



## Untitled diff

— 26 removals

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)

```

+ 10 additions

201 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

```

```

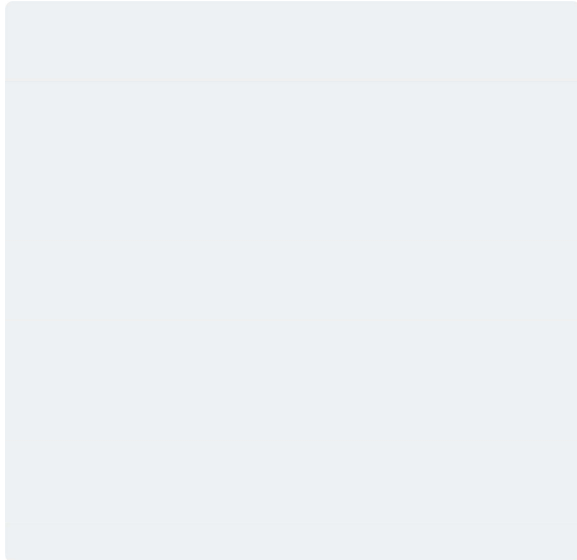
34         if isa(neighbor, T) &&
           neighbor.street == agent.street &&
           is_ahead_fn(agent, neighbor, :check)
35             dist = is_ahead_fn(agent,
           neighbor, :distance)
36             if dist < min_distance
37                 min_distance = dist
38                 closest = 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

```



```

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

```

```

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 function add_cars(model, n,
    street::Streets, orientation::Float64)
78     for _ in 1:n
79         pos = street == av1 ?
            (rand(5.0:0.5:20.0), rand(7:8)) :
            (rand(13:14), rand(0.0:0.5:10.0))
80         vel = street == av1 ?
            SVector(0.1, 0.0) : SVector(0.0, 0.1)
81         add_agent!(Car, model; pos=pos,
            vel=vel, street=street,
            orientation=orientation)
82     end
83 end
84

```

```

85 const SMOOTHING_FACTOR = 0.18
86
87 function compute_speed(agent::Car)
88     return agent.street === av1 ?
        agent.vel[1] + 0.6 : agent.vel[2] + 2.0
89 end
90
91 function compute_back(agent::Car)
92     return agent.street === av1 ?
        agent.vel[1] - 0.2 : agent.vel[2] - 0.6
93 end
94
95 function compute_velocities(agent::Car,
    speed, back, dist)
96     if agent.street === av1
97         stop =
            (cos(agent.orientation) * max(back * (1
            - dist * (1 - SMOOTHING_FACTOR)), 0.0),
            0.0)
98         accelerate =
            (cos(agent.orientation) * max(0.0,
            speed * (1 - SMOOTHING_FACTOR / (0.3 +
            SMOOTHING_FACTOR))), 0.0)
99         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
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
109     speed = compute_speed(agent)
110     back  = compute_back(agent)
111     dist  = min(dist_to_car,
dist_to_light)
112     stop, accelerate, reverse =
compute_velocities(agent, speed, back,
dist)
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

```

```

- dist * (1 - SMOOTHING_FACTOR)), 1.0),
0.0)
100     else
101         stop      = (0.0, -
sin(agent.orientation) * max(back * (1
- SMOOTHING_FACTOR), 0.0))
102         accelerate = (0.0,
sin(agent.orientation) * max(0.0, speed
* (1 - SMOOTHING_FACTOR / (0.1 +
SMOOTHING_FACTOR))))
103         reverse   = (0.0, -
sin(agent.orientation) * max(back,
0.15))
104     end
105     return stop, accelerate, reverse
106 end
107
108 # Comportamiento del auto
109 function agent_step!(agent::Car, model)
110     # Verificar el coche más cercano
111     closest_car, dist_to_car =
closest_agent_ahead(agent, model, Car,
20.5, is_car_ahead)
112
113     # Verificar el semáforo más cercano
114     light, dist_to_light =
closest_agent_ahead(agent, model,
stopLight, 20.0, is_light_ahead)
115
116
117     speed = compute_speed(agent)
118     back  = compute_back(agent)
119     dist  = min(dist_to_car,
dist_to_light)
120     stop, accelerate, reverse =
compute_velocities(agent, speed, back,
dist)
121
122     new_vel = accelerate
123
124     if closest_car !== nothing &&
dist_to_car < dist_to_light
125         if 1.2 <= dist_to_car <= 2.5
126             new_vel = stop
127         elseif dist_to_car <
(agent.street === av2 ? 2.5 : 2.75)
128             new_vel = reverse
129     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
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
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

```

```

130     elseif light != nothing &&
      (light.status == red || light.status ==
yellow)
131         if dist_to_light <=
      (agent.street === av2 ? 9.5 : 3.5) &&
      dist_to_light >= 1.2
132             new_vel = stop
133         elseif dist_to_light < 1.4
134             new_vel = reverse
135         end
136     end
137
138     agent.vel = agent.vel .* (1 -
SMOOTHING_FACTOR) .+ new_vel .*
SMOOTHING_FACTOR
139
140     if agent.pos[1] < 0.5
141         agent.vel = (0.15, 0.0)
142     end
143
144     move_agent!(agent, model, 0.4)
145 end
146
147 function agent_step!(agent::stopLight,
model)
148     cycle_length = 2 * (green_duration
+ yellow_duration) # Ciclo completo de
28 pasos
149
150     # Incrementamos el contador de
tiempo del agente
151     agent.time_counter += 1
152
153     # Si el contador alcanza el final
del ciclo, lo reiniciamos
154     if agent.time_counter >
cycle_length
155         agent.time_counter = 1
156     end
157
158     # Cambiamos el estado del semáforo
en función del contador
159     if agent.time_counter <=
green_duration
160         agent.status = green
161     elseif agent.time_counter <=
green_duration + yellow_duration
162         agent.status = yellow
163     else

```

```

156         agent.status = red
157     end
158 end
159
160
161 function initialize_model(extent = (28,
162     15); numCarsN = 0, numCars0 = 1)
163
164     space2d = ContinuousSpace(extent;
165     spacing = 0.5, periodic = true)
166
167     rng = Random.MersenneTwister()
168
169     model = StandardABM(Union{Car,
170     stopLight}, space2d; rng, agent_step!,
171     scheduler = Schedulers.fastest)
172
173     #model = StandardABM(stopLight,
174     space2d; agent_step!, scheduler =
175     Schedulers.Randomly())
176
177     #model = StandardABM(Car, space2d;
178     rng, agent_step!, scheduler =
179     Schedulers.Randomly())
180
181     add_agent!(stopLight, model; pos =
182     SVector{2, Float64}(12, 3.5), vel =
183     SVector{2, Float64}(0.0, 0.0))
184
185     add_agent!(stopLight, model; pos =
186     SVector{2, Float64}(16.3, 8.5), vel =
187     SVector{2, Float64}(0.0, 0.0))
188
189     changing = true
190     for agent in allagents(model)
191         if changing === true
192             agent.status = green
193             agent.street = av2
194             changing = false
195         else
196             agent.status = red
197             agent.time_counter =
198             green_duration + yellow_duration
199             agent.street = av1
200             changing = true
201         end
202     end
203
204     first = true
205     range_x = (5.0, 20.0) # Rango de
206     posiciones X para av1
207     range_y = (0.0, 10.0) # Rango de
208     posiciones Y para av2
209
210
211     if numCarsN != 0

```

```

164         agent.status = red
165     end
166 end
167
168
169 function initialize_model(extent = (28,
170     15); numCarsN = 0, numCars0 = 1)
171
172     space2d = ContinuousSpace(extent;
173     spacing = 0.5, periodic = true)
174
175     rng = Random.MersenneTwister()
176
177     model = StandardABM(Union{Car,
178     stopLight}, space2d; rng, agent_step!,
179     scheduler = Schedulers.fastest)
180
181     #model = StandardABM(stopLight,
182     space2d; agent_step!, scheduler =
183     Schedulers.Randomly())
184
185     #model = StandardABM(Car, space2d;
186     rng, agent_step!, scheduler =
187     Schedulers.Randomly())
188
189     add_agent!(stopLight, model; pos =
190     SVector{2, Float64}(12, 3.5), vel =
191     SVector{2, Float64}(0.0, 0.0))
192
193     add_agent!(stopLight, model; pos =
194     SVector{2, Float64}(16.3, 8.5), vel =
195     SVector{2, Float64}(0.0, 0.0))
196
197     changing = true
198     for agent in allagents(model)
199         if changing === true
200             agent.status = green
201             agent.street = av2
202             changing = false
203         else
204             agent.status = red
205             agent.time_counter =
206             green_duration + yellow_duration
207             agent.street = av1
208             changing = true
209         end
210     end
211
212     first = true
213     range_x = (5.0, 20.0) # Rango de
214     posiciones X para av1
215     range_y = (0.0, 10.0) # Rango de
216     posiciones Y para av2
217
218

```

```

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 numCars0 != 0
201     first = true
202     for _ in 1:numCars0
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)
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

```

```

196     add_cars(model, numCarsN, av2,
Float64(right))
197     add_cars(model, numCars0, av1,
Float64(normal))

```

```
213 end
214     model
215 end
216
217 #Semáforo = 10 pasos en Verde, 4 pasos
    en Amarillo, 14 pasos en Rojo
```

```
198     model
199 end
200
201 #Semáforo = 10 pasos en Verde, 4 pasos
    en Amarillo, 14 pasos en Rojo
```