Data structure, synchronization

So for my data structure, I have:

Vehicle that contains:

- Id (identify vehicle): so each car is unique
- Type (Car or Truck): each has a different assigned weight
- direction of the vehicle (North or South): Direction of the vehicle that I need to go
- Conditional Variable for each of them: so you can wake them up in the correct order inside the queue

Global Variables represent:

- Bridge Weight: total weight of all vehicles currently on the bridge
- Current direction of flow (North or South): the active traffic direction on the bridge, so that the other flow can't go onto the bridge
- Consecutive count (so no more than 6 consecutive allowed): number of consecutive vehicles that have crossed in the current flow, so that yielding is enabled
- Vehicles On Bridge: list of vehicles currently on the bridge.
- North Queue (waiting queue, North Side): preserving arrival order.
- South Queue (waiting queue South Side): preserving arrival order.

For synchronization:

- Mutex lock to protect all shared state
    - Used to protect all shared state
    - prevents race conditions and ensures that updates are atomic
- Condition Variables:
    - Each vehicle uses its condition variable to block until it is eligible to cross. This allows fine-grained control so that vehicles wake up in queue order and only when constraints (weight, flow, fairness) are satisfied.
- Wait group for goroutine when creating a vehicle:
    - Used to wait for all vehicle goroutines to complete before the program terminates. This ensures that the simulation runs to completion even though vehicles are crossing concurrently.

High-level overview of the overall logic

Vehicles are created based on group: Each group can have delays attached to it, so they can arrive later/at a specific time. Vehicle created at random, random type, and random direction based on the user's want probability.

Each vehicle is a goroutine, so they go concurrently to each other, and after all goroutines are done, the program finishes.

So when a vehicle is created, it goes through 3 phases: Arrive, Cross, Leave the bridge.

In Arrive: They will get added to the waiting North or South Queue based on their direction. They will then be checked to see if they are allowed to cross the bridge or not. They are allowed if:

- They didn't make the weight of the bridge > 750 units
- If the traffic flow is the same as their direction
- If they are the 1st one in the queue
- If they are not the 7th or more consecutive of the same flow and the opposite direction has someone waiting to cross (they have to yield)

Else: They got blocked until they got woken up when they are allowed to cross (all conditions meet when woken up)

In Cross: This means that the vehicles are allowed to cross. Remove the vehicle from the head of 1 of the 2 queues (If they are crossing, they are at the head at the moment). Update bridge weight, add a vehicle on the bridge, add to consecutive (yield mechanism), and assign a flow if the bridge is previously empty. Then wait 2 seconds because that's the time it took to cross.

In Leave: This means that you finish crossing and are off the bridge now. So you remove it from the bridge, update the bridge weight (subtract it).
Then you implement reassignment of flow logic and wake-up logic here:

- If you are still allowed to cross due to consecutive, just keep going
- If the bridge is Empty (will happen due to the arrival condition whe the other side need to cross), and the consecutive number is over 6 (you need to yield to the opposite lane), and there's a vehicle waiting:
    - You switch flow, and revert consecutive to 0, and wake up the opposing vehicle in the other queue
- If consecutive is > 6 but the other lane has no vehicle, then continue crossing but yield the moment they come
- Our current flow has no more vehicles, even though consecutive lanes allow them to switch to other lanes and revert to consecutive lanes
- Or if no one is waiting at all, then just reset the flow and consecutive