# Roadmap-based flocking behaviour

Mattia Crispino

## 1 Introduzione

Alla fine degli anni '80, Craig W. Reynolds ha introdotto il concetto di *boid* con l'obiettivo di offrire un modello comportamentale distribuito efficiente per la simulazione di movimenti complessi nella animazioni; il *flock* che viene simulato è l'elaborazione di un sistema di particelle in cui il singolo uccello (*boid*) è la particella stessa.
La caratteristica peculiare di tale modello risiede nel fatto che ogni *boid* è implementato come un attore indipendente che si muove nello spazio grazie alle percezioni che questo ha sull'ambiente, alle leggi della fisica che governano il suo moto e da una serie di comportamenti base; il risultato finale è un un movimento coordinato dell'intero *flock*, ottenuto dalle interazioni tra i singoli *boid*.
L'obiettivo del progetto è realizzare una simulazione del comportamento dei boid per la corretta esecuzione dei task che si basano sull'utilizzo di una roadmap quali *homing*, *covering*, *goal searching* e infine *sheperding*.

## 2 Architettura del sistema

Il sistema, realizzato in Java, si sviluppa su un'architettura multi-thread e si basa sull'implementazione del framework ActoDES il quale consente di sviluppare un'applicazione distribuita basata sull'interazione tra gli attori.
Un attore è un oggetto autonomo che può interagire con gli altri attraverso uno scambio di messaggi asincrono. L'attore, in risposta ad un messaggio, può inviare dei messaggi agli altri attori o a sè stesso, creare altri attori, adottare un *behaviour* differente, terminare il proprio processo o aggiornare il proprio stato locale.
Il sistema si compone di diversi package:

1. **app**: suddiviso a sua volta nei package **view** e **controller**, contiene le classi principali per la definizione e gestione dell'interfaccia grafica
2. **flocking**: contiene le classi che definiscono il comportamento dei *boid* e dell'*initiator*
3. **interaction**: contiene le classi che definiscono le tipologie di messaggi che vengono scambiati durante le interazioni
4. **resources**: contiene il file di configurazione del sistema

## 2.1 Message Pattern

Ogni attore può adottare differenti comportamenti e ognuno di questi ha il compito di processare i messaggi in entrata attraverso dei *message handler*; ciascuno di questi può processare solo messaggi che soddisfano un *message pattern* specifico.

La tabella sottostante fornisce una breve descrizione dei vari pattern di messaggi che vengono scambiati nel sistema.

| ***Message pattern*** | ***Descrizione*** |
|---|---|
| INFO | Identifica il messaggio che contiene le caratteristiche del Boid |
| CONTINUE | Identifica il messaggio che consente al Boid di continuare la sua esecuzione e procedere col round successivo |
| STOP | Identifica il messaggio che interrompe l'esecuzione del Boid |
| UPDATE_POSITION | Identifica il messaggio che consente di comunicare la nuova posizione del Boid nello spazio |
| GOAL_SEARCHING | Identifica il messaggio che consente al Boid di aggiornare la posizione del goal |
| COVERING_PATT | Identifica il messaggio che include l'edge il cui peso va aggiornato |
| COVERING_STOP | Identifica il messaggio che consente di interrompere l'esecuzione del comportamento *covering* |
| SHEPERD_INFO | Identifica il messaggio che consente al Boid di ottenere le informazioni riguardo lo sheperd |
| SHEPERD_START | Identifica il messaggio che consente di avviare il comportamento dello *sheperding* |

Tramite un file di configurazione è possibile modificare i parametri del sistema. Di seguito si mostrano questi:

| ***Parametro*** | ***Descrizione*** |
|---|---|
| boid | Identifica il numero totale di boid da creare nella simulazione |
| neighborhood.distance | Distanza entro la quale viene definito il neighborhood di un boid |
| boid.obstacle.distance | Distanza minima entro cui il boid esercita una forza di repulsione dall'ostacolo |
| goal.distance | Distanza dalla posizione di *goal* usata per verificare il numero di boid che hanno raggiunto l'obiettivo |

| | |
|---|---|
| sheperd.distance | Distanza minima entro cui il boid esercita una forza di repulsione dallo *sheperd* |
| map.width | Larghezza totale della mappa (di default ha un valore pari a 1000) |
| map.height | Altezza totale della mappa (di default ha un valore pari a 600) |
| sensor.range | Indentifica il raggio visivo di ciascun boid |
| vertices | Numero totale di nodi che compongono la *Probabilistic Roadmap (PRM)* |
| vertex.edges | Numero totale di edge di ogni nodo della PRM |
| sheperding | Flag che, se abilitato, consente di adottare la regola dello *sheperding* |
| covering | Flag che, se abilitato, consente di adottare la regola del *covering* |
| goal.searching | Flag che, se abilitato, consente di adottare la regola del *goal searching* |

# 3 Funzionamento

## 3.1 Interfaccia grafica

A partire dai dati di log generati durante l'esecuzione delle varie simulazioni è possibile fornire una evoluzione grafica dell'intera simulazione.

Siccome non è possibile definire un numero predefinito di passi di simulazione utili al completamento dei task, l'intera simulazione è suddivisa in round di esecuzione del sistema; questa viene considerata terminata quando una percentuale dell'insieme totale di boid (modificabile tramite un parametro) si trova nell'intorno della posizione finale.

L'interfaccia grafica è stata realizzata tramite la libreria *JavaFX* e consente di selezionare una tra le mappe predefinite fornite, avviare la simulazione e, infine, visualizzare il round attuale di esecuzione.

L'obiettivo di ogni mappa è quello di simulare un ambiente differente e, per questo motivo, all'interno di queste sono stati posti degli ostacoli (rappresentati da circonferenze aventi raggi differenti) che consentono ai boid di adottare comportamenti più complessi tramite la *collision avoidance*.

## 3.2 Creazione dei boid

In fase iniziale l'initiator ha il compito di creare $N$ boid sulla mappa 2D; ad ognuni di questi è associata la posizione iniziale ricoperta sulla mappa, una velocità lungo entrambi gli assi e una direzione compresa nell'intervallo [0,360] gradi.

Figura 1: Interfaccia grafica dell'applicazione

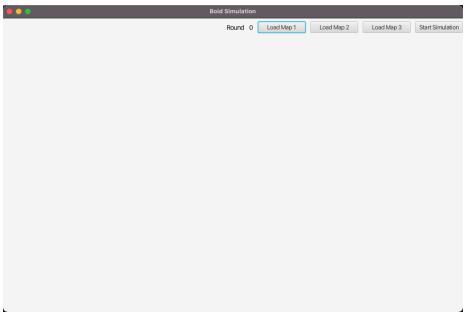
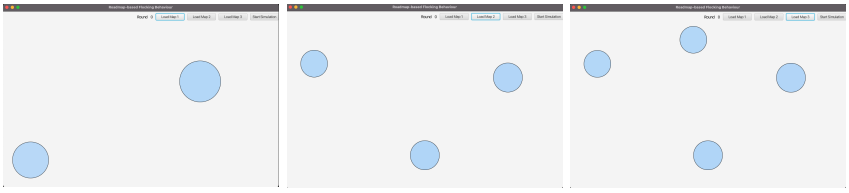



Figura 2: Mappe fornite

(a) mappa 1 (b) mappa 2 (c) mappa 3



Sia le coordinate della posizione che le componenti della velocità vengono generate in maniera casuale (all'interno di intervalli limitati) in modo tale che ogni boid, in fase iniziale, possa muoversi differentemente dagli altri.

La creazione dei boid è soggetta a due vincoli: il primo consente di non associare la stessa posizione a due o più boid mentre il secondo assicura che un boid non venga creato all'interno di un ostacolo della mappa.

Successivamente l'*initiator* invia un messaggio di *INFO* sia al boid che in broadcast all'interno dell'actor space per comunicare le caratteristiche del boid appena creato.

Nel caso in cui il flag *sheperding* sia abilitato, l'initiator provvede anche alla creazione dello *sheperd* associando a questo una velocità casuale.

## 3.3 Moto dei boid

Dopo che un boid è stato creato, in seguito alla ricezione un messaggio di *CONTINUE* inviato dall'initiator, questo inizia il proprio movimento all'interno della mappa. Terminato il moto parziale, il boid invia un messaggio di *UPDATE* all'initiator specificando la sua nuova posizione, la velocità e il round di esecuzione.
L'initiator, dopo aver ricevuto il messaggio di *UPDATE* da parte di tutti i boid, invia in broadcast il messaggio di *CONTINUE* per proseguire col round successivo.

La legge del moto per il boid stabilisce che la posizione sia definita come

$$x = x_0 + v * t$$

con $x_0$ posizione iniziale e $v$ la velocità del boid data dalla relazione

$$v = v_0 + v_1 + ... + v_n$$

con $v_0$ velocità iniziale e $v_i, i = 1, ..N$ i contributi delle regole a cui il boid è soggetto.

Le regole principali che definiscono il comportamento base di un boid sono la *cohesion*, *separation* e *alignment* (le quali si basano sulla posizione e sulla velocità dei boid che costituiscono il vicinato di quello attuale), la *collision avoidance* e, infine, l'*attraction force*.

### 3.3.1 Cohesion

Questa regola permette al boid di muoversi verso il centro del proprio vicinato in modo che ognuno rimanga unito al resto del gruppo.

Il vicinato di un boid è composto da quelli che si trovano ad una distanza minore o uguale al parametro *neighborhood.distance*.
L'implementazione di questa funzione consiste nel calcolare il centro del vicinato del boid e ritornare questo valore diviso per un fattore (il coefficiente di coesione) che consente di regolare la componente del moto verso tale centro di massa.

### 3.3.2 Separation

L'obiettivo di questa regola è far sì che i boid mantengano una distanza minima tra gli oggetti (inclusi gli altri boid) in modo tale che non entrino in collisione l'uno con l'altro.
L'implementazione di questa funzione consiste nel ricercare tutti i boid che hanno una distanza inferiore ad un valore prefissato (il coefficiente di separazione) e allontanare il boid muovendolo in senso contrario.

### 3.3.3 Alignment

Questa regola consente di adattare la velocità del boid alla velocità media del proprio vicinato.
L'implementazione è analoga a quella della coesione con la sola differenza che anzichè identificare il centro di massa vengono sommate le velocità; infine viene ritornata il vettore delle velocità diviso per il coefficiente di allineamento.

### 3.3.4 Collision Avoidance

Le regole precedenti sono utili per definire un movimento coordinato ma, volendo simulare ambienti differenti e più complessi, non sono sufficienti per gestire la presenza degli ostacoli; per questo motivo viene introdotta la regola della *collision avoidance* la quale viene implementata tramite l'adozione di una

forza repulsiva che allontana il boid dall'ostacolo e concorre nel calcolo delle componenti della velocità.

#### 3.3.5 Attraction Force

I boid, seguendo alcuni comportamenti, devono essere attratti verso determinate posizioni e quindi è necessaria la presenza di un ulteriore forza di campo, quella attrattiva; l'implementazione di questa permette di modellarne l'intensità e viene ritornata una componente della velocità utile nella ricerca della velocità finale del boid.

#### 3.3.6 Bound position

Affinchè fosse possibile mantenere i boid all'interno dell'area della mappa, è stata implementata una regola di *position bounding* la quale somma un determinato valore di offset nel caso in cui il boid, durante il proprio moto, ecceda le dimensioni della mappa.

#### 3.3.7 Limit velocity

Per simulare una situazione reale in cui un corpo non può assumere una velocità fisicamente impossibile, si è proceduto con l'implementazione di questa regola che consente di limitare, in base ad un parametro, la velocità dei boid in modo tale che questi non assumano una velocità troppo elevata.

### 3.4 PRM - Probabilistic Roadmap

Adottando le regole precedentemente definite è solamente possibile indirizzare un boid lungo la direzione dello stormo e non in una particolare direzione; per questo motivo, affinchè i boid possano muoversi in ambienti complessi, è possibile introdurre una *roadmap* la quale fornisce informazioni supplementari riguardo la topologia dell'ambiente. Una *roadmap* è un grafo composto da nodi e archi e ciascun arco identifica un percorso libero nell'ambiente.
Al fine di poter sfruttare tale informazioni addizionali, i boid necessitano di un algoritmo di *path finding*; per questo motivo, si è utilizzato il metodo *Probabilistic Roadmap (PRM)* per generare la roadmap e, successivamente, si è ricercato lo *shortest path*.
Tale PRM è stata generata, in prima fase, inserendo in posizione casuale un numero definito di nodi (dato dal parametro *vertices* del file di configurazione) e, in ultima fase, si è tentato di connettere tali nodi con degli archi nel caso non ci fossero ostacoli nel mezzo.

#### 3.4.1 Creazione dei nodi

I nodi vengono generati in posizione casuale nella mappa e la creazione di questi è soggetta ai seguenti vincoli:

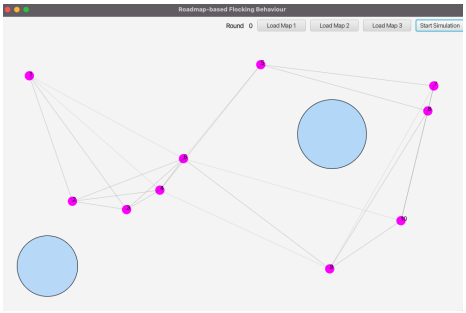
- non sono ammessi nodi duplicati nella stessa posizione
- i nodi devono avere una distanza minima tra loro
- i nodi devono avere una distanza minima da ogni ostacolo presente nella mappa

L'ultimo nodo creato viene identificato come *goal*, ovvero è la posizione finale verso cui indirizzare il moto dei boid.
Ad ogni nodo viene associato un peso proporzionale alla distanza tra questo e il *goal*, ovvero maggiore è la distanza minore è il peso; supponendo di dover creare $M$ nodi, il *goal* avrà peso massimo pari a $M$ mentre il nodo più distante avrà peso minimo pari a 1.
Infine viene effettuato un sorting dei nodi in ordine decrescente rispetto al loro peso.

### 3.4.2 Creazione degli archi

Dal parametro di configurazione *vertex.edges* è possibile definire il numero di archi in uscita da ogni nodo.
La creazione degli archi è soggetta a due vincoli: il primo consiste nel connettere un nodo solamente con quelli aventi distanza minima mentre il secondo impone la presenza di almeno un arco che connetta tale nodo con uno avente peso superiore. Quest'ultimo vincolo è stato introdotto perchè, dato che il peso dei nodi aumenta più questi sono vicini al *goal*, in questo modo si assicura che esista almeno un *path* che connetta il nodo a peso minimo col nodo a peso massimo.
In seguito alla creazione dei nodi e degli archi, questi vengono salvati in file esterni di log che saranno successivamente utilizzati per fornire l'evoluzione grafica della simulazione.

Figura 3: Nodi e archi presenti sulla mappa; i primi (rappresentati in magenta) sono caratterizzati da un valore numerico che rappresenta il peso a loro associato



### 3.4.3 Shortest path

La ricerca dello *shortest path* è fondamentale per indirizzare il moto dei boid verso l'obiettivo finale.
Per ogni nodo $i$ viene calcolata la distanza tra questo e il nodo $j$ con $j = i + 1, ..., N$ (con $N$ numero totale di nodi) e viene memorizzata la distanza minima; in questo modo la distanza viene calcolata solo per quei nodi aventi peso maggiore rispetto a quello attuale.
Successivamente viene selezionato il nodo a distanza minima nel caso in cui il nodo attuale e quello identificato siano connessi da un arco (ovvero nel caso in cui non ci siano degli ostacoli nel percorso), altrimenti si procede con l'identificazione del successivo nodo a distanza minima e si procede con le verifiche.

## 3.5 Roadmap-based behaviour

Di seguito vengono presentati le varie tipologie di comportamenti che possono essere adottati dai boid in presenza della roadmap.
Nel caso in cui i criteri per la terminazione della simulazione vengano rispettati, l'*initiator* invia in broadcast un messaggio di *KILL* per terminare la sua esecuzione e quella dei boid.

### 3.5.1 Homing rule

Questo comportamento è costituito da due modelli: il primo rappresenta il comportamento individuale di ogni boid mentre l'altro influenza il comportamento globale dell'intero *flock*.
Il comportamento individuale è soggetto alle regole base che consentono di evitare la collisione con altri boid, adattare la velocità di un boid con quella dei suoi vicini e infine stare coeso col proprio vicinato.
Il comportamento globale, invece, viene simulato introducendo delle forze presenti nella mappa che sono responsabili dell'attrazione del boid verso il *goal* e della repulsione dagli ostacoli.
Un possibile problema che si può presentare in questa situazione è il fatto che un boid possa rimanere bloccato in un minimo locale nella mappa; per questo motivo, dopo aver identificato lo *shortest path*, questo viene discretizzato nei *subgoal* che lo compongono in modo tale che quando un *subgoal* rientra nel raggio visivo del boid questo sia attratto dal *subgoal* successivo.

I due comportamenti successivi rientrano all'interno del più generale *exploring behaviour* in cui si assume che il boid abbia una conoscenza a priori della mappa e utilizzi questa per muoversi ed esplorare tutte le posizioni.
Affinchè ciò sia possibile, si usa una roadmap in cui gli archi sono caratterizzati da pesi che possono variare e quindi essere d'ausilio ai boid per indirizzare il loro movimento.

### 3.5.2 Covering

Questo comportamento ha l'obiettivo di far sì che ci sia almeno un boid che abbia visitato ogni luogo dell'ambiente.
In fase iniziale ogni edge della mappa viene inizializzato avente peso pari a 1.
Il boid, in seguito alla sua creazione e alla ricezione del messaggio di *CONTINUE* da parte dell'initiator, viene attratto dal subgoal che ha distanza minima; una volta che questo rientra nel raggio visivo del boid, quest'ultimo seleziona, tra gli edge uscenti da tale nodo, quello avente peso minimo e segue quel path che lo porta al subgoal successivo.
L'edge selezionato viene comunicato agli altri boid tramite l'invio di un messaggio di *Covering* in modo che questi possano incrementare il peso di tale arco.
Infine, viene salvato il subgoal successivo verso cui tale boid viene diretto in modo da poter interrompere l'esecuzione non appena sia presente almeno un boid che abbia visitato tutti i nodi presenti nella mappa.

### 3.5.3 Goal searching

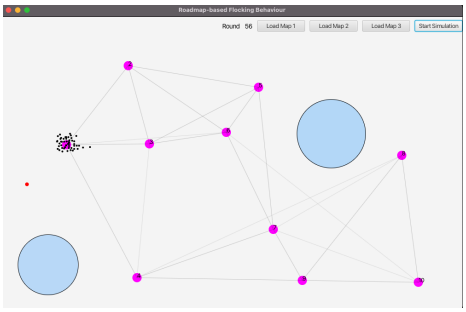
In questo comportamento i boid non conoscono la posizione del *goal* anche se possiedono una conoscenza a priori dell'ambiente.
L'obiettivo di tale *behaviour* è far si che quando un boid raggiunge una posizione nella mappa per cui il *goal* rientri nel suo raggio visivo allora questo viene comunicato agli altri boid in modo tale che questi possano raggiungere tale posizione.
L'implementazione consiste nell'adottare le regole base del moto del boid in modo che ognuno di questi possa muoversi nella mappa in accordo col suo vicinato e, non appena il *goal* possiede una distanza inferiore al raggio visivo di un boid, questo invia un messaggio di *GoalSearching* in broadcast affinchè gli altri boid possano subire una forza attrattiva ed essere indirizzati verso tale posizione.

### 3.5.4 Sheperding

In questo scenario è presente un agente esterno (lo *sheperd*) che guida i boid da una posizione iniziale fino all'obiettivo finale.
In seguito alla creazione dei boid, l'initiator procede con la creazione dello *sheperd* in una posizione casuale nell'intorno della posizione iniziale. Successivamente ogni boid subisce una forza attrattiva verso lo *starting point*, ovvero il subgoal della mappa avente peso minimo; una volta che tutti i boid hanno una distanza da tale punto inferiore ad un valore fissato, l'initiator abilita il movimento dello *sheperd* inviando un messaggio di *Sheperd.START*.
A questo punto lo *sheperd* calcola lo *shortest path* dallo starting point al goal finale e fino a quando questo non è raggiunto identifica il prossimo subgoal e indirizza il proprio moto verso questo; lo sheperd guida il moto dei boid poichè questi esercitano una forza di repulsione non appena la distanza con lo *sheperd* è inferiore al parametro di configurazione *sheperd.distance*.
Nel caso in cui il *flock* si separi, lo *sheperd* torna indietro nel percorso per recuperare i boid che si sono allontanati ricongiungendoli col resto del gruppo.

Infine, l'initiator termina la simulazione quando i boid possiedono una distanza minore o uguale al parametro di *goal.distance*.

Figura 4: Esempio della simulazione di *sheperding*: lo *sheperd* è identificato in rosso mentre in nero si notano i boid nell'intorno dello *starting point*




# 4 Installazione

L'installazione del sistema richiede la presenza di una versione di Java aggiornata, con le rispettive JDK, e del framework *ActoDES*.
Il sistema è stato realizzato con la versione *LTS* di Java 17.0.2 e JDK 8.
Infine, dal momento che viene offerta la possibilità di avere l'evoluzione grafica della simulazione, è necessario installare anche la libreria *JavaFX*.
Affinchè sia possibile simulare uno dei comportamenti presenti tra *sheperding*, *covering* e *goal searching* si deve procedere con l'abilitazione del flag corrispondente (impostando a questo il valore **true**) nel file di configurazione mentre il comportamento dell'*homing* si ottiene con tutti i precedenti flag impostati a **false**.
L'avvio del sistema è possibile tramite l'esecuzione della classe *MainFX.java*.

# 5 Risultati

Di seguito vengono presentati i risultati ottenuti dalle simulazioni coinvolgendo 50, 100 e 200 boid e i principali parametri di configurazione adottati.

| ***Parametri*** | ***Valori*** |
|---|---|
| *neighborhood.distance* | 50 |
| *boid.obstacle.distance* | 50 |
| *goal.distance* | 80 |
| *sheperd.distance* | 40 |
| *map.width* | 1000 |
| *map.height* | 600 |
| *sensor.range* | 50 |
| *vertices* | 10 |
| *vertex.edges* | 4 |

| | *mappa* | *boid* | *% boid al goal* | *round simulazione* |
|---|---|---|---|---|
| ***Homing*** | 1 | 50 | 80 | 419 |
| ***Homing*** | 1 | 100 | 80 | 295 |
| ***Homing*** | 1 | 200 | 80 | 385 |
| ***Goal Searching*** | 3 | 50 | 80 | 53 |
| ***Goal Searching*** | 3 | 100 | 80 | 41 |
| ***Goal Searching*** | 3 | 200 | 80 | 61 |
| ***Covering*** | 1 | 50 | 80 | 170 |
| ***Covering*** | 1 | 100 | 80 | 154 |
| ***Covering*** | 1 | 200 | 80 | 141 |

Analizzando i risultati ottenuti dal *Goal Searching* si può notare come, aumentando il numero di boid, questi convergano più rapidamente all'obiettivo finale ma il numero di round, considerando la simulazione con 200 boid, aumenta leggermente in quanto più attori devo essere indirizzati verso il goal.
Per quanto riguarda il *Covering*, si può notare che si ha progressiva diminuzione del numero totale di round di simulazione all'aumentare del numero di boid in quanto questi riescono a coprire un'area maggiore sulla mappa e si ha una maggiore interazione tra questi.

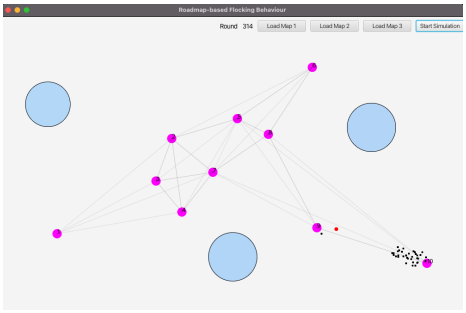
Infine, vengono mostrati i risultati ottenuti dalla simulazione dello *Sheperding* e i relativi parametri.

| ***Parametri*** | ***Valori*** |
|---|---|
| *neighborhood.distance* | 50 |
| *boid.obstacle.distance* | 30 |
| *goal.distance* | 80 |
| *sheperd.distance* | 40 |
| *map.width* | 1000 |
| *map.height* | 600 |
| *sensor.range* | 30 |
| *vertices* | 10 |
| *vertex.edges* | 4 |

| | *mappa* | *boid* | *% boid al goal* | *round simulazione* |
|---|---|---|---|---|
| ***Sheperding*** | 2 | 50 | 90 | 242 |
| ***Sheperding*** | 2 | 100 | 90 | 284 |
| ***Sheperding*** | 2 | 200 | 90 | 220 |

In questo caso si può notare come all'aumentare dei boid la simulazione termini in un numero di round che rimane pressochè costante; ciò è dovuto al maggior lavoro richiesto allo *sheperd* più la dimensione del *flock* aumenta.

Figura 5: Snapshot finale della simulazione di sheperding




I risultati presentati sono stati ottenuti adottando una forza attrattiva dall'intensità elevata in modo tale che i boid potessero convergere in un numero limitato di round; tramite la modifica dei parametri di configurazione, è possibile ottenere una simulazione caratterizzata da un maggior realismo nel moto dei boid, la quale però richiede un maggior tempo di esecuzione.