



Untitled diff

— 75 removals

242 lines

```

1 using Agents, Random
2 using StaticArrays: SVector
3
4 # Estados de los Semáforos
5 @enum LightColor green yellow red
6 @enum Streets av1 av2
7
8 normal = 0
9 left =  $\pi/2$ 
10 down =  $\pi$ 
11 right =  $3\pi/2$ 
12
13 @agent struct
14     Car(ContinuousAgent{2,Float64})
15     accelerating::Bool = true
16     street::Streets = av1
17     orientation::Float64 = normal
18 end
19 @agent struct
20     stopLight(ContinuousAgent{2,Float64})
21     status::LightColor = red
22     time_counter::Int = 0
23     street::Streets = av1
24 end
25 green_duration = 45
26 yellow_duration = 15
27
28 # Verificar el semáforo más cercano
29 # delante en el eje X
30
31 function closest_car_ahead(agent::Car,
32     model)
33     closest_car = nothing

```

+ 46 additions

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

```

```

31     min_distance = Inf
32
33     # Buscar coches dentro de un radio
    de 20.0 unidades
34     for neighbor in
nearby_agents(agent, model,
(agent.street === av1 ? 20.5 : 1.8))
35         if isa(neighbor, Car) &&
agent.street === neighbor.street #
Solo considerar coches en la misma
calle
36         # Para av1 (movimiento en
X)
37         if agent.street === av1 &&
agent.pos[2] === neighbor.pos[2]
38         if neighbor.pos[1] >
agent.pos[1] # Solo autos adelante (X
mayor)
39         dist_to_neighbor =
neighbor.pos[1] - agent.pos[1]
40         # Seleccionar el
coche más cercano adelante
41         if dist_to_neighbor
< min_distance
42         min_distance =
dist_to_neighbor
43         closest_car =
neighbor
44         end
45     end
46     # Para av2 (movimiento en
Y)
47     elseif agent.street === av2
&& agent.pos[1] === neighbor.pos[1]
48         if neighbor.pos[2] <
agent.pos[2]
49         dist_to_neighbor =
(neighbor.pos[2] - agent.pos[2]) * -1
50         # Seleccionar el
coche más cercano adelante
51         if dist_to_neighbor
< min_distance
52         min_distance =
dist_to_neighbor
53         closest_car =
neighbor
54     end
55 end

```

```

31     min_distance = Inf
32
33     for neighbor in
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
37         if dist < min_distance
38             min_distance = dist
39
40             closest = neighbor

```

```

56         end
57     end
58 end

59
60     return closest_car, min_distance
61 end
62
63 # Verificar el semáforo más cercano
64 # delante en el eje X
65 function
66     closest_light Ahead(agent::Car, model)
67     closest_light = nothing
68     min_distance = Inf
69
70     # Buscar semáforos dentro de un
71     # radio de 30.0 unidades
72     for neighbor in
73         nearby_agents(agent, model, 20.0)
74         if neighbor.street ===
75             agent.street && neighbor isa stopLight
76             # Asegurar que están en la misma calle
77             # (misma posición en Y)
78             # Verificar que el semáforo
79             # esté delante del coche en el eje X
80             if agent.street === av1
81                 if neighbor.pos[1] >
82                     agent.pos[1]
83
84                     dist_to_light =
85                     neighbor.pos[1] - agent.pos[1]
86
87                     # Seleccionar el
88                     # semáforo más cercano en el eje X
89                     if dist_to_light <
90                         min_distance
91                         min_distance =
92                         dist_to_light
93                         closest_light =
94                         neighbor
95                     end
96                 end
97             elseif agent.street === av2
98                 if neighbor.pos[2] <
99                     agent.pos[2] + 3

```

```

39         end
40     end
41 end

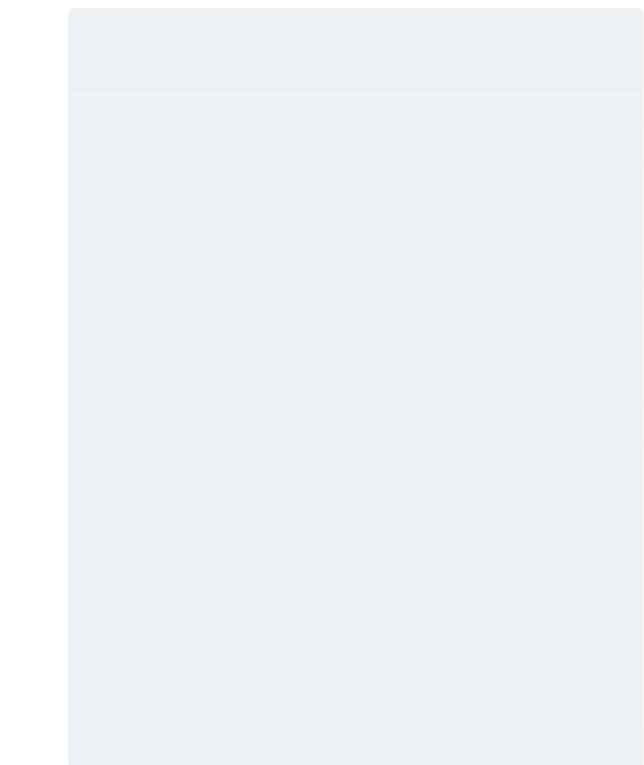
42     return closest, min_distance
43 end
44
45 function is_car Ahead(agent, neighbor,
46     mode = :check)
47     if agent.street == av1
48         if mode == :check
49             return neighbor.pos[1] >
50                 agent.pos[1] && agent.pos[2] ==
51                 neighbor.pos[2]
52         else # :distance
53             return neighbor.pos[1] -
54                 agent.pos[1]
55         end
56     else # av2
57         if mode == :check
58             return neighbor.pos[2] <
59                 agent.pos[2] && agent.pos[1] ==
60                 neighbor.pos[1]
61         else # :distance
62             return (agent.pos[2] -
63                 neighbor.pos[2])

```

```

83         dist_to_light =
      (neighbor.pos[2] - agent.pos[2]-1.5) *
      -5
84
85         # Seleccionar el
      semáforo más cercano en el eje X
86         if dist_to_light <
      min_distance
87             min_distance =
      dist_to_light
88             closest_light =
      neighbor
89         end
90     end
91 end
92
93 end
94
95 return closest_light, min_distance
96

```



```

97 end
98
99 # Comportamiento del auto
100 function agent_step!(agent::Car, model)
101     # Verificar el coche más cercano
102
103     closest_car, dist_to_car =
      closest_car_ahead(agent, model)

```

```

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 # Comportamiento del auto
78 function agent_step!(agent::Car, model)
79     # Verificar el coche más cercano
80
81     closest_car, dist_to_car =
      closest_agent_ahead(agent, model, Car,

```

```

103
104     # Verificar el semáforo más cercano
105     light, dist_to_light =
closest_light_ahead(agent, model)
106
107     x = 0.18
108     # Suavizado para hacer la
transición de velocidad más fluida
109     speed = agent.street == av1 ?
agent.vel[1] + 0.6 : agent.vel[2] + 2.0
110     back = agent.street == av1 ?
agent.vel[1] - 0.2 : agent.vel[2] - 0.6
111     # Definir decremento y aceleración
según la calle (X o Y)
112     if agent.street == av1
113         # Para av1, los autos se mueven
en el eje X
114         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
115         accelerate =
(cos(agent.orientation)*max(0.0, speed
* (1-x/(0.3+x))), 0.0) # Aumenta la
velocidad gradualmente
116         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
117     else # agent.street == av2
118         # Para av2, los autos deben
moverse hacia arriba (velocidad
positiva en Y)
119         stop = (0.0, -
sin(agent.orientation)*max(back * (1-
x), 0.0)) # Reduce la velocidad
suavemente
120         accelerate = (0.0,
sin(agent.orientation)*max(0.0, speed *
(1-x/(0.1+x)))) # Aumenta la velocidad
suavemente hacia arriba (positivo en Y)

```

```

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     if agent.street == av1
91         # Para av1, los autos se mueven
en el eje X
92         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
93         accelerate =
(cos(agent.orientation)*max(0.0, speed
* (1-x/(0.3+x))), 0.0) # Aumenta la
velocidad gradualmente
94         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
95     else # agent.street == av2
96         # Para av2, los autos deben
moverse hacia arriba (velocidad
positiva en Y)
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)

```

```
121         reverse = (0.0, -  
    sin(agent.orientation) * max(back,  
    0.15)) # Retrocede suavemente con un  
    valor máximo  
122     end  
123  
124  
125     new_vel = accelerate  
  
242 #Semáforo = 10 pasos en Verde, 4 pasos  
    en Amarillo, 14 pasos en Rojo
```

```
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  
  
220 #Semáforo = 10 pasos en Verde, 4 pasos  
    en Amarillo, 14 pasos en Rojo
```