

Instituto Tecnológico de Buenos Aires

# Boids

72.25 Simulación de Sistemas

Horacio Gomez (*L* : 50825)  
Juan Pablo Orsay (*L* : 49373)

Segundo Semestre 2018

## **Resumen**

En el presente informe, se realiza un estudio del comportamiento de bandadas de agentes autopropulsados mediante la implementación y análisis del paper propuesto por Reynolds, C. W. [2].

# Índice

<b>1. Fundamentos</b>	<b>2</b>
1.1. Boids . . . . .	2
1.1.1. Vecinos . . . . .	2
1.1.2. Reglas básicas . . . . .	3
1.1.3. Reglas extendidas . . . . .	4
1.2. Polarización . . . . .	6
<b>2. Implementación</b>	<b>6</b>
2.1. Universo . . . . .	6
2.2. Grilla . . . . .	7
2.3. Entidades . . . . .	7
2.4. Reglas . . . . .	8
2.5. Simulación . . . . .	9
2.6. Optimizaciones . . . . .	9
2.6.1. Cálculo de vecinos . . . . .	9
2.6.2. Cálculo de ángulo entre vectores . . . . .	9
2.7. Visualización . . . . .	10
<b>3. Resultados</b>	<b>10</b>
3.1. Sin boids especiales . . . . .	11
3.1.1. Polarización: Alineamiento . . . . .	12
3.1.2. Polarización: Cohesión . . . . .	12
3.1.3. Polarización: Separación . . . . .	13
3.2. Con boids especiales . . . . .	14
<b>4. Conclusiones</b>	<b>17</b>

# 1. Fundamentos

## 1.1. Boids

Reynolds propuso un modelo para simular bandadas de agentes que se basa en la aplicación de una lista determinada de reglas a cada agente. Las mismas influencian como cada agente se mueve y como interactúa con lo que lo rodea.

A alto nivel, el funcionamiento del algoritmo es el siguiente:

---

### Algoritmo 1 Boids

---

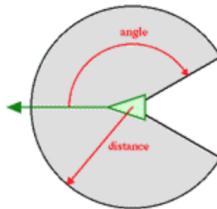
```
1: procedure Step(rules, factors, boids, dT)
2:   for boid  $\leftarrow$  boids do
3:     v = boid.velocity
4:     for rule  $\leftarrow$  rules do
5:       v += factori * rule(boid, boids)
6:     end for
7:     if magnitude(v) > MAX_SPEED then
8:       v = v/magnitude(v) * MAX_SPEED
9:     end if
10:    boid.velocity = v
11:    boid.position += boid.velocity * dT
12:  end for
13: end procedure
```

---

A cada una de las reglas se la multiplica por un factor, para hacer que su influencia sobre el sistema sea mayor o menor.

### 1.1.1. Vecinos

Dado que todas las reglas son de interacción entre *boids*, se define un radio *distance* y un ángulo *angle* dentro del cual deben de estar los otros *boids* para ser considerados vecinos:

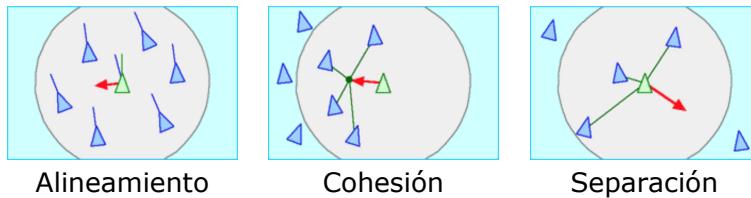


**Figura 1:** Vecinos

### 1.1.2. Reglas básicas

Reynolds propone tres reglas básicas para poder generar un comportamiento de bandada.

Unicamente los *boids* que están dentro del área de interacción afectan al *boid*.



La regla de **alineamiento** hace que cada *boid* busque alinearse con los vecinos:

---

#### Algoritmo 2 Alineamiento

---

```
1: procedure Alineamiento(boid, boids)
2:    $v_{out} = \{0, 0, 0\}$ 
3:   for neighbour  $\leftarrow$  boids do
4:      $v_{out} += \text{neighbour.velocity}$ 
5:   end for
6:    $v_{out} = v_{out}/\text{length(neighbours)}$   $\triangleright$  Velocidad promedio de vecinos
7:    $v_{out} = v_{out} - \text{boid.velocity}$ 
8:   return  $v_{out}$ 
9: end procedure
```

---

La regla de **cohesión** hace que cada *boid* busque moverse hacia el *centro de masa* de la bandada, siendo el *centro de masa*, la posición promedio de los vecinos:

---

#### Algoritmo 3 Cohesión

---

```
1: procedure Cohesión(boid, boids)
2:    $v_{out} = \{0, 0, 0\}$ 
3:   for neighbour  $\leftarrow$  boids do
4:      $v_{out} += \text{neighbour.position}$ 
5:   end for
6:    $v_{out} = v_{out}/\text{length(neighbours)}$   $\triangleright$  Posición promedio de vecinos
7:    $v_{out} = v_{out} - \text{boid.position}$ 
8:   return  $v_{out}$ 
9: end procedure
```

---

La regla de **separación** hace que cada *boid* busque estar a una cierta distancia de cada uno de sus vecinos. A diferencia de las reglas anteriores, un *boid* es considerado vecino si está a una distancia menor que la distancia de interacción que se aplica para el resto de las reglas:

---

#### **Algoritmo 4** Separación

---

```

1: procedure Separación(boid, boids)
2:    $v_{out} = \{0, 0, 0\}$ 
3:   for neighbour  $\leftarrow$  boids do
4:     if distance(neighbour, boid)  $>$  unsafeDistance then
5:       continue
6:     end if
7:      $v_{out} -= boid.position - neighbour.position$ 
8:   end for
9:    $v_{out} = v_{out}/length(neighbours)$      $\triangleright$  Posición promedio de vecinos
10:   $v_{out} = v_{out} - boid.position$ 
11:  return  $v_{out}$ 
12: end procedure

```

---

#### 1.1.3. Reglas extendidas

A demás de las reglas básicas, en el trabajo se implementaron un conjunto de reglas extras que buscan agregar complejidad al sistema.

La regla **TendencyTo** es una regla que puede interpretarse de dos maneras dependiendo del factor que se le aplique. Un factor positivo hace que las entidades se acerquen a otras de un tipo específico, mientras que un factor negativo hace que se alejen:

---

#### **Algoritmo 5** TendencyTo

---

```

1: procedure TendencyTo(boid, neighbours, interactWith)
2:    $v_{out} = \{0, 0, 0\}$ 
3:   for neighbour  $\leftarrow$  neighbours do
4:     if neighbour.type  $\neq$  interactWith then
5:       continue
6:     end if
7:      $v_{out} += (neighbour.position - boid.position)/10$ 
8:   end for
9: end procedure

```

---

La regla **Boundary** busca hacer que si un *boid* se escapa del universo sin condiciones de contorno, el mismo vuelva de una manera orgánica, sin tener saltos abruptos en la velocidad:

---

#### **Algoritmo 6** Boundary

---

```

1: procedure Boundary(boid, universe)
2:    $x_{min} \leftarrow \text{universe.metadata.boundaries.minx}$ 
3:    $x_{max} \leftarrow \text{universe.metadata.boundaries.maxx}$ 
4:    $y_{min} \leftarrow \text{universe.metadata.boundaries.miny}$ 
5:    $y_{max} \leftarrow \text{universe.metadata.boundaries.maxy}$ 
6:    $z_{min} \leftarrow \text{universe.metadata.boundaries.minz}$ 
7:    $z_{max} \leftarrow \text{universe.metadata.boundaries.maxz}$ 
8:   speed  $\leftarrow \text{CONST}$ 
9:    $v \leftarrow \{0, 0, 0\}$ 
10:  if boid.x <  $x_{min}$  then
11:     $v_x = speed$ 
12:  end if
13:  if boid.x >  $x_{max}$  then
14:     $v_x = -speed$ 
15:  end if
16:  if boid.y <  $y_{min}$  then
17:     $v_y = speed$ 
18:  end if
19:  if boid.y >  $y_{max}$  then
20:     $v_y = -speed$ 
21:  end if
22:  if boid.z <  $z_{min}$  then
23:     $v_z = speed$ 
24:  end if
25:  if boid.z >  $z_{max}$  then
26:     $v_z = -speed$ 
27:  end if
28:  return v
29: end procedure

```

---

## 1.2. Polarización

Polarización es un parámetro que permite tener un indicador sobre el estado de alineamiento de las velocidades de los *boids* dentro del universo (ignorando los depredadores). La misma adquiere valores entre 0 (desorden, cada *boid* se mueve en una dirección distinta al resto) y 1 (orden, todos los *boids* se mueven en la misma dirección).

Para definir polarización en tres dimensiones se sigue el modelo propuesto por Vicsek et al.[3] ya que aplica de igual manera para dos que para tres dimensiones. Se modifica el denominador  $Nv$  ya que cada *boid* tiene su propia rapidez:

$$v_{avg} = \frac{\sum_{i=1}^N |\mathbf{v}_i|}{N}$$

$$v_a = \frac{1}{v_{avg}} \left| \sum_{i=1}^N \mathbf{v}_i \right|$$

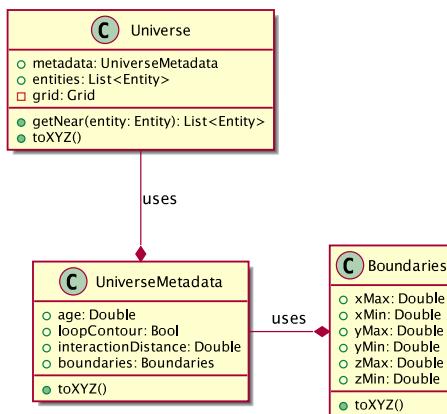
donde  $N$  es la cantidad de agentes en el universo y  $\mathbf{v}_i$  es la velocidad de cada agente.

## 2. Implementación

### 2.1. Universo

El universo utilizado para realizar la simulación es un hiperrectángulo con un largo  $W$ , alto  $H$  y profundidad  $D$ . Es posible crear simulaciones con o sin condiciones de contorno.

Una vez creado, el universo no puede ser modificado (ni sus dimensiones, ni parámetros, ni entidades), por lo que al avanzar la simulación equivale a crear fotos del universo para cada tiempo específico.



**Figura 2:** Universo

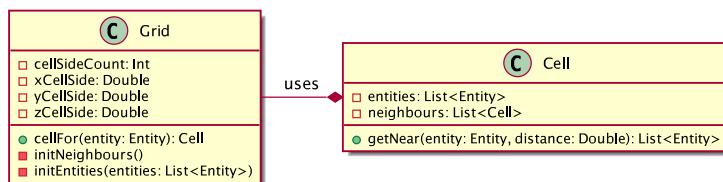
## 2.2. Grilla

*Grid* es la implementación de *Cell index method*.

Para construir una *Grid*, es necesario especificar cuantas *Cell* se deben construir por eje. A su vez es necesario especificar cual es el largo, ancho y alto de cada una de esas celdas.

Internamente, *Grid* se encarga de crear todas las *Cell* necesarias, de asignar cada *Entity* a la *Cell* que le corresponde y asignarle todos los vecinos a cada *Cell*.

De esta manera, dada una *Entity*, es posible determinar a que celda pertenece y acceder a las celdas vecinas para buscar aquellas entidades que estén a una determinada distancia sin necesidad de iterar por todas las entidades.



**Figura 3:** Grilla

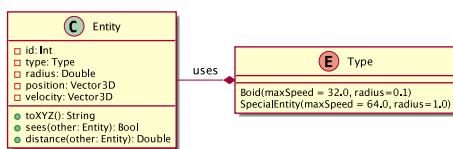
## 2.3. Entidades

Todos los agentes, no importa si son depredadores o no, son representados utilizando la clase *Entity*.

Cada entidad cuenta con las siguientes propiedades:

- **id**: Identificador de la entidad. Son únicos.
- **type**: Determina el tipo de la entidad.
- **radius**: Determina el radio de la entidad.
- **position**: Posición ( $x, y, z$ ) en el espacio.
- **velocity**: Velocidad ( $v_x, v_y, v_z$ ) en el espacio.

A su vez, cada *Entity* cuenta con una manera de poder calcular la distancia (desde los bordes) a otra *Entity* y de poder determinar si otra *Entity* está en su rango de visión.



**Figura 4:** Entidad

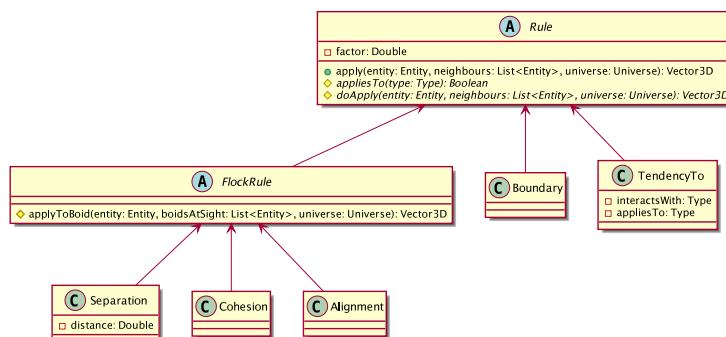
## 2.4. Reglas

Para la implementación de las reglas, se opta por tener una clase abstracta base *Rule*, la cual cuenta con un factor que se le aplica rá para aumentar o reducir su influencia. Tiene también un método público que es invocado por el algoritmo de simulación mediante el cual recibe todo lo necesario para que las implementaciones concretas puedan aplicarse.

Cada implementación debe implementar los métodos:

- *appliesTo*: Define a que tipos de entidades aplica la regla.
- *doApply*: Aplica la regla y devuelve el ajuste de velocidad.

Existe otra clase abstracta llamada *FlockRule* cuya finalidad es simplificar y agrupar aquellas reglas que apliquen únicamente para las entidades de tipo *Boid*, ignorando al resto de las entidades. Sus implementaciones conforman al set básico de reglas propuesto por Reynolds.

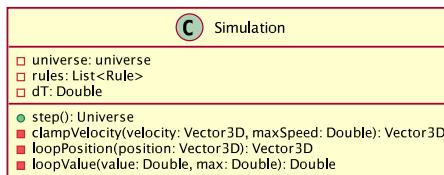


**Figura 5:** Reglas

## 2.5. Simulación

*Simulation* es la clase encargada de realizar la simulación. Internamente se ocupa de:

- Aplicar todas las reglas configuradas.
- Limitar la velocidad de todas las entidades ya que de ser únicamente por las reglas, la velocidad de cada *Entity* tiende a aumentar.
- Mover aquellas entidades que se hayan escapado del universo en caso de haber condiciones de contorno.



**Figura 6:** Simulación

## 2.6. Optimizaciones

### 2.6.1. Cálculo de vecinos

Para el cálculo de vecinos se utiliza el método llamado *Cell Index Method*. El mismo consiste en subdividir el espacio de simulación (Universo) en celdas de un determinado tamaño. Debido a que cada agente pertenece a una única celda, el método permite reducir el costo de búsqueda de vecinos de  $N^2$  a un número más cercano a  $N$ . Dado que las dimensiones del universo son más de un orden de magnitud mayores que la distancia de búsqueda, se decide fijar una cantidad de celdas determinadas para cada eje.

### 2.6.2. Cálculo de ángulo entre vectores

El cálculo del ángulo entre dos vectores era la operación que más tiempo demoraba, por lo que se implementa la función *acos* propuesta por NVidia[1].

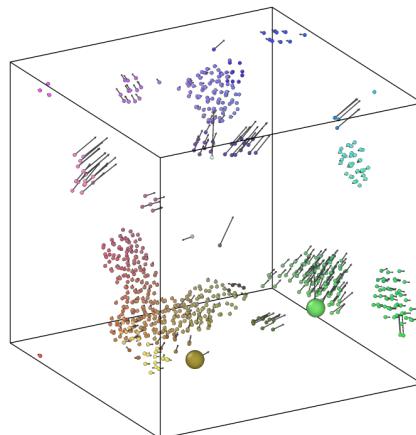
## 2.7. Visualización

Para realizar las animaciones y visualizaciones de las simulaciones se utiliza *Ovito*.

Para poder dar una idea de la posición en un espacio que cuenta con tres dimensiones, se decide utilizar la posición de cada *boid* para determinar el color que el mismo tendrá siguiendo las siguientes reglas:

- $R = 0,2 + (\text{position.x}/\text{width}) * 0,8$
- $G = 0,2 + (\text{position.y}/\text{height}) * 0,8$
- $B = 0,2 + (\text{position.z}/\text{depth}) * 0,8$

De esa manera, aquellos *boids* que se encuentren en el mismo área de la simulación tendrán un color similar, lo que permite tener una idea de la distribución de las entidades dentro del universo.



**Figura 7:** Ovito: Visualización

A además del color, se utiliza la funcionalidad de *displacement vectors* de *Ovito* para poder visualizar la dirección de movimiento de cada agente.

## 3. Resultados

Si bien la implementación realizada soporta simular con o sin condiciones de contorno, se decide que únicamente se analizarán escenarios con condiciones de contorno.

Se utilizaron 4 simulaciones para obtener los errores de cada gráfico.

### **3.1. Sin boids especiales**

Para las simulaciones sin boids especiales, se estudia el comportamiento del sistema utilizando los siguientes parámetros:

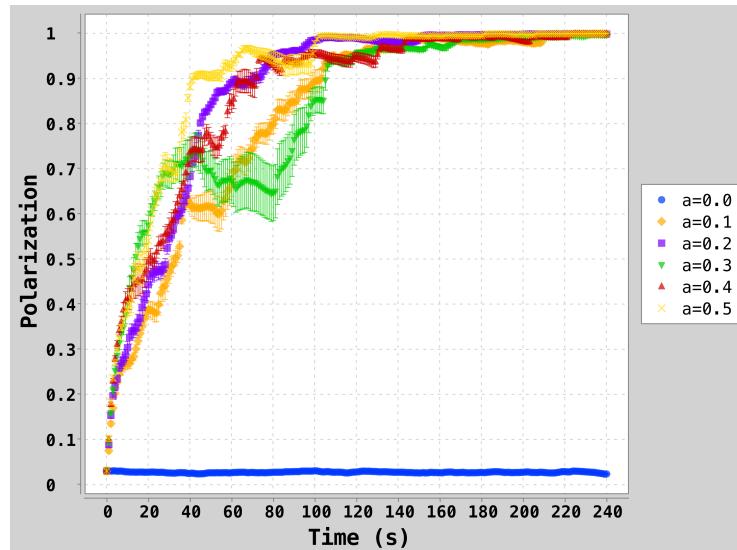
Dimensión del universo	$32m \times 32m \times 32m$
Condiciones de contorno	activadas
Número de boids	1024
Número de especiales	0
Distancia de interacción	$2m$
Radio de boids	$0,1m$
Velocidad máxima de boids	$32m/s$
Radio de especiales	$1,0m$
Velocidad máxima de especiales	$64m/s$

Los factores de multiplicación base para cada una de las reglas son los siguientes:

Alineamiento	0,25
Cohesión	0,25
Separación	0,25
Tendencia boid a especial	-0,25
tendencia especial a boid	0,25

### 3.1.1. Polarización: Alineamiento

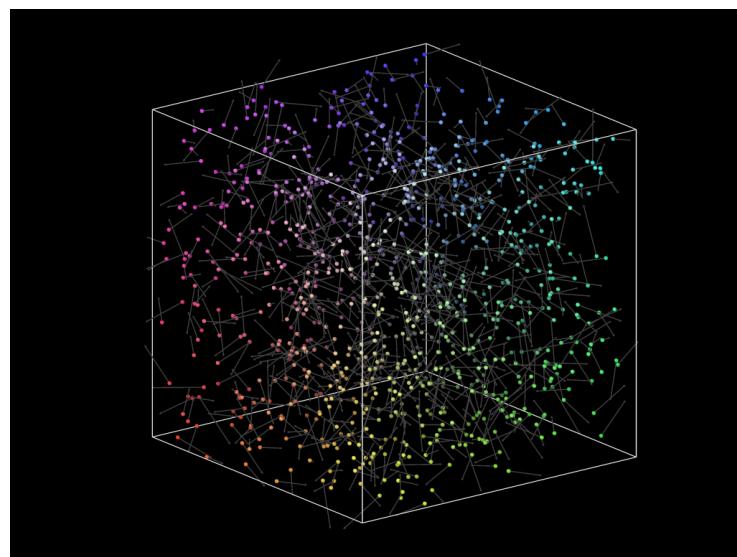
Se varía el factor de la regla de *alineamiento* para ver su impacto en la polarización del sistema.



**Figura 8:** Polarización variando factor alineamiento

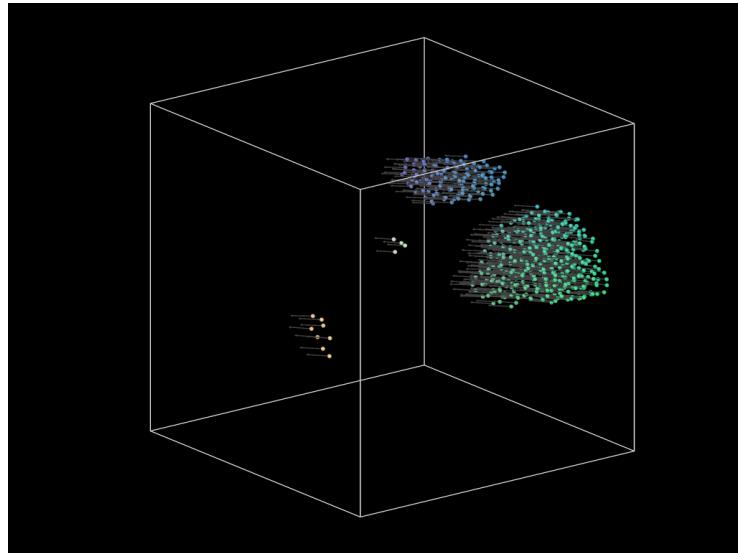
Se observa que variar el factor de la regla de *polarización* tiene uno de dos resultados:

Si el factor es 0 (regla desactivada), el sistema nunca se polariza; cualquier otro valor no afecta en mayor medida la rapidez con la que el sistema se polariza.



**Figura 9:** Comportamiento: Alineamiento ( $a = 0.0$ )

Sin alineamiento se genera caos constante.

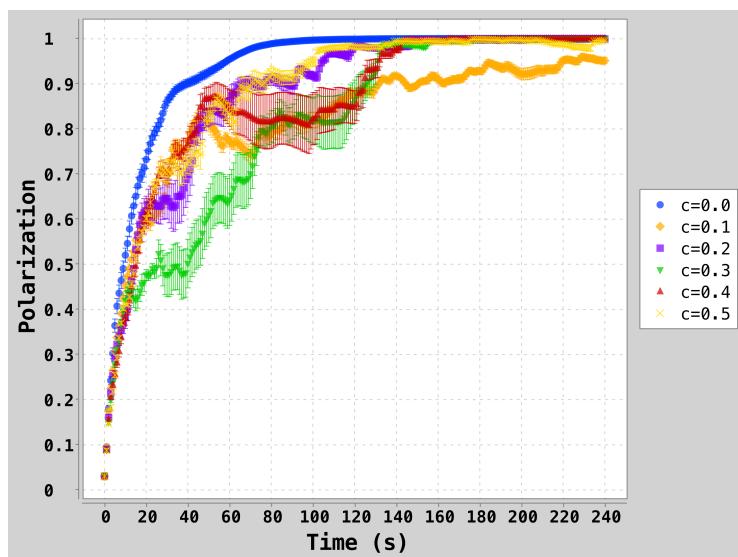


**Figura 10:** Comportamiento: Alineamiento ( $a = 0,5$ )

Con un factor de alineamiento alto, se observa que se generan bandadas de alta velocidad y polarización.

### 3.1.2. Polarización: Cohesión

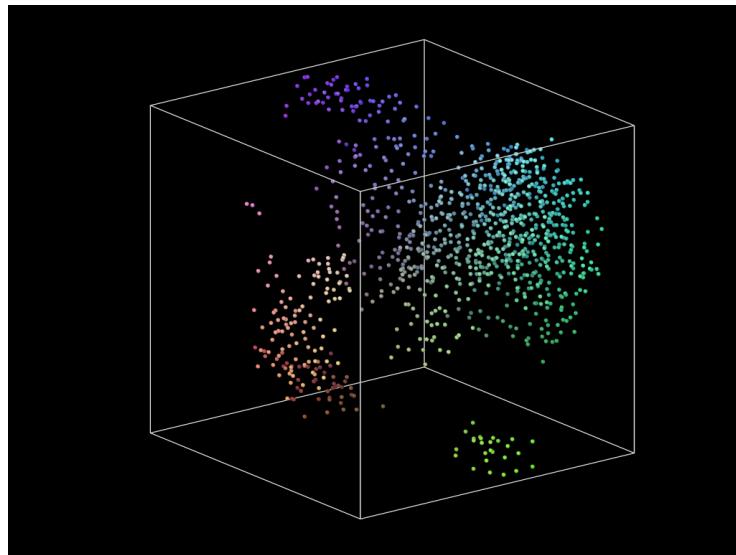
Se varía el factor de la regla de *cohesión* para ver su impacto en la polarización del sistema.



**Figura 11:** Polarización variando factor cohesión

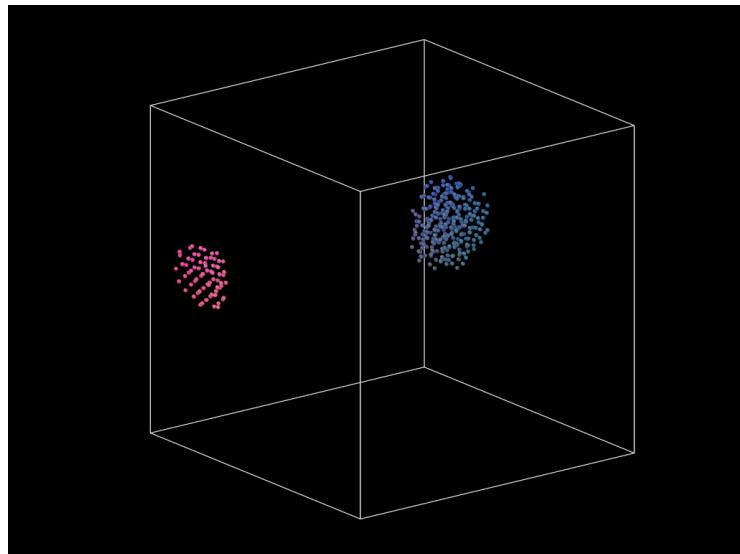
Se observa que cuanto menos peso tenga esta regla (factor teniendo a cero), el sistema tiende a polarizarse con mayor rapidez, teniendo a la más rápida aquella que tiene la regla de *cohesión* apagada.

Ésto se genera ya que cada *boid* no busca acercarse al centro de masa, por lo que como están las reglas de *polarización* y *separación* activadas, al estar cerca de otro *boid* se alinea con el mismo.



**Figura 12:** Comportamiento: Cohesión ( $c = 0,0$ )

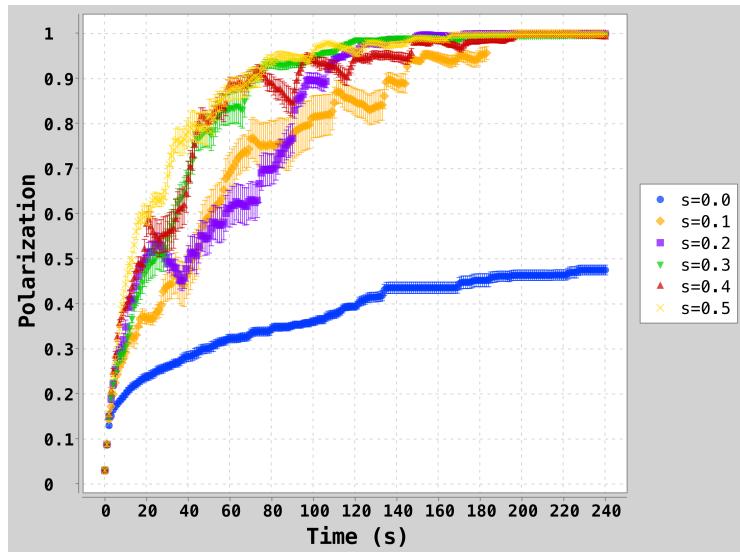
Se observa que se generan bandadas mas grandes cuando no hay cohesión.



**Figura 13:** Comportamiento: Cohesión ( $c = 0,5$ )

### 3.1.3. Polarización: Separación

Se varía el factor de la regla de *separación* para ver su impacto en la polarización del sistema.

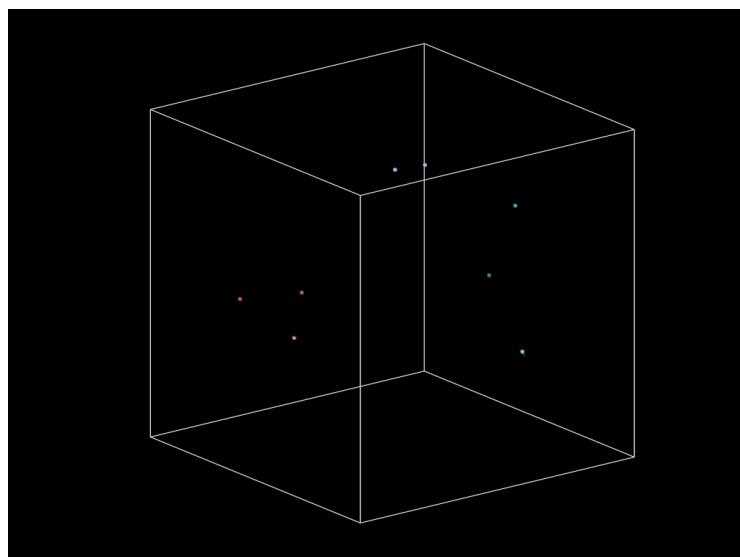


**Figura 14:** Polarización variando factor separación

Se observa que cuanto menos efecto tiene la regla de separación (factor tiendiendo a cero), más lento polariza el sistema.

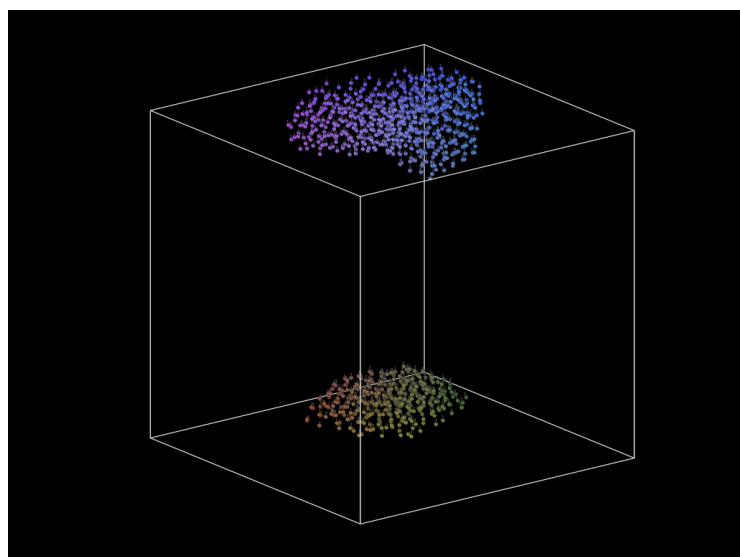
Ésto se debe a que, como cada *boid* tiende a ir al centro de masa debido a la regla de *cohesión*, se termina generando una superpo-

sición de boids, por lo que el volúmen de influencia de esa bandada se reduce a un volúmen de influencia similar al de un único *boid* y dado que el volúmen del universo es grande, la probabilidad de que dos boids se crucen es baja, por lo que toma más tiempo llegar a la polarización.



**Figura 15:** Comportamiento (frame final): Separación ( $s = 0,0$ )

Sin separación los boids se unen puntualmente por la cohesión y alineamiento entre ellos.



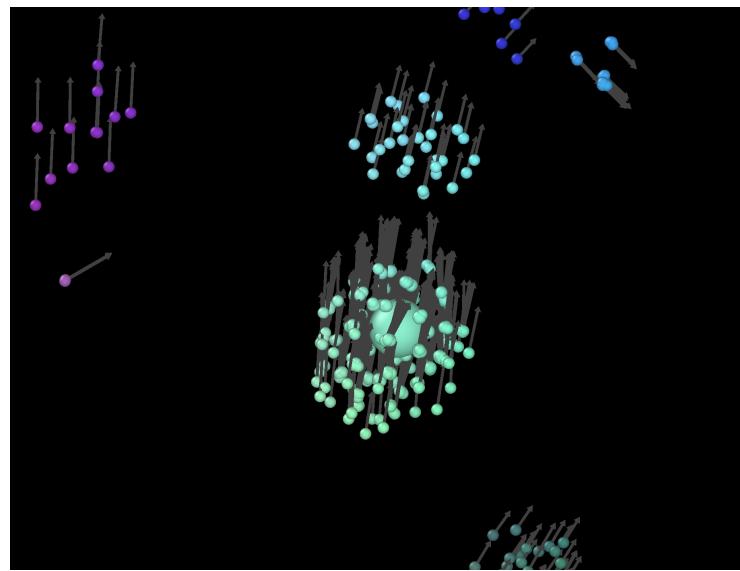
**Figura 16:** Comportamiento (frame final): Separación ( $s = 0,5$ )

Si la separación es alta, se generan bandadas grandes con mayor

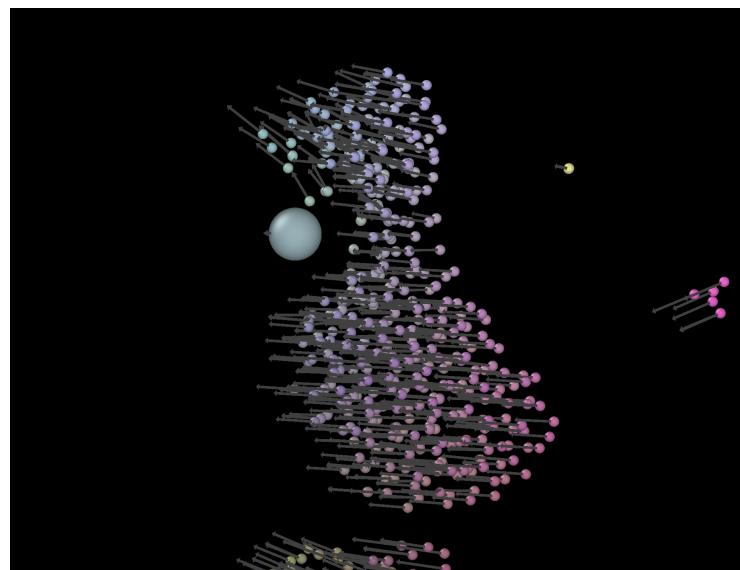
velocidad que las generadas sin cohesión.

### 3.2. Con boids especiales

Se agregan 2 boids especiales y se estudia la variación del factor de la regla "*Tendencia a*".

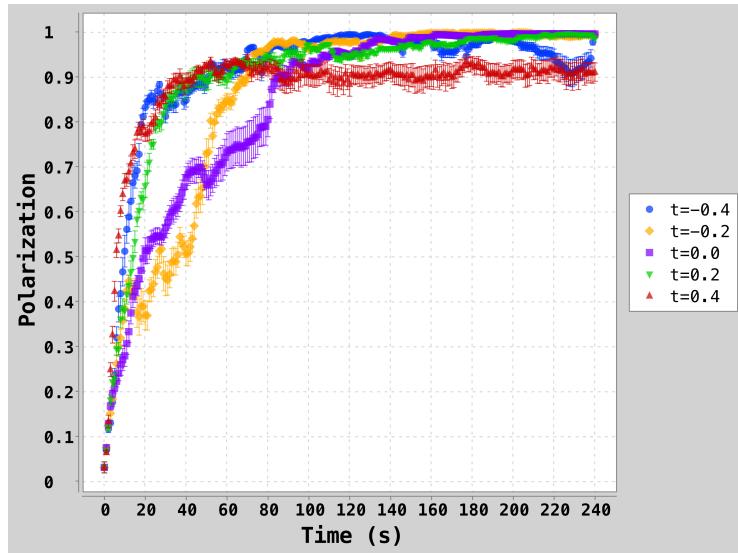


**Figura 17:** Comportamiento: Tendencia a acercarse ( $t > 0$ )

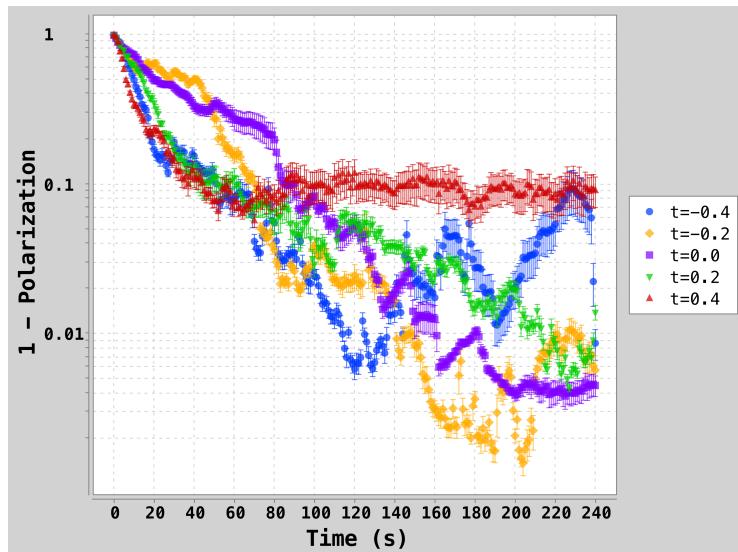


**Figura 18:** Comportamiento: Tendencia a alejarse ( $t < 0$ )

Se observa como afecta el signo del factor para la regla "Tendencia a." el comportamiento de la bandada. Con  $t > 0$  la bandada toma a la entidad especial como un líder de bandada, mientras que con  $t < 0$ , la bandada reacciona como si fuese un depredador.



**Figura 19:** Polarización: variando "Tendencia a"



**Figura 20:** 1 - Polarización: variando "Tendencia a"

Se observa que ambos extremos ( $-0.4$  y  $0.4$ ) en los que la regla de escapar o acercarse a los *boids* especiales se encuentra con mayor intensidad son los que menos logran polarizar debido a las perturbaciones introducidas por éstas entidades nuevas. Con  $t > 0$ , se logra

una polarización del sistema similar a o mejor que sin la existencia de entidades especiales debido a el comportamiento de liderazgo que se genera. Mientras que con  $t < 0$ , es posible hacer que el sistema no llegue a polarizarse.

## 4. Conclusiones

Luego del análisis de los resultados, podemos concluir que a la hora de buscar una polarización más veloz del sistema basta con desactivar la regla de *cohesión* para lograrlo. Sin embargo al hacerlo se pierde el comportamiento de bandada, haciendo que la simulación no pueda ser una buena representación de un sistema real.

La regla de *tendencia* a permite generar comportamientos complejos permitiendo simular líderes de bandada o depredadores dependiendo de qué signo se utilice en su factor.

Por otro lado, la configuración de los parámetros iniciales de la simulación como así también los factores de las reglas cambian drásticamente el funcionamiento del sistema.

Se observa que tener una alta Separación genera grandes bandadas que se movilizan más rápidamente que las generadas por baja cohesión

## Referencias

- [1] Nvidia. *Implementación rápida de acos*. url: <http://developer.download.nvidia.com/cg/acos.html>.
- [2] C. W. Reynolds. «Flocks, Herds, and Schools A Distributed Behavioral Model, in Computer Graphics». En: *SIGGRAPH '87 Conference Proceedings* (1987), págs. 25-34. url: <http://www.cs.toronto.edu/~dt/siggraph97-course/cwr87/>.
- [3] Tamás Vicsek y col. «Novel Type of Phase Transition in a System of Self-Driven Particles». En: *Phys. Rev. Lett.* 75 (6 ago. de 1995), págs. 1226-1229. doi: [10.1103/PhysRevLett.75.1226](https://doi.org/10.1103/PhysRevLett.75.1226). url: <https://link.aps.org/doi/10.1103/PhysRevLett.75.1226>.