



Untitled diff

— 55 removals

220 lines

```

1 using Agents, Random
2 using StaticArrays: SVector
3 using LinearAlgebra
4
5 # Estados de los Semáforos
6 @enum LightColor green yellow red
7 @enum Streets av1 av2
8
9 normal = 0
10 left =  $\pi/2$ 
11 down =  $\pi$ 
12 right =  $3\pi/2$ 
13
14 @agent struct
15     Car(ContinuousAgent{2,Float64})
16     accelerating::Bool = true
17     street::Streets = av1
18     orientation::Float64 = normal
19 end
20 @agent struct
21     stopLight(ContinuousAgent{2,Float64})
22     status::LightColor = red
23     time_counter::Int = 0
24     street::Streets = av1
25 end
26 green_duration = 45
27 yellow_duration = 15
28
29 function
30     closest_agent_ahead(agent::Car, model,
31         ::Type{T}, radius, is_ahead_fn) where
32         {T}
33     closest = nothing
34     min_distance = Inf
35
36     for neighbor in
37         nearby_agents(agent, model, radius)

```

+ 43 additions

217 lines

```

1 using Agents, Random
2 using StaticArrays: SVector
3 using LinearAlgebra
4
5 # Estados de los Semáforos
6 @enum LightColor green yellow red
7 @enum Streets av1 av2
8
9 normal = 0
10 left =  $\pi/2$ 
11 down =  $\pi$ 
12 right =  $3\pi/2$ 
13
14 @agent struct
15     Car(ContinuousAgent{2,Float64})
16     accelerating::Bool = true
17     street::Streets = av1
18     orientation::Float64 = normal
19 end
20 @agent struct
21     stopLight(ContinuousAgent{2,Float64})
22     status::LightColor = red
23     time_counter::Int = 0
24     street::Streets = av1
25 end
26 green_duration = 45
27 yellow_duration = 15
28
29 function
30     closest_agent_ahead(agent::Car, model,
31         ::Type{T}, radius, is_ahead_fn) where
32         {T}
33     closest = nothing
34     min_distance = Inf
35
36     for neighbor in
37         nearby_agents(agent, model, radius)

```

```

34         if isa(neighbor, T) &&
            neighbor.street == agent.street &&
            is_ahead_fn(agent, neighbor, :check)
35             dist = is_ahead_fn(agent,
neighbor, :distance)
36             if dist < min_distance
37                 min_distance = dist
38                 closest = neighbor
39             end
40         end
41     end
42     return closest, min_distance
43 end
44
45 function is_car_ahead(agent, neighbor,
mode = :check)
46     if agent.street == av1
47         if mode == :check
48             return neighbor.pos[1] >
agent.pos[1] && agent.pos[2] ==
neighbor.pos[2]
49         else # :distance
50             return neighbor.pos[1] -
agent.pos[1]
51         end
52     else # av2
53         if mode == :check
54             return neighbor.pos[2] <
agent.pos[2] && agent.pos[1] ==
neighbor.pos[1]
55         else # :distance
56             return (agent.pos[2] -
neighbor.pos[2])
57         end
58     end
59 end
60
61 function is_light_ahead(agent, light,
mode = :check)
62     if agent.street == av1
63         if mode == :check
64             return light.pos[1] >
agent.pos[1]
65         else
66             return light.pos[1] -
agent.pos[1]
67         end
68     else
69         if mode == :check

```

```

34         if isa(neighbor, T) &&
            neighbor.street == agent.street &&
            is_ahead_fn(agent, neighbor, :check)
35             dist = is_ahead_fn(agent,
neighbor, :distance)
36             if dist < min_distance
37                 min_distance = dist
38                 closest = neighbor
39             end
40         end
41     end
42     return closest, min_distance
43 end
44
45 function is_car_ahead(agent, neighbor,
mode = :check)
46     if agent.street == av1
47         if mode == :check
48             return neighbor.pos[1] >
agent.pos[1] && agent.pos[2] ==
neighbor.pos[2]
49         else # :distance
50             return neighbor.pos[1] -
agent.pos[1]
51         end
52     else # av2
53         if mode == :check
54             return neighbor.pos[2] <
agent.pos[2] && agent.pos[1] ==
neighbor.pos[1]
55         else # :distance
56             return (agent.pos[2] -
neighbor.pos[2])
57         end
58     end
59 end
60
61 function is_light_ahead(agent, light,
mode = :check)
62     if agent.street == av1
63         if mode == :check
64             return light.pos[1] >
agent.pos[1]
65         else
66             return light.pos[1] -
agent.pos[1]
67         end
68     else
69         if mode == :check

```

```

70         return light.pos[2] <
            agent.pos[2] + 3
71     else
72         return (light.pos[2] -
            agent.pos[2] - 1.5) * -5
73     end
74 end
75 end
76

```

```

70         return light.pos[2] <
            agent.pos[2] + 3
71     else
72         return (light.pos[2] -
            agent.pos[2] - 1.5) * -5
73     end
74 end
75 end
76
77 const SMOOTHING_FACTOR = 0.18
78
79 function compute_speed(agent::Car)
80     return agent.street === av1 ?
        agent.vel[1] + 0.6 : agent.vel[2] + 2.0
81 end
82
83 function compute_back(agent::Car)
84     return agent.street === av1 ?
        agent.vel[1] - 0.2 : agent.vel[2] - 0.6
85 end
86
87 function compute_velocities(agent::Car,
        speed, back, dist)
88     if agent.street === av1
89         stop =
            (cos(agent.orientation) * max(back * (1
            - dist * (1 - SMOOTHING_FACTOR)), 0.0),
            0.0)
90         accelerate =
            (cos(agent.orientation) * max(0.0,
            speed * (1 - SMOOTHING_FACTOR / (0.3 +
            SMOOTHING_FACTOR))), 0.0)
91         reverse =
            (cos(agent.orientation) * min(back * (1
            - dist * (1 - SMOOTHING_FACTOR)), 1.0),
            0.0)
92     else
93         stop = (0.0, -
            sin(agent.orientation) * max(back * (1
            - SMOOTHING_FACTOR), 0.0))
94         accelerate = (0.0,
            sin(agent.orientation) * max(0.0, speed
            * (1 - SMOOTHING_FACTOR / (0.1 +
            SMOOTHING_FACTOR))))
95         reverse = (0.0, -
            sin(agent.orientation) * max(back,
            0.15))
96     end

```

```

77 # Comportamiento del auto
78 function agent_step!(agent::Car, model)
79     # Verificar el coche más cercano
80     closest_car, dist_to_car =
        closest_agent_ahead(agent, model, Car,
        20.5, is_car_ahead)
81
82     # Verificar el semáforo más cercano
83     light, dist_to_light =
        closest_agent_ahead(agent, model,
        stopLight, 20.0, is_light_ahead)
84
85     x = 0.18
86     # Suavizado para hacer la
        transición de velocidad más fluida
87     speed = agent.street === av1 ?
        agent.vel[1] + 0.6 : agent.vel[2] + 2.0
88     back = agent.street === av1 ?
        agent.vel[1] - 0.2 : agent.vel[2] - 0.6
89     # Definir decremento y aceleración
        según la calle (X o Y)
90
91     if agent.street === av1
92         # Para av1, los autos se mueven
        en el eje X
93         stop =
        (cos(agent.orientation)*max(back * (1-
        (dist_to_light<dist_to_car ?
        dist_to_light : dist_to_car)*(1-x)),
        0.0), 0.0) # Reduce la velocidad más
        lentamente
94         accelerate =
        (cos(agent.orientation)*max(0.0, speed
        * (1-x/(0.3+x))), 0.0) # Aumenta la
        velocidad gradualmente
95         reverse =
        (cos(agent.orientation)*min(back * (1-
        (dist_to_light<dist_to_car ?
        dist_to_light : dist_to_car)*(1-x)),
        1), 0.0) # Retrocede suavemente
96     else # agent.street === av2
97         # Para av2, los autos deben
        moverse hacia arriba (velocidad
        positiva en Y)

```

```

97     return stop, accelerate, reverse
98 end
99
100 # Comportamiento del auto
101 function agent_step!(agent::Car, model)
102     # Verificar el coche más cercano
103     closest_car, dist_to_car =
        closest_agent_ahead(agent, model, Car,
        20.5, is_car_ahead)
104
105     # Verificar el semáforo más cercano
106     light, dist_to_light =
        closest_agent_ahead(agent, model,
        stopLight, 20.0, is_light_ahead)
107
108     speed = compute_speed(agent)
109
110     back = compute_back(agent)
111
112     dist = min(dist_to_car,
        dist_to_light)
113     stop, accelerate, reverse =
        compute_velocities(agent, speed, back,
        dist)

```

```

97     stop = (0.0, -
sin(agent.orientation)*max(back * (1-
x), 0.0)) # Reduce la velocidad
suavemente
98     accelerate = (0.0,
sin(agent.orientation)*max(0.0, speed *
(1-x/(0.1+x)))) # Aumenta la velocidad
suavemente hacia arriba (positivo en Y)
99     reverse = (0.0, -
sin(agent.orientation) * max(back,
0.15)) # Retrocede suavemente con un
valor máximo
100 end
101
102
103     new_vel = accelerate
104
105     # Prioridad 1: Frenar si hay un
coche delante en la misma calle
106     if closest_car != nothing &&
dist_to_car < dist_to_light
107         if dist_to_car <= 2.5 &&
dist_to_car >= 1.2
108             new_vel = stop
109         elseif dist_to_car <
(agent.street === av2 ? 2.4 : 2.65)
110             new_vel = reverse
111         else # Si está a una distancia
segura, continuar acelerando
112             new_vel = accelerate
113     end
114     # Prioridad 2: Evaluar el semáforo,
pero solo si está en rojo o amarillo
115     elseif light != nothing
116         if light.status === red ||
light.status === yellow
117             if dist_to_light <= 3.5 +
(agent.street === av2 ? 5 : 0) &&
dist_to_light >= 1.8
118                 new_vel = stop #
Desacelerar si está cerca del semáforo
119             elseif dist_to_light < 1.8

```

```

113
114     new_vel = accelerate
115
116     if closest_car != nothing &&
dist_to_car < dist_to_light
117         if 1.2 <= dist_to_car <= 2.5
118             new_vel = stop
119         elseif dist_to_car <
(agent.street === av2 ? 2.5 : 2.75)
120             new_vel = reverse
121     end
122     elseif light != nothing &&
(light.status == red || light.status ==
yellow)
123         if dist_to_light <=
(agent.street === av2 ? 9.5 : 3.5) &&
dist_to_light >= 1.2
124             new_vel = stop
125         elseif dist_to_light < 1.4
126             new_vel = reverse

```

```

120         new_vel = reverse
121     end
122     else
123         new_vel = accelerate # Si
        el semáforo está en verde o lejos,
        acelerar
124     end
125 else
126     # Si no hay semáforo ni coche
    adelante, continuar acelerando
127     new_vel = accelerate
128     end # Multiply the direction by
    the speed scalar
129
130     # Aplicar suavizado en la velocidad
131     agent.vel = agent.vel .* (1 - x) .+
    new_vel .* x
132
133     # Verificar si el auto ha sido
    teletransportado
134
135     if agent.pos[1] < 0.5
136         agent.vel = (0.15,0)
137     end
138
139     # Mover el auto en el espacio sin
    cambiar de dirección
140
141     move_agent!(agent, model, 0.4)
142 end
143
144 function agent_step!(agent::stopLight,
    model)
145     cycle_length = 2 * (green_duration
    + yellow_duration) # Ciclo completo de
    28 pasos
146
147     # Incrementamos el contador de
    tiempo del agente
148     agent.time_counter += 1
149
150     # Si el contador alcanza el final
    del ciclo, lo reiniciamos
151     if agent.time_counter >
    cycle_length

```

```

127     end
128 end
129
130 agent.vel = agent.vel .* (1 -
    SMOOTHING_FACTOR) .+ new_vel .*
    SMOOTHING_FACTOR
131
132 if agent.pos[1] < 0.5
133     agent.vel = (0.15, 0.0)
134 end
135
136 move_agent!(agent, model, 0.4)
137 end
138
139 function agent_step!(agent::stopLight,
    model)
140     cycle_length = 2 * (green_duration
    + yellow_duration) # Ciclo completo de
    28 pasos
141
142     # Incrementamos el contador de
    tiempo del agente
143     agent.time_counter += 1
144
145     # Si el contador alcanza el final
    del ciclo, lo reiniciamos
146     if agent.time_counter >
    cycle_length

```

```

150         agent.time_counter = 1
151     end
152
153     # Cambiamos el estado del semáforo
    en función del contador
154     if agent.time_counter <=
    green_duration
155         agent.status = green
156     elseif agent.time_counter <=
    green_duration + yellow_duration
157         agent.status = yellow
158     else
159         agent.status = red
160     end
161 end
162
163
164 function initialize_model(extent = (28,
    15); numCarsN = 0, numCars0 = 1)
165     space2d = ContinuousSpace(extent;
    spacing = 0.5, periodic = true)
166
167     rng = Random.MersenneTwister()
168
169     model = StandardABM(Union{Car,
    stopLight}, space2d; rng, agent_step!,
    scheduler = Schedulers.fastest)
170     #model = StandardABM(stopLight,
    space2d; agent_step!, scheduler =
    Schedulers.Randomly())
171     #model = StandardABM(Car, space2d;
    rng, agent_step!, scheduler =
    Schedulers.Randomly())
172     add_agent!(stopLight, model; pos =
    SVector{2, Float64}(12, 3.5), vel =
    SVector{2, Float64}(0.0, 0.0))
173     add_agent!(stopLight, model; pos =
    SVector{2, Float64}(16.3, 8.5), vel =
    SVector{2, Float64}(0.0, 0.0))
174     changing = true
175     for agent in allagents(model)
176         if changing === true
177             agent.status = green
178             agent.street = av2
179             changing = false
180         else
181             agent.status = red
182             agent.time_counter =
    green_duration + yellow_duration

```

```

147         agent.time_counter = 1
148     end
149
150     # Cambiamos el estado del semáforo
    en función del contador
151     if agent.time_counter <=
    green_duration
152         agent.status = green
153     elseif agent.time_counter <=
    green_duration + yellow_duration
154         agent.status = yellow
155     else
156         agent.status = red
157     end
158 end
159
160
161 function initialize_model(extent = (28,
    15); numCarsN = 0, numCars0 = 1)
162     space2d = ContinuousSpace(extent;
    spacing = 0.5, periodic = true)
163
164     rng = Random.MersenneTwister()
165
166     model = StandardABM(Union{Car,
    stopLight}, space2d; rng, agent_step!,
    scheduler = Schedulers.fastest)
167     #model = StandardABM(stopLight,
    space2d; agent_step!, scheduler =
    Schedulers.Randomly())
168     #model = StandardABM(Car, space2d;
    rng, agent_step!, scheduler =
    Schedulers.Randomly())
169     add_agent!(stopLight, model; pos =
    SVector{2, Float64}(12, 3.5), vel =
    SVector{2, Float64}(0.0, 0.0))
170     add_agent!(stopLight, model; pos =
    SVector{2, Float64}(16.3, 8.5), vel =
    SVector{2, Float64}(0.0, 0.0))
171     changing = true
172     for agent in allagents(model)
173         if changing === true
174             agent.status = green
175             agent.street = av2
176             changing = false
177         else
178             agent.status = red
179             agent.time_counter =
    green_duration + yellow_duration

```

```

183         agent.street = av1
184         changing = true
185     end
186 end
187 first = true
188 range_x = (5.0, 20.0) # Rango de
posiciones X para av1
189 range_y = (0.0, 10.0) # Rango de
posiciones Y para av2
190
191 if numCarsN != 0
192     for _ in 1:numCarsN
193         if first
194             pos_y =
rand(range_y[1]:0.5:range_y[2]) #
Rango para av2
195             add_agent!(Car, model;
pos = (rand(13:14), pos_y),
vel=SVector{2, Float64}(0.0, 0.1),
street = av2, orientation = right)
196             first = false # Ya no
es el primer auto
197         else
198             pos_y =
rand(range_y[1]:0.5:range_y[2]) #
Rango para av2
199             add_agent!(Car, model;
pos = (rand(13:14), pos_y),
vel=SVector{2, Float64}(0.0, 0.1),
street = av2, orientation = right)
200         end
201     end
202 end
203 if numCarsO != 0
204     first = true
205     for _ in 1:numCarsO
206         if first
207             pos_x =
rand(range_x[1]:0.5:range_x[2]) #
Rango para av1
208             add_agent!(Car, model;
pos = (pos_x, rand(7:8)),
vel=SVector{2, Float64}(0.1, 0.0))
209             first = false # Ya no
es el primer auto
210         else
211             # Añadir auto en av1
(horizontal)

```

```

180         agent.street = av1
181         changing = true
182     end
183 end
184 first = true
185 range_x = (5.0, 20.0) # Rango de
posiciones X para av1
186 range_y = (0.0, 10.0) # Rango de
posiciones Y para av2
187
188 if numCarsN != 0
189     for _ in 1:numCarsN
190         if first
191             pos_y =
rand(range_y[1]:0.5:range_y[2]) #
Rango para av2
192             add_agent!(Car, model;
pos = (rand(13:14), pos_y),
vel=SVector{2, Float64}(0.0, 0.1),
street = av2, orientation = right)
193             first = false # Ya no
es el primer auto
194         else
195             pos_y =
rand(range_y[1]:0.5:range_y[2]) #
Rango para av2
196             add_agent!(Car, model;
pos = (rand(13:14), pos_y),
vel=SVector{2, Float64}(0.0, 0.1),
street = av2, orientation = right)
197         end
198     end
199 end
200 if numCarsO != 0
201     first = true
202     for _ in 1:numCarsO
203         if first
204             pos_x =
rand(range_x[1]:0.5:range_x[2]) #
Rango para av1
205             add_agent!(Car, model;
pos = (pos_x, rand(7:8)),
vel=SVector{2, Float64}(0.1, 0.0))
206             first = false # Ya no
es el primer auto
207         else
208             # Añadir auto en av1
(horizontal)

```



```
212         pos_x =
            rand(range_x[1]:0.5:range_x[2]) #
            Rango para av1
213         add_agent!(Car, model;
            pos = (pos_x, rand(7:8)),
            vel=SVector{2, Float64}(0.1, 0.0))
214     end
215 end
216 end
217 model
218 end
219
220 #Semáforo = 10 pasos en Verde, 4 pasos
    en Amarillo, 14 pasos en Rojo
```

```
209         pos_x =
            rand(range_x[1]:0.5:range_x[2]) #
            Rango para av1
210         add_agent!(Car, model;
            pos = (pos_x, rand(7:8)),
            vel=SVector{2, Float64}(0.1, 0.0))
211     end
212 end
213 end
214 model
215 end
216
217 #Semáforo = 10 pasos en Verde, 4 pasos
    en Amarillo, 14 pasos en Rojo
```