Diffchecker

Untitled diff

− 75 removals	242 lines	+ 46 additions	220 lines

```
using Agents, Random
using StaticArrays: SVector


# Estados de los Semáforos
@enum LightColor green yellow red
@enum Streets av1 av2

normal = 0
left = π/2
down = π
right = 3π/2

@agent struct
Car(ContinuousAgent{2,Float64})
    accelerating::Bool = true
    street::Streets = av1
    orientation::Float64 = normal
end

@agent struct
stopLight(ContinuousAgent{2,Float64})
    status::LightColor = red
    time_counter::Int = 0
    street::Streets = av1
end

green_duration = 45
yellow_duration = 15

# Verificar el semáforo más cercano
delante en el eje X


function closest_car_ahead(agent::Car,
model)
    closest_car = nothing
```

```
using Agents, Random
using StaticArrays: SVector
using LinearAlgebra

# Estados de los Semáforos
@enum LightColor green yellow red
@enum Streets av1 av2

normal = 0
left = π/2
down = π
right = 3π/2

@agent struct
Car(ContinuousAgent{2,Float64})
    accelerating::Bool = true
    street::Streets = av1
    orientation::Float64 = normal
end

@agent struct
stopLight(ContinuousAgent{2,Float64})
    status::LightColor = red
    time_counter::Int = 0
    street::Streets = av1
end

green_duration = 45
yellow_duration = 15

function
closest_agent_ahead(agent::Car, model,
::Type{T}, radius, is_ahead_fn) where
{T}
    closest = nothing
```

```
    min_distance = Inf

    # Buscar coches dentro de un radio de 20.0 unidades
    for neighbor in nearby_agents(agent, model, (agent.street === av1 ? 20.5 : 1.8))
        if isa(neighbor, Car) && agent.street === neighbor.street  # Solo considerar coches en la misma calle
            # Para av1 (movimiento en X)
            if agent.street === av1 && agent.pos[2] === neighbor.pos[2]
                if neighbor.pos[1] > agent.pos[1]  # Solo autos adelante (X mayor)
                    dist_to_neighbor = neighbor.pos[1] - agent.pos[1]
                    # Seleccionar el coche más cercano adelante
                    if dist_to_neighbor < min_distance
                        min_distance = dist_to_neighbor
                        closest_car = neighbor
                    end
                end
            # Para av2 (movimiento en Y)
            elseif agent.street === av2 && agent.pos[1] === neighbor.pos[1]
                if neighbor.pos[2] < agent.pos[2]
                    dist_to_neighbor = (neighbor.pos[2] - agent.pos[2]) * -1
                    # Seleccionar el coche más cercano adelante
                    if dist_to_neighbor < min_distance
                        min_distance = dist_to_neighbor
                        closest_car = neighbor
                    end
                end
```

```
    min_distance = Inf

    for neighbor in nearby_agents(agent, model, radius)
        if isa(neighbor, T) && neighbor.street == agent.street && is_ahead_fn(agent, neighbor, :check)
            dist = is_ahead_fn(agent, neighbor, :distance)


            if dist < min_distance
                min_distance = dist
                closest = neighbor
```

```
            end
        end
    end

    return closest_car, min_distance
end

# Verificar el semáforo más cercano delante en el eje X
function closest_light_ahead(agent::Car, model)
    closest_light = nothing
    min_distance = Inf


    # Buscar semáforos dentro de un radio de 30.0 unidades
    for neighbor in nearby_agents(agent, model, 20.0)
        if neighbor.street === agent.street && neighbor isa stopLight
# Asegurar que están en la misma calle (misma posición en Y)
            # Verificar que el semáforo esté delante del coche en el eje X
            if agent.street === av1
                if neighbor.pos[1] > agent.pos[1]

                    dist_to_light = neighbor.pos[1] - agent.pos[1]


                # Seleccionar el semáforo más cercano en el eje X
                    if dist_to_light < min_distance
                        min_distance = dist_to_light
                        closest_light = neighbor
                    end
                end
            elseif agent.street === av2
                if neighbor.pos[2] < agent.pos[2] + 3
```

```
            end
        end
    end

    return closest, min_distance

end

function is_car_ahead(agent, neighbor, mode = :check)
    if agent.street == av1

        if mode == :check
            return neighbor.pos[1] > agent.pos[1] && agent.pos[2] == neighbor.pos[2]
        else  # :distance

            return neighbor.pos[1] - agent.pos[1]
        end




    else  # av2

        if mode == :check
            return neighbor.pos[2] < agent.pos[2] && agent.pos[1] == neighbor.pos[1]
        else  # :distance

            return (agent.pos[2] - neighbor.pos[2])
```

```
                    dist_to_light =
(neighbor.pos[2] - agent.pos[2]-1.5) *
-5
                
                    # Seleccionar el
semáforo más cercano en el eje X
                    if dist_to_light <
min_distance
                        min_distance =
dist_to_light
                        closest_light =
neighbor
                    end
                end
            end
        end
    end

    return closest_light, min_distance


end

# Comportamiento del auto
function agent_step!(agent::Car, model)
    # Verificar el coche más cercano
    closest_car, dist_to_car =
closest_car_ahead(agent, model)
```

```
        end
    end
end


function is_light_ahead(agent, light,
mode = :check)
    if agent.street == av1
        if mode == :check
            return light.pos[1] >
agent.pos[1]
        else
            return light.pos[1] -
agent.pos[1]
        end
    else
        if mode == :check
            return light.pos[2] <
agent.pos[2] + 3
        else
            return (light.pos[2] -
agent.pos[2] - 1.5) * -5
        end
    end
end

# Comportamiento del auto
function agent_step!(agent::Car, model)
    # Verificar el coche más cercano
    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
```