fanno parte di questa categoria i bottoni ed il centro di comando, i bottoni possono essere "once button" cioè avere una sola esecuzione oppure "forever button" ed essere eseguiti fino a che non vengono ripremuti. Il centro di comando invece prende posto nella parte bassa dell'interfaccia e permette all' *Observer* di interagire con la simulazione attraverso l'inserimento di istruzioni.

Settings: fanno parte di questa categoria gli sliders, gli switches ed i choosers presenti nell'interfaccia e servono per impostare i parametri iniziali della nostra simulazione. A fronte del cambio di queste variabili è possibile osservare un mutamento nell'evoluzione della simulazione.

Views: nelle views ricadono i monitors, plots, output e parti grafiche della finestra. Sono i campi dove vengono riassunti dei dati numerici, in maniera testuale o graficati con le curve e l'area dell'interfaccia grafica stessa.

4.2 Schema di comportamento

Per rendere la simulazione del nostro progetto più veritiera possibile ci siamo affidati agli articoli [1, 4] che abbiamo letto ed analizzato per capire gli schemi di comportamento degli uccelli in stormi (prede) e della loro relazione con i predatori. Esaminando con attenzione la formazione degli stormi emerge il concetto di *coordinamento distribuito* come risultato di un processo di *auto-organizzazione*, un tema di grande interesse sia in robotica collettiva che nello studio di comportamenti di gruppo negli animali [8, 9]. Infatti i movimenti fluidi dello stormo sono il risultato *emergente* di ogni singolo uccello che segue semplici regole in maniera autonoma.
Per completezza evidenziamo come dopo l'analisi fatta attraverso gli articoli letti, gli schemi comportamento hanno effettivamente tutti gli elementi tipici dell' *auto-organizzazione*:

1. **Incremento di ordine:** E' lapalissiano come la creazione di uno stormo rende il sistema meno caotico.

2. **Autonomia:** sia a livello individuale che a livello di stormo locale vi è sicuramente un concetto di autonomia.

3. **Adattatività:** sia il sistema che l'uccello individuale rispondono opportunamente ai cambiementi posti dall'esterno.

4. **Dinamica:** sistema e individuo sono continuamente soggetti a fasi di feedback, questo lo rende un processo e non uno stato finale.

4.2.1 Formazione dello stormo

Anche se è diffusa l'opinione che ciò che porta gli uccelli a formare movimenti organizzati in stormi sia la presenza di un "capo" all'interno del gruppo, non è così. Piuttosto ogni elemento dello stormo prende a riferimento l'uccello a lui più vicino, applicando un concetto di *coordinazione distribuita* oltre che di *auto organizzazione*, cercando di allinearsi alla direzione da esso assunta. È questo che determina il movimento "a gruppi". Per il nostro progetto ci siamo attenuti rigorosamente allo studio di [1], ed andremo quì di seguito ad estrapolarne i concetti chiave:

1. Ogni uccello ha una certa direzione e mantiene una distanza rispetto ai suoi vicini.

2. il singolo individuo può andare dove vuole ma il sistema impone all'uccello di reagire sulla base dei movimenti dei suoi vicini nello stormo.

3. Ogni esemplare visualizza una "propria" regione da seguire (sulla base della propria percezione), fuori da quello tutto sarà ignorato. Questo è il motivo cardine per cui lo stormo vira.

Sulla base delle premesse sopracitate lo studio estrapola tre semplici regole che sono alla base della creazione degli stormi di uccello:

1. **Separazione:** applicata dall'uccello quando lo stormo locale è troppo affollato (evitando quindi possibili collisioni) che quindi vira per uscirne e trovarne uno "più vivibile".

2. **Allineamento:** applicata dall'uccello quando deve entrare nello stormo che quindi cerca di orientarsi verso il centro dello stesso.

3. **Coesione:** applicata una volta conclusa la fase di allineamento, quì l'esemplare cerca di virare verso il centro dello stormo locale.

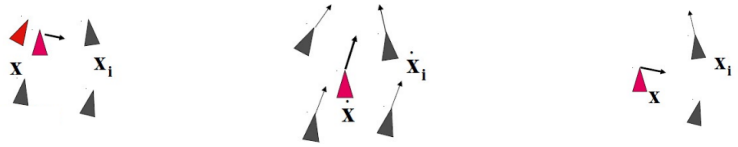

Figura 1: Separazione, allineamento e coesione

4.2.2 Paura dei predatori

Uno dei motivi per cui gli stormi esistono è quello della protezione, avendo innanzitutto maggior probabilità di indivuare un possibile predatore e conseguentemente avere una maggior probabilità di sopravvivere ad un attacco. Date le specie prese in esame in questo progetto ci discostiamo leggermente dal paper [4] per quanto rigarda la tipologia di attacco del rapace e la conseguente risposta dello stormo, ma applichiamo peressiquamente i punti cardine riportati in esso. Anche in questo caso abbiamo di fronte una coordinazione distribuita da parte dello stormo

1. **Dispersione:** La situazione in cui il predatore si avvicina lo stormo ed ogni individuo che ne percepisce la presenza triggera in esso un allarme.

2. **Deviazione:** Azione conseguente all'allarme triggerato da parte di ogni uccello, quì la preda cerca di calcolare la direzione del predatore e deviare la propria per poter sopravvivere all attacco.

3. **Aggregazione:** Una volta passato il pericolo ogni preda cerca di ricostruire lo stormo in accordo con le regole dello studio [1].

4.3 Diagramma delle classi

Tutti gli uccelli presenti nel progetto (uccelli preda e uccelli predatori) derivano direttamente dalla classe agente base delle *Turtles* di NetLogo. Per poter graficare l'andamento nel tempo degli uccelli, le variabili *energy*, *flockmates* e *nearest-neighbor* sono impostate in maniera globale. *avoid-obstacle* e *death* sono gli unici metodi che comprendono ambedue le specie, il primo fa evitare gli ostacoli agli agenti in campo (*Patches* nere) che l' *Observer* può inserire durante la simulazione, mentre il secondo gestisce la morte dell'agente per mancanza di energia, il metodo si preoccupa di "killare" l'agente facendo scomparire dal campo.
Nel codice è stata fatta una suddivisione sulla base della specie (dichiarando attraverso le variabili built-in *breed birds* e *rapaciouses*) così da poter gestire separatamente i due tipi di *Turtles*.
Per quanto concerne la specie *birds* abbiamo diverse variabili e metodi. La variabile *eaten-bird* regola la propensione da parte dell'agente di cibarsi sulla

base del parametro *sized-stomach-bird*, la variabile *velocity-b* serve a definire una velocità tipica del singolo individuo, *find-rapacious*, *nearest-rapacious* e *direction*(direzione personale per capire la traiettoria di fuga sulla base della traiettoria del rapace in avvicinamento) sono le variabili imputate al riconoscimento di un rapace in avvicinamento così da triggerare il concetto di paura nella preda, infine con *libido-b* andremo ad aumentare la propensione alla riproduzione da parte della preda.Per quanto riguarda ai metodi invece, andremo quì di seguito ad analizzarli:

move-bird: gestisce il movimento dei singoli animali all'interno dell'arena, esso varia sulla base innanzitutto della velocità che ogni indivuo ha acquisito nel momento del settaggio iniziale (randomico) e sulla base del appetito del volatile.

eat-berry: gestisce l'assunzione di cibo da parte degli animali, quando è disponibile il cibo l'animale lo assume, convertendo la *Patch* da verde a marrone inoltre verrà aggiornata l'energia aggiungendo all'energia residua quella impostata dal parametro *bird-gain-from-food*.

avoid-rapacious: gestisce la possibile sopravvivenza da parte della preda rispetto il predatore, questo metodo infatti

reproduce-bird: è la funzione che permette lo spawn di un nuovo agente. Per semplicità la riproduzione è randomica sulla base del parametro *bird-reproduce*, inoltre la libido è attivata all'interno dell'animale la probabilità di riprodursi raddoppia.

flock: gestisce la propensione da parte di un agente ad entrare/muoversi/uscire da uno stormo sulla base di determinati parametri.

Per quanto concerne la specie *rapaciouses* abbiamo invece la variabile *eaten-rapacious* che regola la propensione da parte dell'agente di cibarsi sulla base del parametro *sized-stomach-bird*, la variabile *velocity-r* serve a definire una velocità tipica del singolo individuo similmente a quanto accade per gli uccelli preda, *find-bird* e *nearest-bird* sono le variabili imputate al riconoscimento di una preda in avvicinamento così da poterla seguire per cacciarla, infine con *libido-r* andremo ad aumentare la propensione alla riproduzione da parte del rapace.Per quanto riguarda ai metodi invece, andremo quì di seguito ad analizzarli:

move-rapacious: in maniera simile alle prede questo metodo gestisce il movimento dei singoli animali all'interno dell'arena, varia sulla base della velocità individuale acquisita nel momento del settaggio iniziale (randomico) e sulla base del appetito del volatile.

follow-bird: metodo con l'obiettivo di seguire i rapaci al fine di cattarli e potersi così sostentare.

catch-bird: procedura atta all'uccisione delle prede con lo scopo di riacquisire energia. Una volta che la preda è stata presa, viene fagocitata dal rapace e viene sommata all'energia rimanente quella impostata dal parametro *rapacious-gain-from-food*.

reproduce-rapacious: similmente a quanto accade per gli agenti preda, questa è la funzione che permette lo spawn di un nuovo agente. Per semplicità la riproduzione è randomica sulla base del parametro *bird-reproduce*, inoltre la libido è attivata all'interno dell'animale la probabilità di riprodursi raddoppia.

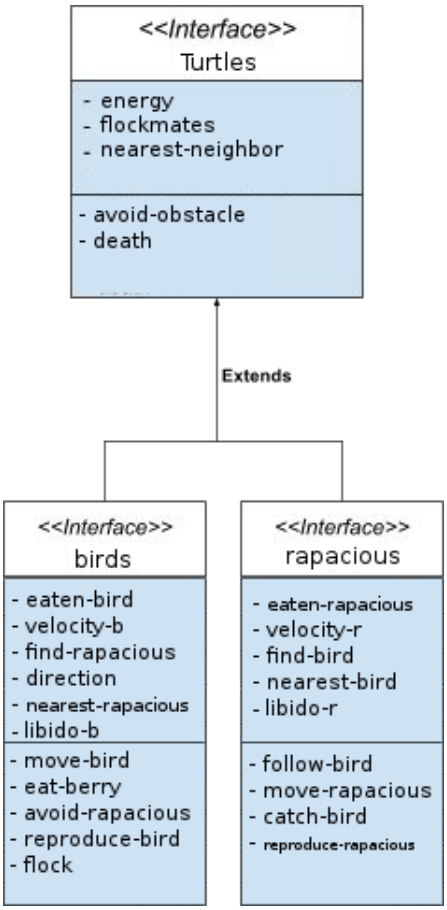

Figura 2: diagramma delle classi

4.4 Flowchart delle componenti

In questa sezione andremo a descrivere ogni componente presente nel nostro progetto attraverso diagrammi di flusso. Avremo tre entità principali: gli *uccelli preda* e gli *uccelli predatori* che estendono la classe *Turtle* di *NetLogo* e le *Patches*.

4.4.1 Flowchart delle prede

Le prede sono una delle due specie (che estende la classe *Turtle*) dihiarate in Netlogo per il nostro progetto. Dopo una fase di setup iniziale la preda entra nello stato *Ready*(con le sue variabili interne inizializzate), tramite il tasto Go l'agente "prende vita" andando ad eseguire le azioni che comporranno il suo comportamento dinamico all'interno dell'area creata. Mentre è in questo stato l'agente può compiere diverse azioni che mirano tutte a cercare di sopravvivere, infatti l'animale mangerà ogni qual volta trovarà cibo ed è affamato, aggirerà gli ostacoli in modo da non subire danni, si riprodurrà sulla base della propria libido(che gli costerà metà della sua energia vitale); Tutto quesasdasdto ovviamente costerà all'animale un certo quantitativo di energia. Se l'agente avvista uno stormo già formato, come descritto in [1] l'agente cercherà di inserirsi nello stormo, dall'altra parte se l'agente si trova già nello stormo ma questo è troppo popoloso cercherà di allontanarsi. Essendoci agenti predatori le prede nutrono un naturale senso di paura dettato anche questa volta dal sopravvivere, quindi quando questo sentimento è triggerato nell'animale esso entra nello stato di panico (sia che sia nello stormo o in solitaria) e cercherà di scappare dall'agente predatore, anche in questo caso vi è un dispendio di energie (doppio rispetto al caso di riposo).La *Turtle* autonomamente cercherà sempre di portare a termine le azioni che gli permetteranno di sopravvivere, ma la situazione di cambiamento costante all'interno dell'area influenzerà questa sua capacità.*L'observer* può sopraggiungere in aiuto e modificare i parametri in itinere supervisionando così l'andamento della simulazione.

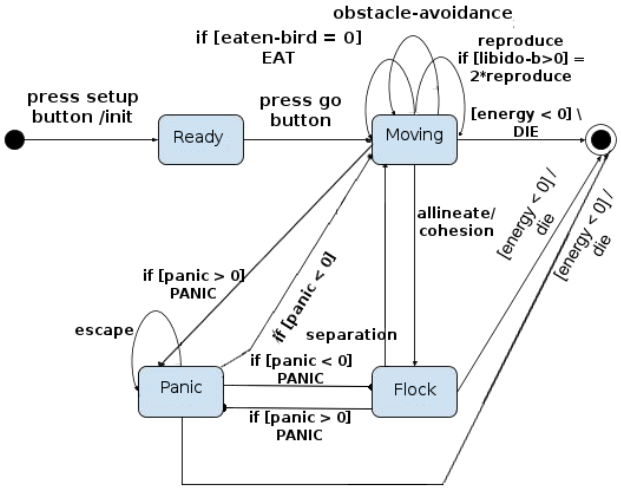

Figura 3: Prey Flowchart

4.4.2 Flowchart dei predatori

I predatori, come le prede, sono una specie dichiarate in NetLogo per portare avanti la nostra simulazione. Come per le prede anche i predatori hanno uno stato iniziale di settaggio che li porta allo stato di *Ready*. Una volta premuto il bottone "Go" inizierà anche per i predatori la simulazione nella quale potersi muovere,scansare gli ostacoli e riprodursi sulla base della propria libido, tutto questo ha un costo in termini di energia. L'unica chance di sopravvivenza da parte del predatore è quella di cacciare gli agenti preda che, una volta avvistati l'agente si troverà nello stato di "Hunt" in cui cercherà in tutti i modi di uccidere una preda così da poter aumentare la sua energia (anche per lui questo ha un costo doppio); Nel caso non ci siano più prende nelle vicinanze il predatore tornerà nello stato di "Moving" pronto a cacciare appena avvisterà una nuova preda.

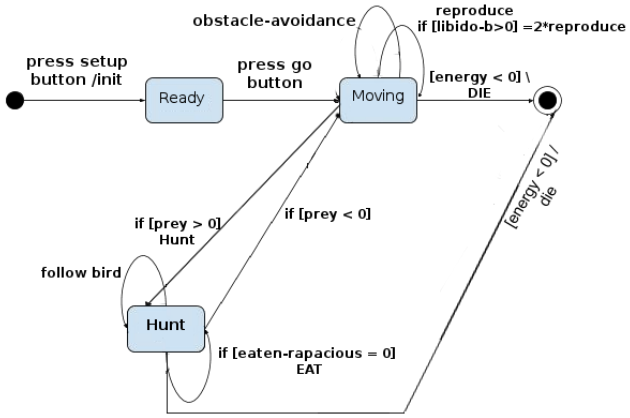

Figura 4: Predator Flowchart

4.4.3 Flowchart delle patch

Nell'immagine vengono presentati tutti gli stati che può assumere l'agente patch durante l'esecuzione del nostro programma. Dopo aver premuto il tasto di setup iniziale questa assumerà o il colore verde (che contraddistingue la presenza di cibo per gli agenti preda) o il colore marrone (che identifica il terreno). Se la *Turtle* preda andrà sopra ad una *Patch* di color verde si potrà rinfocillare nel caso abbia appetito e renderà quest'ultima terreno senza cibo (marrone). Nel caso invece la *Turtle* preda cada su una *Patch* color marrone passerà senza potersi cibare, dopo un determinato quantitativo di tempo il cibo ricrescerà e la *Patch* associata tornerà ad essere di colore verde.L' *Observer* può interagire con la simulazione attraverso un tasto che inserisca ostacoli (*Patches* color nero) che tutte le *Turtle* dovranno schivare.

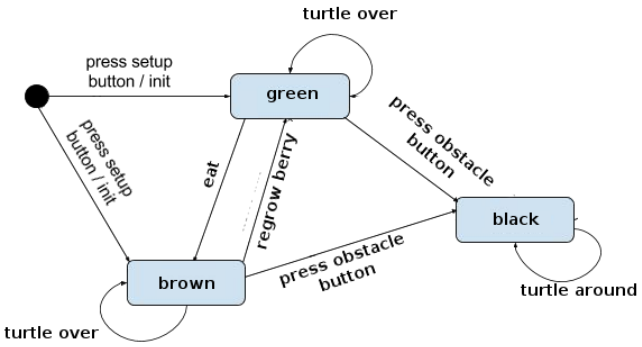

Figura 5: Patch Flowchart

5 Implementazione in NetLogo

Di seguito vengono presentate le parti di codice più importanti. È possibile consultare il codice completo ampiamente commentato nel file allegato al progetto.

5.1 Declare

Nella sezione Declare vengono dichiarate le variabili e le entità che compongono il nostro sistema.
Vengono dichiarate due specie: birds e rapaciouses. Ciascun elemento di una specie possiede uno stato determinato dalle variabili in *turtles-own*; a seconda dell'appartenenza a una delle due specie, le turtles avranno un set di variabili diverso.

```
;== DECLARE
globals [berry]  ;; keep track of how much berry there is
breed [birds bird]
breed [rapaciouses rapacious]
patches-own [countdown]
turtles-own [
  energy
  flockmates          ;; agentset of nearby turtles
  nearest-neighbor    ;; closest one of our flockmates
]
birds-own[
  eaten-bird
  velocity-b
  find-rapacious
  direction
  nearest-rapacious
  libido-b
  ]
rapaciouses-own[
  eaten-rapacious
  velocity-r
  find-bird
  nearest-bird
  libido-r
]
```

5.2 Setup

Setup è un insieme di istruzioni che permette di azzerare la simulazione e ripartire con un nuovo set di parametri selezionati dall'observer.

```
;== SETUP
to setup
  clear-all
  ;; check berry?.
  ;; if it is true, then berry grows and the bird eat it
```

```
  ;; if it false, then the bird don't need to eat
  if berry? [
    ask patches [
       ;; ask patches to inizialize "world"
      set pcolor one-of [green brown]
         ;; set color (green or brown)
      if-else pcolor = green
        [ set countdown berry-regrowth-time ]
           ;; initialize berry grow clocks for green
           patches
        [ set countdown random berry-regrowth-time ]
           ;; initialize berry grow clocks randomly for
           brown patches
    ]
  ]

  ;; create breed
  set-default-shape rapaciouses "airplane"
  create-rapaciouses population-rapacious
     ;; create the rapacious, then initialize your
     variables
  [
    set color red - 2 + random 7
    set velocity-r random-float 1
    set size 4
       ;; a lttle bit bigger than birds
    set label-color blue - 2
    set energy random 150
    setxy random-xcor random-ycor
  ]

  set-default-shape birds "default"
  create-birds population-bird
    [ set color yellow - 2 + random 7
       ;; random shades look nice
      set velocity-b random-float 1
      set size 3.5
         ;; easier to see
      set direction direction setxy random-xcor random-
         ycor  ;; coordinate
      set energy random 50
         ;; default energy (random)
      set flockmates no-turtles
         ;; set "flockmates = 0"
    ]

  set berry count patches with [pcolor = green]
     ;; this is for monitor in inteface tab
  reset-ticks
end
```

5.3 Go

Go è il set di istruzioni che viene eseguito ad ogni iterazione e permette di far proseguire la simulazione. In questa sezione del codice viene definito il comportamento che viene eseguito ad ogni iterazione da ogni agente sulla base dell'interazione con gli altri agenti e l'ambiente.

```
;== GO
to go
  if not any? turtles [ stop ]
  ask turtles [ avoid-obstacle ]
     ;; obstacle avoidance per tutte le turtles
  ask birds [ flock ]
     ;; flocking move for all birds

  ;; birds thing
  ask birds[
    if berry? [                                          ;;
        if berry? is set
      move-bird
      set energy energy - 1                              ;;
          reduce energy for bird only if berry? switch is
          on
      if eaten-bird = 0 [eat-berry]
      set eaten-bird eaten-bird - 1
      ]
      avoid-rapacious
      death
      reproduce-bird
  ]

  ;; rapaciouses thing
  ask rapaciouses[
    follow-bird
    move-rapacious
    set energy energy - 1
   ; output-print eaten-rapacious
    if eaten-rapacious = 0 [ catch-bird]               ;;
       if it hungry, catch bird
    if eaten-rapacious > 0
    [set eaten-rapacious eaten-rapacious - 1]
    death
    reproduce-rapacious
  ]

  if berry? [ ask patches [ grow-berry ] ]             ;;
     subroutine for grow food (for birds).
  set berry count patches with [pcolor = green]        ;;
     this is for monitor in inteface tab.
  tick                                                 ;;
     advances the tick counter by one.
```

```
  display-labels                                                ;;
     subroutine for display turtles energy
end
```

5.4 Flock

Questa procedura viene eseguita solamente dagli uccelli preda e come si vedrà nel prossimo pezzo di codice vi sono incorporate le regole che permettono il volo in stormi.

```
;== FLOCK

to flock  ;; birds procedure
  find-flockmates
  if any? flockmates
                                                          ;; if
     exist at least one bird in the flockmates
    [ find-nearest-neighbor
                                                  ;; find the
       nearest
      ;output-print distance nearest-neighbor
      ifelse distance nearest-neighbor < minimum-
         separation       ;; if too close, separate else
         align and cohesion
        [ separate ]
                                                             ;;
            turn away to prevent overcrowding in local
           flock
        [ align
          cohere ]
    ]
end

to find-flockmates  ;; birds procedure
  set flockmates other birds in-radius vision
                        ;; set flockmates of a bird like
      other birds in "vision" patches
end

to find-nearest-neighbor ;; birds procedure
  set nearest-neighbor min-one-of flockmates [distance
     myself]    ;; set nearest neighbor of bird like the
     distance from me and bird's flockmates closer
end
```

5.5 Regole di flocking: Separate, Align e Cohesion

Regole tratte da [1] che permettono agli uccelli preda di raggrupparsi e assumere una formazione a stormo.

Queste regole permettono al singolo uccello di valutare se unirsi ad uno stormo o proseguire da solo.

```
;== FLOCK RULEZ

;; SEPARATE
to separate  ;; turtle procedure
  turn-away ([heading] of nearest-neighbor) max-separate-
     turn
end

;; ALIGN
to align  ;; turtle procedure
  turn-towards average-flockmate-heading
  max-align-turn
end

;; COHESION
to cohere  ;; turtle procedure
  turn-towards average-heading-towards-flockmates
  max-cohere-turn
end

;== ALIGN PROCEDURE

to-report average-flockmate-heading  ;; birds procedure
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  let x-component sum [dx] of flockmates
  let y-component sum [dy] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;== COHESION PROCEDURE

to-report average-heading-towards-flockmates  ;; birds
    procedure
  ;; "towards myself" gives us the heading from the other
      bird
  ;; to me, but we want the heading from me to the other
     bird,
  ;; so we add 180
  let x-component mean [sin (towards myself + 180)] of
     flockmates
  heading from me to other birds
  let y-component mean [cos (towards myself + 180)] of
     flockmates
```

```
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end
```

6 Sviluppi futuri

Come già detto questa simulazione è il risultato di una modellazione semplificata rispetto alla realtà ma è comunque possibile migliorarla aggiungendo ulteriori vincoli al modello:

1. **Aggiunta del vento:** il vento può, in alcuni casi essere un parametro discriminatorio in termini di sopravvivenza, sicuramente una sua implementazione potrebbe rendere la simulazione più reale.
2. **Stagioni:** sicuramente si potrebbero introdurre le stagione e conseguentemente tutte le variabili metereologiche del caso che andranno ad influire il comportamento dell'individuo.
3. **Migliorare la riproduzione:** una volta introdotte le stagione si potrebbe migliorare il processo di riproduzione degli uccelli.
4. **Rappresentazione in 3D:** sicuramente l'*Observer* potrebbe determinare in maniera migliore il comportamente degli stormi e quindi impostare la simulazione in modo più preciso.

7 Conclusioni

Il lavoro svolto con questo progetto ci ha permesso di applicare alcuni concetti sull'autonomia e sull'auto-organizzazione visti a lezione e legati alla programmazione ad agenti. Grazie a *NetLogo* abbiamo implementato un semplice MAS che permette ad alcuni agenti di interagire in un ambiente simulato così da modellare diversi gradi di autonomia e rappresentarli all'interno delle entità che sono state create. Siamo soddisfatti dell'IDE *NetLogo* che ci ha permesso una rapida prototipizzazione senza doverci occupare di meccanismi di basso livello (risorse, metodi di interazione e thread) ne di gestione di interfaccia grafica focalizzando le energie sulla simulazione vera e propria.
La simulazione che abbiamo presentato soddisfa in pieno tutti i goal che ci eravamo preposti a inizio progetto.

Riferimenti bibliografici

[1] Craig E. Reynolds *Flocks, Herds and Schools: A Distributed Behavioral Model*, Computer Graphics, volume 21, number 4, July 1987;

[2] William A. Thompson, Ilan Vertinsky, John R, Krebs *The survival value of flocking in birds: a simulation model*, Journal of Animal Ecology, Vol.43, No. 3 (Oct, 1974), pp. 785-820;

[3] Glen A. Dimock, Michael S. Selig *The aerodynamic benefit of self-organization in bird flocks*, 41st AIAA Aerospace Sciences Meeting and Exhibit, January 6-9, 2003/Reno, NV;

[4] Naga Chaitanya Byrisetty *Agent-based modeling to simulate the movement of a flock of birds*, October 2012, Fargo, North Dakota;

[5] U. Wilensky, *NetLogo Flocking model*, Center for Connected Learning and Computer-Based Modeling, Northwestern University, 1988. [Online]; Available: http://ccl.northwestern.edu/netlogo/models/Flocking

[6] David N. Bonter, Benjamin Zuckerberg, Carolyn W. Segwick, Wesley M. Hochachka *Daily foraging patterns in free-living birds: exploring the predation-starvation trade-off*, Proc R Soc B 280: 20123087. http://dx.doi.org/10.1098/rspb.2012.3087

[7] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandini, G. Parisi, A. Procaccini, M. Viale, V Zdravkovic *Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study*, PNAS, 29 Gennaio, 2008, vol. 105, no. 4, pp. 1232-1237;

[8] Baldassarre et al., 2002; Ijspeert et al., 2001 *Evolving mobile robots able to display collective behaviour. In Hemel-rijk C.K.(ed.)*, International Workshop on Self-Organisation and Evolution of Social Behaviour, pp. 11-22. Zurich: Swiss Federal Institute of Technology.;

[9] Ijspeert A.J., Martinoli A., Billard A., Gambardella L.M. (2001). *Collaboration through the exploitation of local interactions in autonomous collective robotics: The stick pulling experiment.* Autonomous Robots, vol. 11, pp. 149-171.