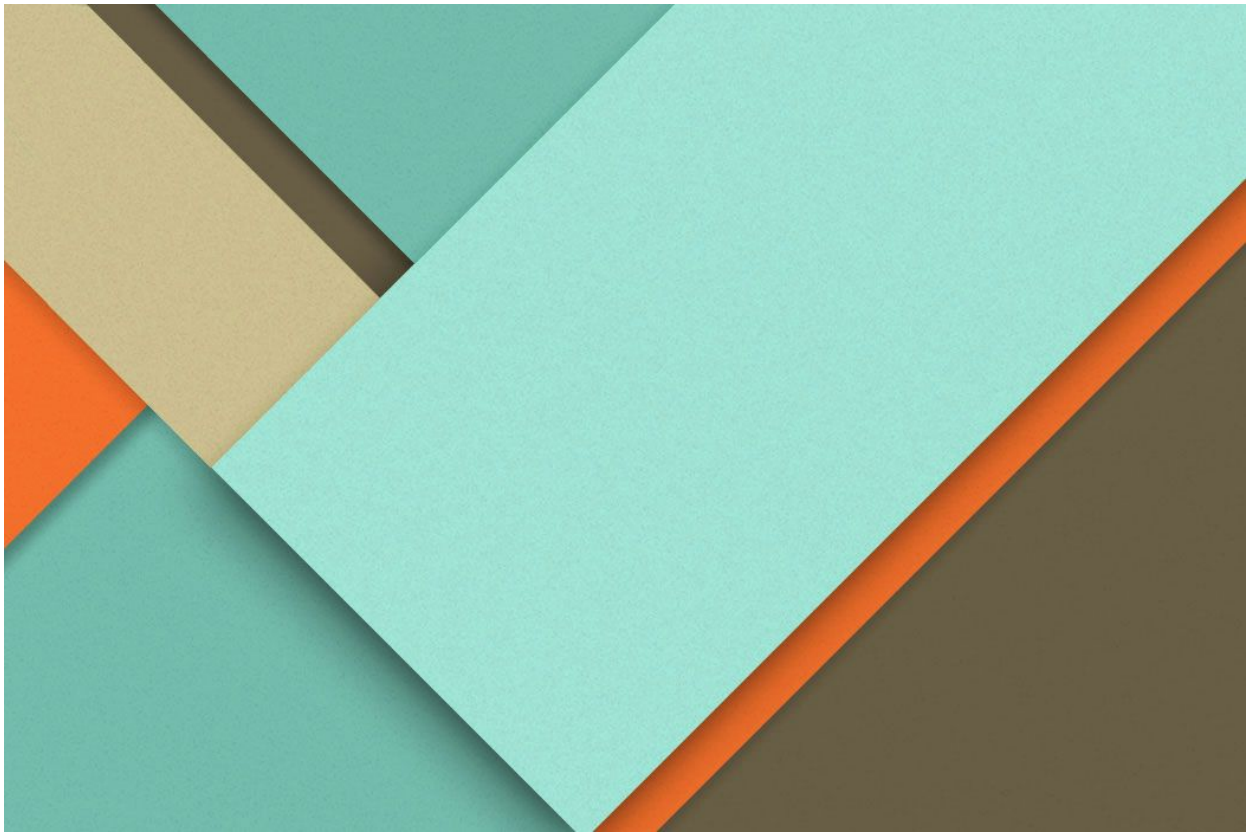


Farmagochi

Progetto di Sistemi Autonomi

Matteo Bocchetti
matteo.bocchetti@studio.unibo.it

Francesco Maughelli
francesco.maughelli@studio.unibo.it

Introduzione

Lo scopo di questo progetto è quello di dimostrare il concetto di autonomia all'interno di un MAS ed utilizzarla nelle diverse parti che lo compongono attraverso l'impiego della simulazione come mezzo per testarne l'efficacia e la correttezza. Il tema scelto è quello dell'implementazione di un gioco di tipo life-simulator dove i protagonisti sono degli animali (entità autonome) all'interno di una fattoria. Questa tipologia di gioco (ad es. Tamagochi) si concentra sulla gestione e cura delle entità principali e possiede aspetti sia di autonomia da un lato che di interazione con l'utente (osservatore) dall'altro. La simulazione che scaturisce è semplificata ma mira a seguire un modello comportamentale che emula quello degli animali da cortile. 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.

Modello

Il modello tenuto in considerazione è semplificato ma serve a veicolare in modo snello la rappresentazione virtuale della vita degli animali (dai loro primi movimenti fino alla loro morte) realizzata mediante l'uso di entità autonome. Al centro di questo modello prende posto una fattoria (area delimitata dalla finestra grafica, composta da numerose *patches* NetLogo ed identificata da un colore iniziale verde) dove vengono calati alcuni esemplari di animali (le *turtles* NetLogo), l'utente rimane come *Observer* del sistema, ne osserva i cambiamenti e può decidere di interagire con questo mediante l'utilizzo di appositi bottoni. Sia il terreno della fattoria che gli animali avranno delle proprietà che li descriveranno individualmente e delle interazioni che gli permetteranno di cambiare queste variabili personali. Gli animali sono caratterizzati da un livello di energia, si muoveranno in maniera random all'interno del recinto e potranno interagire con il terreno per mangiare o fare i bisogni. Tra animale ed animale invece ci sarà una interazione sociale (positiva o negativa influenzata da più fattori) che determinerà un livello di happiness (felicità). Tutte le interazioni vengono descritte quì di seguito:

1. **Quattro animali previsti**: divisi per sesso e per razza (la razza determina anche la possibilità di riproduzione): Pecora, Montone, Mucca e Toro.
2. **Animale caratterizzato da:**
    2.1. **Movimento:** costante e randomico, se è disponibile del cibo a terra l'animale lo mangia senza esitazione.

2.2. **Ciclo di vita:** l'animale inizia con un certo livello di *energia* e resta in vita fintantochè questa non giunge a zero.

2.3. **Bisogni:** l'assunzione di cibo provoca periodicamente dei bisogni nell'animale che "sporcano" una certa area del recinto.

2.4. **Riproduzione:** due animali di razza compatibile (pecora con montone oppure mucca con toro) possono generare un figlio se il loro stato d'animo è elevato e c'è abbastanza energia a disposizione di entrambi.

3. Felicità dell'animale **(stato d'animo)**

3.1. Ogni animale ha un parametro di *happiness* con valore iniziale differente per ognuno, se raggiunge 0 l'animale si ferma in preda alla depressione e non si muoverà finchè non si rialza questo parametro.

3.2. La *happiness* viene influenzata positivamente ogni volta che l'animale si nutre e decresce sia naturalmente con il passare del tempo sia quando l'animale capita sopra una cella "sporca" contenente dei bisogni.

3.3. L'utente può influenzare positivamente il parametro di un singolo animale utilizzando l'apposito tasto per accudirlo (giocare).

3.4. L'happiness viene aumentata di un pò quando l'animale ne incontra uno della sua stessa specie, rimane neutra se ne incontra altri.

4. Interazione dell'utente

4.1. **Tasto "Food":** il recinto è inizialmente coperto di cibo (celle verdi), con il passaggio degli animali questo viene consumato (celle marroni), è compito dell'utente ripristinare periodicamente il cibo per evitare di far morire di fame i propri animali.

4.2. **Tasto "Clean":** il recinto viene periodicamente sporcato dai bisogni degli animali, è possibile ripulirlo con l'apposito tasto, da notare il fatto che il cibo non viene ripristinato sopra le celle sporche del recinto.

4.3. **Tasto "Play Animal":** consente di accudire lo specifico animale (giocarci) e quindi di incrementare la sua happiness di un certo quantitativo.

Progettazione

NetLogo ed il suo IDE:

NetLogo è un linguaggio di programmazione basato ad agenti utile a modellare fenomeni complessi in un ambiente semplificato che fornisce al programmatore gli strumenti utili anche per sviluppare un'interfaccia grafica in grado sia di mostrare i dati di elaborazione sia di fornire componenti per interagire con l'utente. NetLogo utilizza gli agenti sotto una particolare declinazione logica dividendoli in *turtles, patches* ed aggiungendo *l'observer* (l'utente) come entità esterna che osserva ed influenza il sistema. Le patches sono immobili e vengono dispiegate come una griglia bidimensionale dotata di coordinate cartesiane intere, le turtles si muovono all'interno del sistema formato dalle patches ed hanno sia coordinate (decimali) sia un orientamento (direzione). Sotto l'aspetto tecnico NetLogo è in grado di far cooperare centinaia o migliaia di agenti tra loro per raggiungere l'obiettivo di simulazione, tutti gli agenti vengono eseguiti in parallelo ed indipendentemente su thread differenti, il che diventa uno dei diversi punti di forza insieme alla capacità di mostrare un risultato plottato su grafici che tengono traccia del cambiamento dello stato del sistema in funzione del tempo. Il linguaggio è dotato di funzioni, primitive, procedure e variabili con differenti livelli di scope (globali, di turtle&patch oppure locali) e grazie alla presenza di un IDE dedicato è possibile modellare simulatori di ogni genere. La flessibilità di questo linguaggio gli permette infatti di spaziare liberamente nei campi applicativi e problemi ben risolvibili mediante il paradigma multi-agente. Il lavoro svolto prende in considerazione un sistema formato da un certo numero di *patches*, mappate come le singole unità "pixel" che compongono il recinto degli animali e delle istanze specifiche di *turtles* istanziate con parametri differenti; anche l'utente ha una sua importanza in quanto gli è richiesto di cooperare insieme alla logica di gioco per portare avanti la simulazione correttamente.

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 avere una esecuzione "one time" oppure lanciare una stessa esecuzione di codice finchè non vengono ripremuti (pressed). Il centro di comando invece prende posto nella parte bassa dell'interfaccia ed è possibile inserirvi comandi per andare ad interagire in maniera programmatica sul programma in esecuzione.
- 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.

Architettura del Progetto:

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 (animali) che popolano il recinto virtuale, sia il livello macro del sistema come insieme delle sue interazioni sociali. Data la natura del tema di carattere "gaming" è stata scelto un target di simulazione che richiedesse l'interazione costante con l'utente per poter integrare meglio il concetto di osservatore che valuta lo stato di vita delle entità ed è in grado di influenzarle direttamente con delle azioni sociali. Le due caratteristiche principali tenute in considerazione durante la fase di modellazione sono state l'energia e la felicità: entrambe descrivono lo stato corrente dell'entità in gioco e vengono usate come metro per le valutazioni da parte dell'utente. Come nella vita reale l'energia per l'animale rappresenta un suo bisogno fisiologico di cibo in funzione del movimento autonomo compiuto all'interno dell'area del recinto; la felicità invece rappresenta lo stato d'animo che può essere influenzato da molteplici fattori, primo tra tutti l'energia stessa. Con l'inserimento della felicità come parametro si è deciso di aggiungere come obiettivo visibile anche tutta l'analisi dell'interazione sociale che intercorre tra uomo-animale ed animale-animale. Il parametro di felicità viene infatti modificato ad ogni incontro tra due animali, dal contatto può scaturire una reazione positiva o negativa in base al sesso, la razza di appartenenza ed infine aumenterà con l'interazione da parte del giocatore.

Diagramma delle Classi:

Tutti gli animali presenti nel progetto (*sheep, mutton, cow, bull*) derivano dalla classe agente base delle *turtles* di netlogo (quì rappresentata da Animal), nel codice è stata fatta una personalizzazione individuale di questi 4 agenti per poter avere accesso alle loro variabili di *energy* e *happiness* individualmente, così da poterle rappresentare nei grafici. La variabile *countdown-poo* serve a regolare l'intervallo dei bisogni dell'animale in funzione del cibo assunto mentre *countdown-happiness* è il tasso con cui decresce la felicità in funzione del tempo. I quattro metodi in figura descrivono i principali comportamenti autonomi di cui sono capaci le entità del nostro sistema:

- *move:* gestisce il movimento dei singoli animali all'interno del recinto ed è impostato con la stessa velocità in quanto valutata all'interno di uno spazio chiuso di piccole dimensioni.
- *eat-grass:* gestisce l'assunzione di cibo da parte degli animali, quando è disponibile cibo a terra e l'animale non è al massimo della sua energia lo assume, convertendo la *patch* a terra da verde a marrone. Se la patch a terra è nera invece triggera l'aggiornamento della felicità.
- *update-happiness:* tiene conto del valore di felicità attuale e lo aggiorna ad ogni tick periodico del simulatore, oppure a seguito di una interazione sociale negativa o positiva. Un valore di felicità molto basso inibisce il movimento.
- *death:* gestisce la morte di un agente per mancanza di energia viene gestita da questo metodo che fa scomparire l'animale dal recinto ed azzera i dati nei suoi grafici.
- *reproduce-animal:* è la funzione che permette lo spawn di un nuovo agente animale quando, all'incontro di una coppia compatibile, ci sono sufficienti energia e felicità.

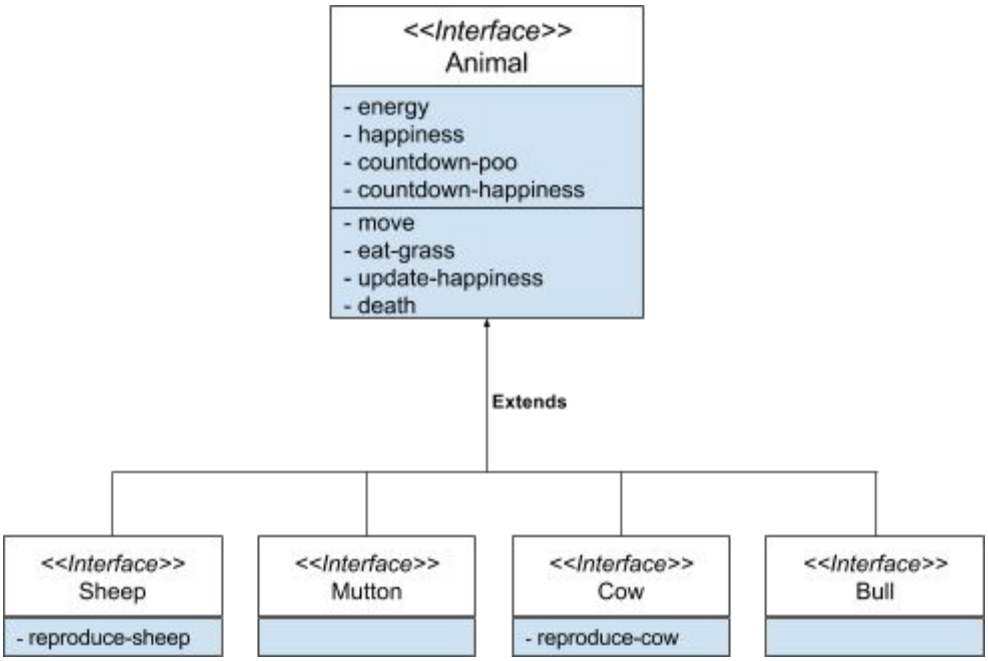
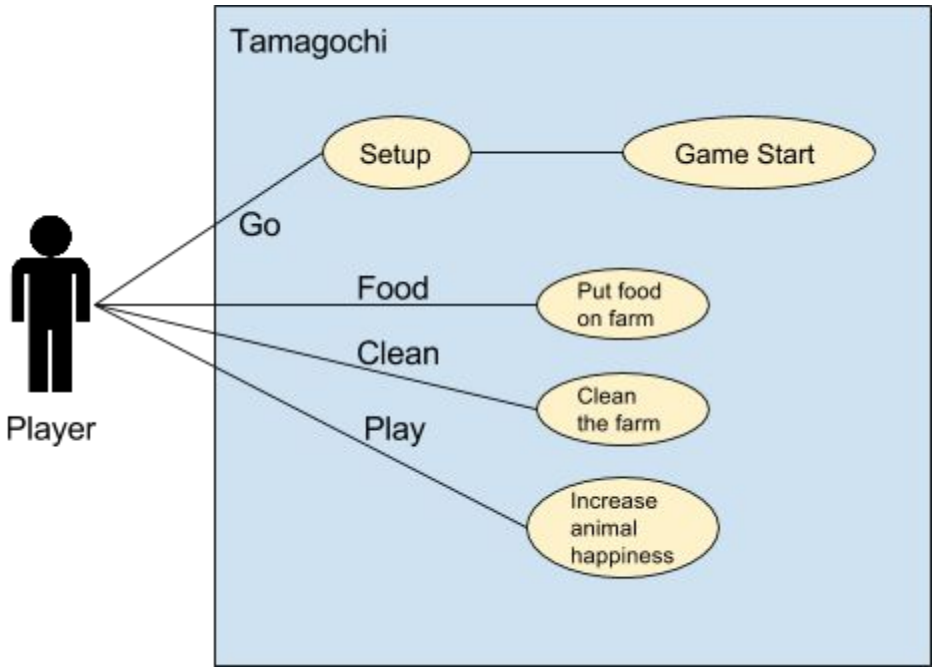


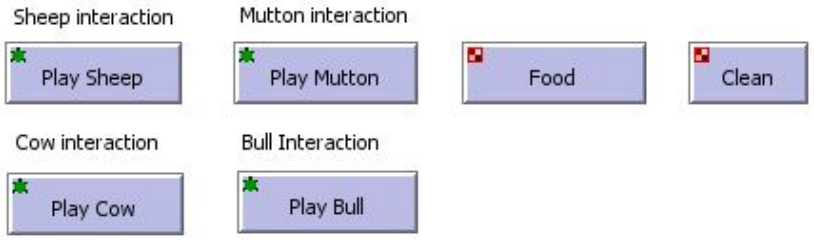
Diagramma dei Casi d'Uso:

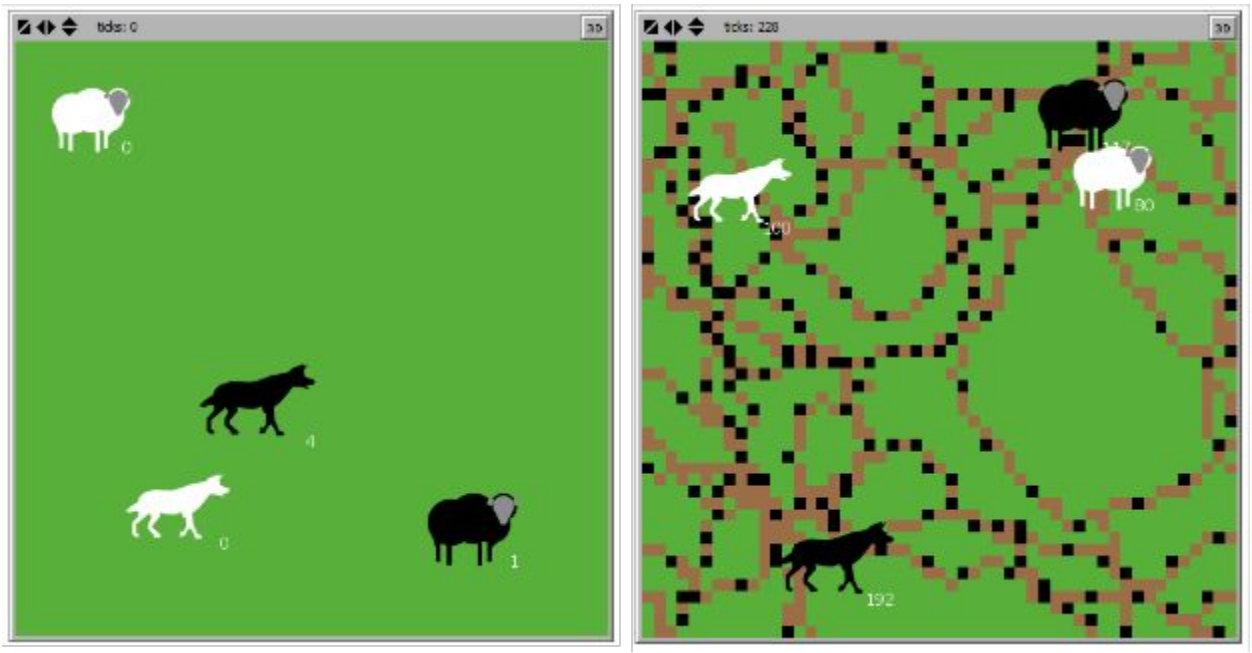
Il diagramma dei casi d'uso per l'utente, come precedentemente anticipo, modella le interazioni che questo ha nei confronti del sistema. Il cuore di queste interazioni sono le componenti utilizzate nell'interfaccia grafica che si differenziano per il tipo di effetto che hanno sul sistema quando vengono impiegate.



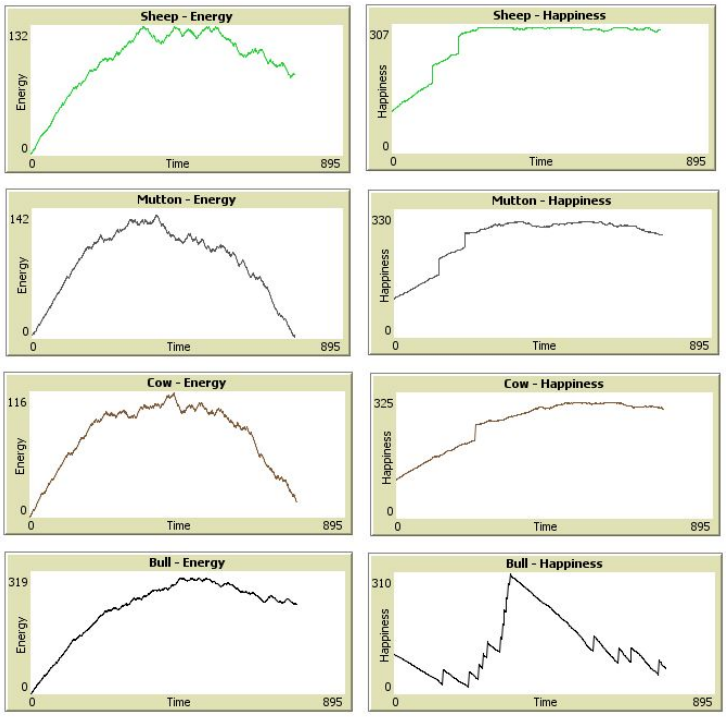
Elementi grafici:

I sei bottoni rappresentati in figura sono i *controls* principali dell'utente e servono a lanciare del codice one-time che influenza le turtles (nel caso di iconcina verde) oppure le patches (iconcina rossa) rispettivamente andando a giocare con l'animale in questione oppure ripristinando cibo e pulendo il recinto. Allo stato iniziale infatti l'interfaccia del sistema si presenta con i 4 animali posizionati nel recinto e tutte le *patches* di colore verde (cioè con cibo a terra). Dopo diversi tick gli animali progrediscono nel loro movimento e mangiano il cibo a terra lasciando bisogni ad intervalli più o meno regolari.

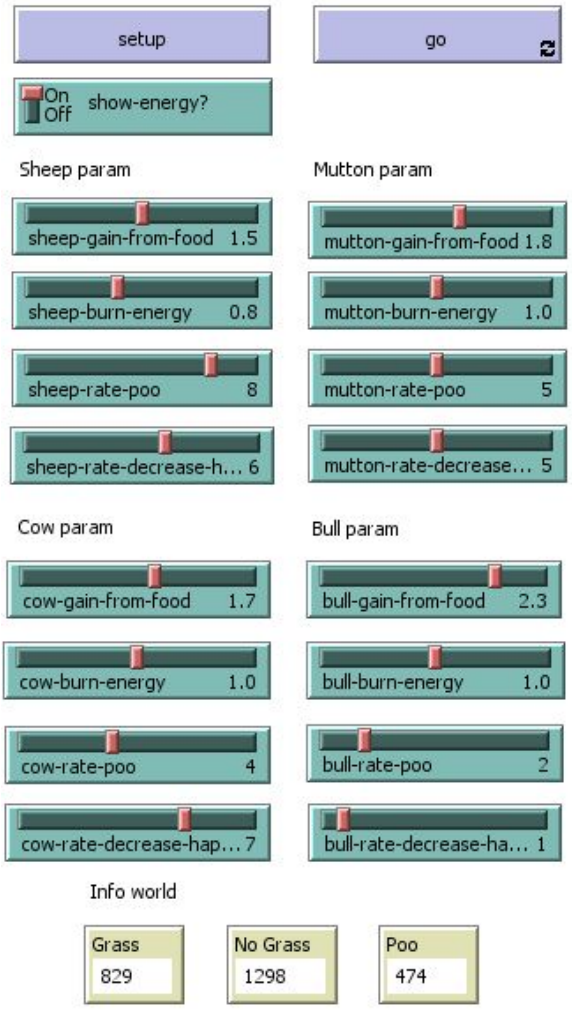




A questo punto è possibile utilizzare il tasto *Clean* per ripulire i quadrati neri oppure il tasto *Food* per ripristinare il cibo sopra i quadrati marroni, il valore indicato al pedice dell'animale rappresenta la sua energia attuale. Durante l'esecuzione è possibile monitorare lo stato delle variabili attraverso i monitor ed i plot rappresentati di seguito.



Questi plots graficano a video in tempo reale come variano energia e felicità dei singoli animali nel tempo, in una situazione tipica si avrà un accrescimento iniziale dei valori poichè nel recinto è disponibile molto cibo. Con il passare del tempo però la scarsità di cibo e la sporcizia influiscono sull'umore degli animali che quindi si vedono costretti a dover fare più strada per trovare del cibo, l'intervento dell'utente è cruciale quindi per scongiurare la morte degli animali oppure per sollevare il loro morale tramite il gioco. La situazione può deteriorare ulteriormente con la nascita di nuovi animali: il cibo infatti tenderà a scarseggiare con una frequenza maggiore e la popolazione richiede una cura maggiore rispetto ai pochi individui. Questa situazione tende infatti a riflettere la realtà, dove la supervisione di un solo individuo può divenire da necessaria ad insufficiente. L'evoluzione di questi numerosi fattori è legata comunque a molte variabili che possono essere personalizzate nell'interfaccia principale, nella zona dei settings.



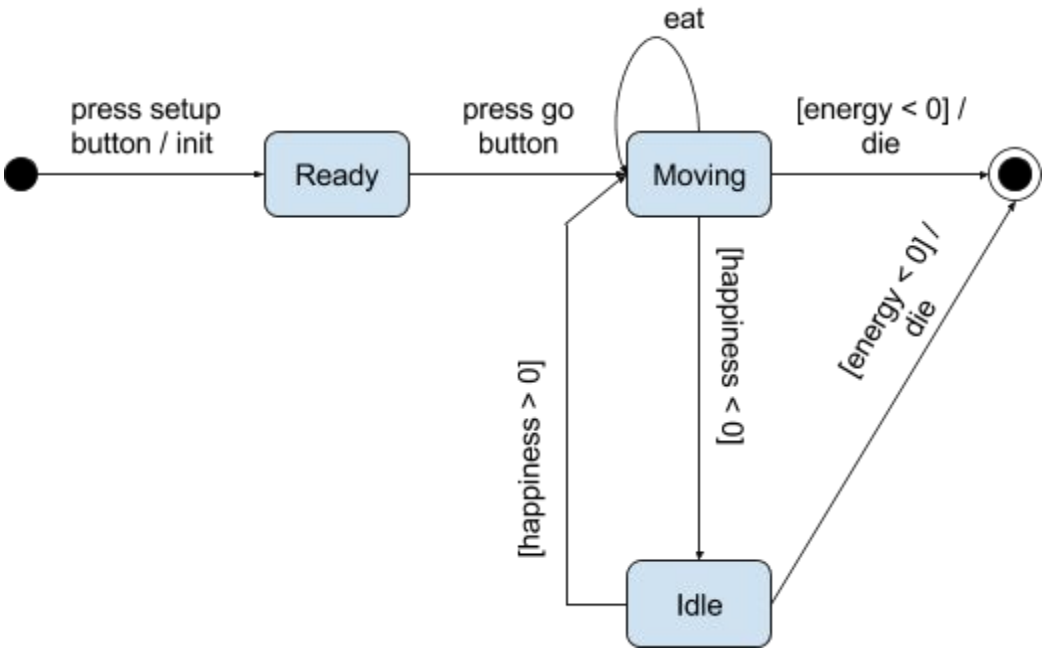
I vari slider nella sezione di sinistra servono proprio per modificare quei parametri che caratterizzano indivdualmente gli animali: vi sono quattro attributi fondamentali che regolano la quantità di cibo assunta per “mangiata”, l’energia bruciata per ogni passo, l’intervallo dei bisogni dell’animale ed il tasso di decrescita dell’umore. Insieme a questi sono presenti il button di setup iniziale e quello per l’avvio del simulatore. Nella parte inferiore invece ci sono tre monitors che tengono traccia dei dati globali riassuntivi (aggregati) della simulazione corrente.

Autonomia dei componenti:

L'analisi dell'autonomia, delle entità fondamentali che compongono il nostro simulatore, serve a mettere in luce quelle peculiarità che un agente deve avere per essere considerato autonomo. Come già affermato precedentemente, NetLogo mette a disposizione il paradigma ad agenti implementando due differenti tipi di agenti: le *turtles* e le *patches*. Nello specifico le turtles vengono rappresentate dagli *animals* mentre le patches dal *grass*. La prima chiara differenza è che mentre i primi agenti sono caratterizzati da un movimento dinamico, gli altri sono statici ed immobili valutando in qualche modo gli eventi che accadono sopra di essi e reagendo agli stimoli che vengono generati dagli animali stessi. Tutti gli agenti del primo e secondo tipo interagiscono solo tramite questi stimoli mentre gli *animals* interagiscono anche tra di loro nel momento in cui entrano in contatto. Ogni agente incapsula comunque il proprio controllo attraverso un thread dedicato che gli viene assegnato da NetLogo, mentre lo stato individuale delle variabili contenute da queste entità ne caratterizza la situazione attuale. Il MAS di *Farmagochi* può essere visto come un'aggregazione di più entità semplici che dal loro punto di vista perseguono un proprio scopo locale atto a favorire l'evoluzione di un sistema più complesso di cui solo l'utente è osservatore finale. Per descrivere meglio l'autonomia dal punto di vista delle entità impiegate nel progetto, verrà illustrato il diagramma degli stati di entrambe.

Animal (turtle):

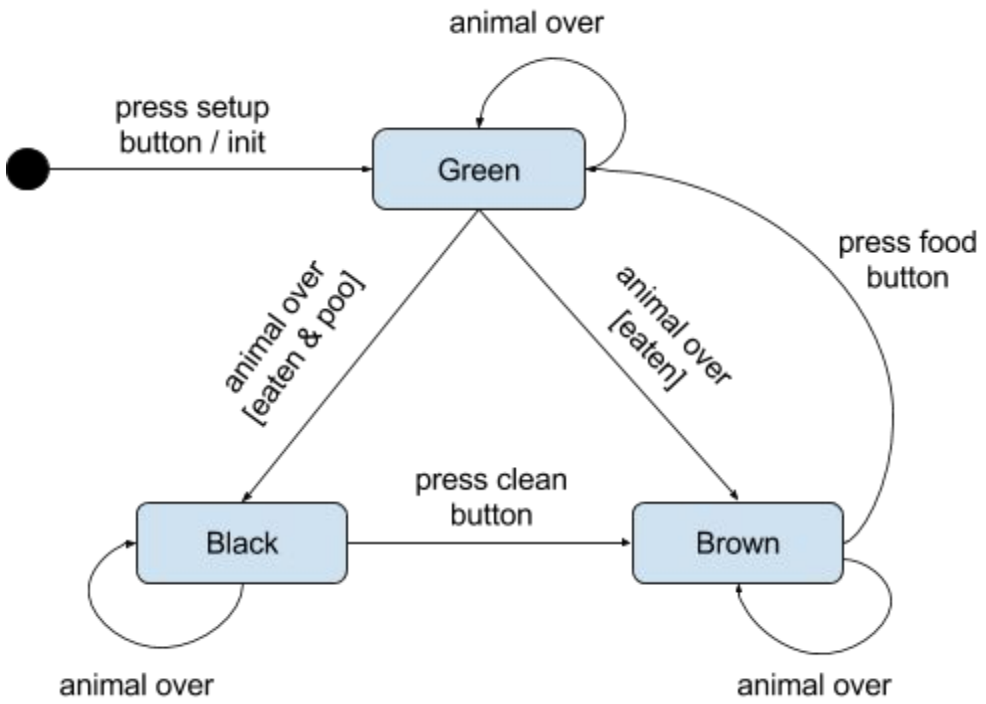
L'animal estende la turtle NetLogo e tutti i suoi stati sono mostrati nel diagramma sottostante. Dopo una fase di setup iniziale l'animal diventa ready o pronto (con le sue variabili interne inizializzate), tramite il tasto "Go" l'agente inizia il suo comportamento dinamico di movimento all'interno del recinto. Mentre è in questo stato l'agente può compiere diverse azioni che mirano tutte quante a cercare di rimanere in vita (il suo "goal"), l'animal infatti mangerà ogni qual volta troverà del cibo a terra (se la sua energia non è al massimo) e contemporaneamente ogni suo spostamento costerà un certo quantitativo di energia. Con il passare del tempo anche la felicità dell'animale calerà, questa può essere influenzata però sia dall'apporto con l'utente, che dall'interazione casuale con altri animali. Il nostro animal dunque presenta tutte le caratteristiche per poter distinguersi come agente proattivo: svolge un suo compito interno completamente autonomo, ovvero portare avanti la sua vita (e riprodursi quando possibile!), per fare questo però deve interagire con il mondo circostante, e valutare le variabili che vanno ad influenzare per forza i suoi due parametri principali di energia e felicità. L'animal autonomamente cercherà sempre di mangiare, ma la situazione di cambiamento costante all'interno del recinto influenzerà la sua capacità di muoversi e di sopravvivere. In questa fase può sopraggiungere l'aiuto dell'osservatore umano (utente) il cui scopo rimane quello di supervisionare l'andamento della sua "fattoria" virtuale e di cercare di sistemare la situazione quando questa diventa insostenibile (troppo poco cibo per la richiesta necessaria oppure morale degli animali a terra).



Grass (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 il colore verde che contraddistingue la presenza di cibo nell'area di recinto relativa alla patch stessa. In seguito ad alcuni eventi che possono scaturire dall'*Animal* oppure dall'utente stesso viene cambiato lo stato interno (colore marrone o nero a seconda del tipo di evento). Come è possibile notare dall'immagine, nel nostro contesto il *grass* si identifica più come un *Attore* che come agente in quanto i suoi cambiamenti di stato sono sempre scatenati da un avvenimento esterno, determinando un comportamento reattivo, contrariamente all'*Animal* analizzato precedentemente. L'insieme dei vari stati degli attori grass però va a formare la trama di *background* del nostro sistema, ovvero la base stessa del recinto nel quale gli agenti principali si muoveranno.

Anche se l'attore non ha una interazione diretta con il mondo circostante o con gli altri agenti, esso espone la sua proprietà principale in maniera chiara all'osservatore esterno in modo tale che questo possa carpirne la situazione in ogni momento. Questa forma di comunicazione non diretta ma "passiva" rende l'attore capace di un certo livello di autonomia in quanto manifesta un cambiamento di stato ed indirettamente il bisogno di attenzione da parte dell'utente (che deve prima ripulire una cella per poterla rifornire di cibo).



Risultati e conclusioni

Il lavoro svolto con Farmagochi ci ha permesso di mettere in pratica alcuni degli aspetti sull'autonomia visti a lezione e legati alla programmazione ad agenti. Tramite l'implementazione di un semplice MAS che permette ad alcune entità di interagire tra di loro e con l'ambiente circostante siamo riusciti a modellare diversi gradi di autonomia e di rappresentarli all'interno delle entità che sono state create. Ci possiamo ritenere soddisfatti dell'impiego della tecnologia di NetLogo che si è dimostrata un facile vettore di apprendimento di questi concetti e che ci ha permesso concentrarci di più sulle nozioni teoriche e pratiche dell'autonomia senza dover curare la parte fisica di gestione degli agenti (thread, risorse e metodi di interazione), spesso critica in progetti simili. La scelta del tema ludico ci ha permesso inoltre di spaziare in un tema come quello della *life-simulaton* in maniera più agile, potendo così adottare un modello semplificato di "realismo" che viene ben rappresentato dalla programmazione ad agenti. Tramite il gioco si è raggiunto l'obiettivo di monitorare lo stato dei singoli agenti che compongono il MAS e di gestirli efficacemente (laddove necessario ed intervenendo in maniera esterna alla loro autonomia individuale). La personalizzazione dei parametri iniziali di ogni singolo animal inoltre dà un valore aggiunto alla simulazione in quanto permette di diversificare ogni singolo lancio del gioco e di valutare ogni outcome derivante dalla combinazione dei parametri.