

A large flock of birds, likely starlings, is captured in flight against a sunset sky. The birds are arranged in a dense, V-shaped formation that stretches across the upper half of the frame. The sky transitions from a pale blue at the top to a warm orange and pink near the horizon. In the background, dark silhouettes of trees and distant mountains are visible under the twilight sky.

Flocking Algorithm



Karla Valeria Perez Perez
Rogelio Robledo Moreno
Raúl Sánchez Vázquez

PSO (Optimización por cúmulo de partículas)

- Los métodos PSO se atribuyen a Kennedy, Eberhart y Shi.
- Fueron concebidos para elaborar modelos de conductas como el movimiento descrito por los organismos vivos en una bandada de aves o un banco de peces.
- El algoritmo se simplificó y se comprobó que era adecuado para problemas de optimización.

Concepto

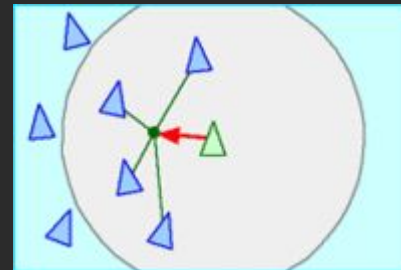
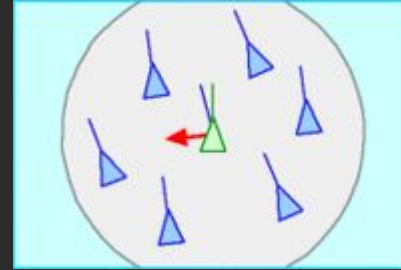
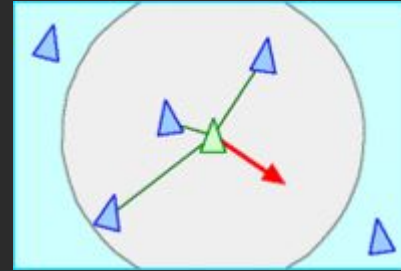
El comportamiento de bandada o flocking es el comportamiento exhibido cuando un grupo de pájaros, llamado bandada busca alimento o está volando.

Antecedentes

Boids es un programa de vida artificial , desarrollado por Craig Reynolds en 1986, que simula el comportamiento de bandadas de aves . Su artículo sobre este tema fue publicado en 1987 en las actas de la conferencia ACM SIGGRAPH .

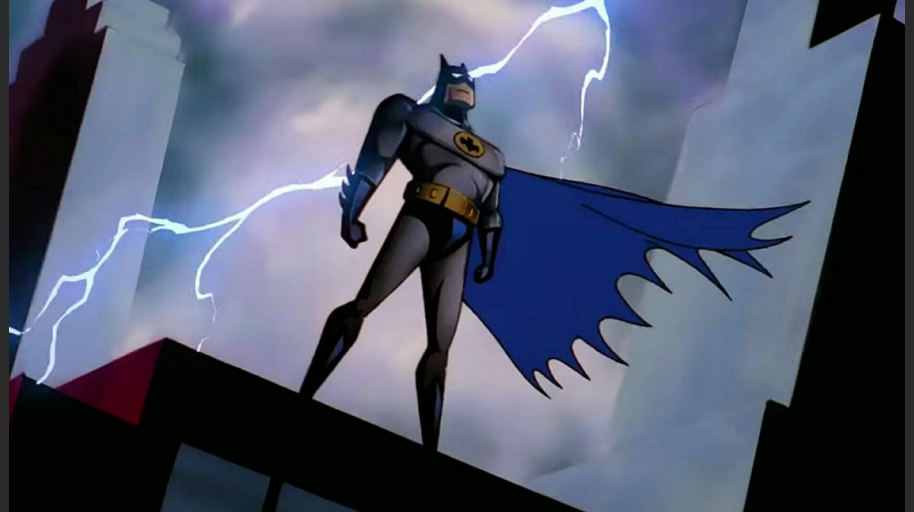
Modelo básico

1. Separación: evitar aglomeración de vecinos (repulsión de corto alcance)
2. Alineación: dirigir hacia un rubro promedio de vecinos.
3. Cohesión: dirigir hacia la posición media de los vecinos (atracción de largo alcance)



Aplicaciones

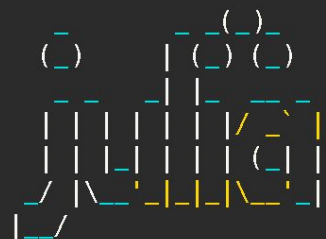
- Medio para controlar el comportamiento de vehículos aéreos no tripulados.
- Para generar multitudes que se mueven de manera más realista en animaciones.



Flocking para la bolsa

En un conjunto de acciones, las acciones líderes son aquellas que se mueven con más velocidad que las demás, bien al alza o a la baja. Si nosotros entramos en el bloque de acciones que empuja hacia arriba la bandada, y seguimos entrando y saliendo sistemáticamente en las acciones que más tiran hacia arriba (o hacia abajo) del grupo, conseguiremos estar siempre en la dirección correcta, siguiendo a la cabeza de la bandada.





A fresh approach to technical computing
Documentation: <http://docs.julialang.org>
Type "?help" for help.

```
Version 0.4.7 (2016-09-18 16:17 UTC)
Debian version 0.4.7-6+b3
x86_64-linux-gnu
```

Una buena forma de analizar esta algoritmo es realizando una implementación mediante un lenguaje optimizado para cómputo científico.

Una buena forma de analizar esta algoritmo es realizando una implementación mediante un lenguaje optimizado para cómputo científico.



```
using Agents, LinearAlgebra

mutable struct Bird <: AbstractAgent
    id::Int
    pos::NTuple{2,Float64}
    vel::NTuple{2,Float64}
    speed::Float64
    cohere_factor::Float64
    separation::Float64
    separate_factor::Float64
    match_factor::Float64
    visual_distance::Float64
end
```

Definición de estructuras centrales

Comenzamos llamando a los paquetes requeridos y definiendo un tipo de agente que representa un pájaro.



```
function initialize_model(;
    n_birds = 100,
    speed = 1.0,
    cohere_factor = 0.25,
    separation = 4.0,
    separate_factor = 0.25,
    match_factor = 0.01,
    visual_distance = 5.0,
    dims = (100, 100),
)
    space2d = ContinuousSpace(2; periodic = true, extend = dims)
    model = ABM(Bird, space2d, scheduler = random_activation)
    for _ in 1:n_birds
        vel = Tuple(rand(2) * 2 .- 1)
        add_agent!(
            model,
            vel,
            speed,
            cohere_factor,
            separation,
            separate_factor,
            match_factor,
            visual_distance,
        )
    end
    index!(model)
    return model
end
```

Definición de estructuras centrales

La función *initialize_model* genera pájaros y devuelve un objeto modelo usando valores predeterminados.



```
function agent_step!(bird, model)
    # Obtain the ids of neighbors within the bird's visual distance
    ids = space_neighbors(bird, model, bird.visual_distance)
    # Compute velocity based on rules defined above
    bird.vel =
        (
            bird.vel .+ cohere(bird, model, ids) .+ separate(bird, model, ids) .+
            match(bird, model, ids)
        ) ./ 2
    bird.vel = bird.vel ./ norm(bird.vel)
    # Move bird according to new velocity and speed
    move_agent!(bird, model, bird.speed)
end

distance(a1, a2) = sqrt(sum((a1.pos .- a2.pos) .^ 2))

get_heading(a1, a2) = a1.pos .- a2.pos
```

Definiendo el agent_step!

agent_step! es la función principal llamada para cada paso y calcula la velocidad de acuerdo con las tres reglas definidas anteriormente.



```
function cohere(bird, model, ids)
    N = max(length(ids), 1)
    birds = model.agents
    coherence = (0.0, 0.0)
    for id in ids
        coherence = coherence .+ get_heading(birds[id], bird)
    end
    return coherence ./ N .* bird.cohere_factor
end
```

Definiendo el agent_step!

cohere calcula la posición promedio de las aves vecinas, ponderada por importancia



```
function separate(bird, model, ids)
    seperation_vec = (0.0, 0.0)
    N = max(length(ids), 1)
    birds = model.agents
    for id in ids
        neighbor = birds[id]
        if distance(bird, neighbor) < bird.separation
            seperation_vec = seperation_vec .- get_heading(neighbor, bird)
        end
    end
    return seperation_vec ./ N .* bird.separate_factor
end
```

Definiendo el agent_step!

separate repele al pájaro
de los pájaros vecinos



```
function match(bird, model, ids)
    match_vector = (0.0, 0.0)
    N = max(length(ids), 1)
    birds = model.agents
    for id in ids
        match_vector = match_vector .+ birds[id].vel
    end
    return match_vector ./ N .* bird.match_factor
end
```


Definiendo el agent_step!

match calcula la trayectoria
promedio de las aves
vecinas, ponderada por
importancia.



```
n_steps = 500  
model = initialize_model()  
step!(model, agent_step!, n_steps)
```

Ejecutando el modelo



```
function bird_triangle(b::Bird)
    φ = atan(b.vel[2], b.vel[1])
    xs = [(i ∈ (0, 3) ? 2 : 1) * cos(i * 2π / 3 + φ) for i in 0:3]
    ys = [(i ∈ (0, 3) ? 2 : 1) * sin(i * 2π / 3 + φ) for i in 0:3]
    Shape(xs, ys)
end
```

Trazando los pájaros

Podemos incorporar la dirección de los pájaros al trazarlos, haciendo que la función "marcador" cree un Shape: un triángulo con la misma orientación que la velocidad del pájaro.



```
using AgentsPlots
model = initialize_model()
e = model.space.extend
anim = @animate for i in 0:100
    i > 0 && step!(model, agent_step!, 1)
    p1 = plotabm(
        model;
        am = bird_triangle,
        as = 10,
        showaxis = false,
        grid = false,
        xlims = (0, e[1]),
        ylims = (0, e[2]),
    )
    title!(p1, "step $(i)")
end
gif(anim, "flock.gif", fps = 30)
```

Animación

Referencias

1. Reynolds, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34.
2. <http://www.cs.toronto.edu/~dt/siggraph97-course/cwr87/>
3. <http://harry.me/blog/2011/02/17/neat-algorithms-flocking/>
4. <https://juliadynamics.github.io/Agents.jl/stable/examples/flock/>
5. <https://github.com/JuliaDynamics/Agents.jl/blob/master/examples/flock.jl>
6. <https://www.oreilly.com/library/view/ai-for-game/0596005555/cho4.html>
7. <https://slowinver.com/algoritmo-bandada/>