Diffchecker

Untitled diff

− 55 removals 220 lines

```
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

    for neighbor in nearby_agents(agent, model, radius)
```

+ 43 additions 217 lines

```
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

    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, 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])
        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
```

```
        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, 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])
        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

```

```
            return light.pos[2] < agent.pos[2] + 3
        else
            return (light.pos[2] - agent.pos[2] - 1.5) * -5
        end
    end
end

const SMOOTHING_FACTOR = 0.18

function compute_speed(agent::Car)
    return agent.street === av1 ? agent.vel[1] + 0.6 : agent.vel[2] + 2.0
end

function compute_back(agent::Car)
    return agent.street === av1 ? agent.vel[1] - 0.2 : agent.vel[2] - 0.6
end

function compute_velocities(agent::Car, speed, back, dist)
    if agent.street === av1
        stop      = (cos(agent.orientation) * max(back * (1 - dist * (1 - SMOOTHING_FACTOR)), 0.0), 0.0)
        accelerate = (cos(agent.orientation) * max(0.0, speed * (1 - SMOOTHING_FACTOR / (0.3 + SMOOTHING_FACTOR))), 0.0)
        reverse   = (cos(agent.orientation) * min(back * (1 - dist * (1 - SMOOTHING_FACTOR)), 1.0), 0.0)
    else
        stop      = (0.0, -sin(agent.orientation) * max(back * (1 - SMOOTHING_FACTOR), 0.0))
        accelerate = (0.0, sin(agent.orientation) * max(0.0, speed * (1 - SMOOTHING_FACTOR / (0.1 + SMOOTHING_FACTOR))))
        reverse   = (0.0, -sin(agent.orientation) * max(back, 0.15))
    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, 20.5, is_car_ahead)

    # Verificar el semáforo más cercano
    light, dist_to_light = closest_agent_ahead(agent, model, stopLight, 20.0, is_light_ahead)

    x = 0.18
    # Suavizado para hacer la transición de velocidad más fluida
    speed = agent.street === av1 ? agent.vel[1] + 0.6 : agent.vel[2] + 2.0
    back = agent.street === av1 ? agent.vel[1] - 0.2 : agent.vel[2] - 0.6
    # Definir decremento y aceleración según la calle (X o Y)

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

```
    return stop, accelerate, reverse
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, 20.5, is_car_ahead)

    # Verificar el semáforo más cercano
    light, dist_to_light = closest_agent_ahead(agent, model, stopLight, 20.0, is_light_ahead)


    speed = compute_speed(agent)
    back  = compute_back(agent)
    dist  = min(dist_to_car, dist_to_light)
    stop, accelerate, reverse = compute_velocities(agent, speed, back, dist)
```

```
        stop = (0.0, -sin(agent.orientation)*max(back * (1-x), 0.0))  # Reduce la velocidad suavemente
        accelerate = (0.0, sin(agent.orientation)*max(0.0, speed * (1-x/(0.1+x)))) # Aumenta la velocidad suavemente hacia arriba (positivo en Y)
        reverse = (0.0, -sin(agent.orientation) * max(back, 0.15))  # Retrocede suavemente con un valor máximo
    end


    new_vel = accelerate

    # Prioridad 1: Frenar si hay un coche delante en la misma calle
    if closest_car !== nothing && dist_to_car < dist_to_light
        if dist_to_car <= 2.5 && dist_to_car >= 1.2
            new_vel = stop
        elseif dist_to_car < (agent.street === av2 ? 2.4 : 2.65)
                new_vel = reverse
        else  # Si está a una distancia segura, continuar acelerando
            new_vel = accelerate
        end
    # Prioridad 2: Evaluar el semáforo, pero solo si está en rojo o amarillo
    elseif light !== nothing
        if light.status === red || light.status === yellow
            if dist_to_light <= 3.5 + (agent.street === av2 ? 5 : 0) && dist_to_light >= 1.8
                new_vel = stop  # Desacelerar si está cerca del semáforo
            elseif dist_to_light < 1.8
```

```

    new_vel = accelerate

    if closest_car !== nothing && dist_to_car < dist_to_light
        if 1.2 <= dist_to_car <= 2.5
            new_vel = stop
        elseif dist_to_car < (agent.street === av2 ? 2.5 : 2.75)
            new_vel = reverse
        end
    elseif light !== nothing && (light.status == red || light.status == yellow)
        if dist_to_light <= (agent.street === av2 ? 9.5 : 3.5) && dist_to_light >= 1.2
            new_vel = stop
        elseif dist_to_light < 1.4
            new_vel = reverse
```

```
                new_vel = reverse
            end
        else
            new_vel = accelerate  # Si el semáforo está en verde o lejos, acelerar
        end
    else
        # Si no hay semáforo ni coche adelante, continuar acelerando
        new_vel = accelerate
    end  # Multiply the direction by the speed scalar

    # Aplicar suavizado en la velocidad
    agent.vel = agent.vel .* (1 - x) .+ new_vel .* x

    # Verificar si el auto ha sido teletransportado
    if agent.pos[1] < 0.5
        agent.vel = (0.15,0)
    end

    # Mover el auto en el espacio sin cambiar de dirección
    move_agent!(agent, model, 0.4)
end

function agent_step!(agent::stopLight, model)
    cycle_length = 2 * (green_duration + yellow_duration)  # Ciclo completo de 28 pasos

    # Incrementamos el contador de tiempo del agente
    agent.time_counter += 1

    # Si el contador alcanza el final del ciclo, lo reiniciamos
    if agent.time_counter > cycle_length
```

```
        end
    end

    agent.vel = agent.vel .* (1 - SMOOTHING_FACTOR) .+ new_vel .* SMOOTHING_FACTOR

    if agent.pos[1] < 0.5
        agent.vel = (0.15, 0.0)
    end

    move_agent!(agent, model, 0.4)
end

function agent_step!(agent::stopLight, model)
    cycle_length = 2 * (green_duration + yellow_duration)  # Ciclo completo de 28 pasos

    # Incrementamos el contador de tiempo del agente
    agent.time_counter += 1

    # Si el contador alcanza el final del ciclo, lo reiniciamos
    if agent.time_counter > cycle_length
```

```
        agent.time_counter = 1
    end

    # Cambiamos el estado del semáforo en función del contador
    if agent.time_counter <= green_duration
        agent.status = green
    elseif agent.time_counter <= green_duration + yellow_duration
        agent.status = yellow
    else
        agent.status = red
    end
end


function initialize_model(extent = (28, 15); numCarsN = 0, numCarsO = 1)
    space2d = ContinuousSpace(extent; spacing = 0.5, periodic = true)

    rng = Random.MersenneTwister()

    model = StandardABM(Union{Car, stopLight}, space2d; rng, agent_step!, scheduler = Schedulers.fastest)
    #model = StandardABM(stopLight, space2d; agent_step!, scheduler = Schedulers.Randomly())
    #model = StandardABM(Car, space2d; rng, agent_step!, scheduler = Schedulers.Randomly())
    add_agent!(stopLight, model; pos = SVector{2, Float64}(12, 3.5), vel = SVector{2, Float64}(0.0, 0.0))
    add_agent!(stopLight, model; pos = SVector{2, Float64}(16.3, 8.5), vel = SVector{2, Float64}(0.0, 0.0))
    changing = true
    for agent in allagents(model)
        if changing === true
            agent.status = green
            agent.street = av2
            changing = false
        else
            agent.status = red
            agent.time_counter = green_duration + yellow_duration
```

```
        agent.time_counter = 1
    end

    # Cambiamos el estado del semáforo en función del contador
    if agent.time_counter <= green_duration
        agent.status = green
    elseif agent.time_counter <= green_duration + yellow_duration
        agent.status = yellow
    else
        agent.status = red
    end
end


function initialize_model(extent = (28, 15); numCarsN = 0, numCarsO = 1)
    space2d = ContinuousSpace(extent; spacing = 0.5, periodic = true)

    rng = Random.MersenneTwister()

    model = StandardABM(Union{Car, stopLight}, space2d; rng, agent_step!, scheduler = Schedulers.fastest)
    #model = StandardABM(stopLight, space2d; agent_step!, scheduler = Schedulers.Randomly())
    #model = StandardABM(Car, space2d; rng, agent_step!, scheduler = Schedulers.Randomly())
    add_agent!(stopLight, model; pos = SVector{2, Float64}(12, 3.5), vel = SVector{2, Float64}(0.0, 0.0))
    add_agent!(stopLight, model; pos = SVector{2, Float64}(16.3, 8.5), vel = SVector{2, Float64}(0.0, 0.0))
    changing = true
    for agent in allagents(model)
        if changing === true
            agent.status = green
            agent.street = av2
            changing = false
        else
            agent.status = red
            agent.time_counter = green_duration + yellow_duration
            agent.street = av1
            changing = true
        end
    end
    first = true
    range_x = (5.0, 20.0)  # Rango de posiciones X para av1
    range_y = (0.0, 10.0)   # Rango de posiciones Y para av2

    if numCarsN != 0
        for _ in 1:numCarsN
            if first
                pos_y = rand(range_y[1]:0.5:range_y[2])  # Rango para av2
                add_agent!(Car, model; pos = (rand(13:14), pos_y), vel=SVector{2, Float64}(0.0, 0.1), street = av2, orientation = right)
                first = false  # Ya no es el primer auto
            else
                pos_y = rand(range_y[1]:0.5:range_y[2])  # Rango para av2
                add_agent!(Car, model; pos = (rand(13:14), pos_y), vel=SVector{2, Float64}(0.0, 0.1), street = av2, orientation = right)
            end
        end
    end
    if numCarsO != 0
        first = true
        for _ in 1:numCarsO
            if first
                pos_x = rand(range_x[1]:0.5:range_x[2])  # Rango para av1
                add_agent!(Car, model; pos = (pos_x, rand(7:8)), vel=SVector{2, Float64}(0.1, 0.0))
                first = false  # Ya no es el primer auto
            else
                # Añadir auto en av1 (horizontal)
```

```
            agent.street = av1
            changing = true
        end
    end
    first = true
    range_x = (5.0, 20.0)  # Rango de posiciones X para av1
    range_y = (0.0, 10.0)   # Rango de posiciones Y para av2

    if numCarsN != 0
        for _ in 1:numCarsN
            if first
                pos_y = rand(range_y[1]:0.5:range_y[2])  # Rango para av2
                add_agent!(Car, model; pos = (rand(13:14), pos_y), vel=SVector{2, Float64}(0.0, 0.1), street = av2, orientation = right)
                first = false  # Ya no es el primer auto
            else
                pos_y = rand(range_y[1]:0.5:range_y[2])  # Rango para av2
                add_agent!(Car, model; pos = (rand(13:14), pos_y), vel=SVector{2, Float64}(0.0, 0.1), street = av2, orientation = right)
            end
        end
    end
    if numCarsO != 0
        first = true
        for _ in 1:numCarsO
            if first
                pos_x = rand(range_x[1]:0.5:range_x[2])  # Rango para av1
                add_agent!(Car, model; pos = (pos_x, rand(7:8)), vel=SVector{2, Float64}(0.1, 0.0))
                first = false  # Ya no es el primer auto
            else
                # Añadir auto en av1 (horizontal)
                pos_x = rand(range_x[1]:0.5:range_x[2])  # Rango para av1
                add_agent!(Car, model; pos = (pos_x, rand(7:8)), vel=SVector{2, Float64}(0.1, 0.0))
            end
        end
    end
    model
end

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

```
                pos_x = rand(range_x[1]:0.5:range_x[2])  # Rango para av1
                add_agent!(Car, model; pos = (pos_x, rand(7:8)), vel=SVector{2, Float64}(0.1, 0.0))
            end
        end
    end
    model
end

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