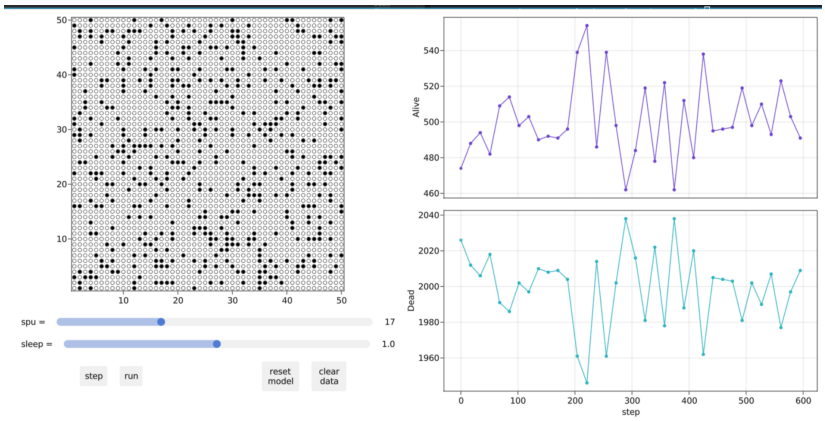
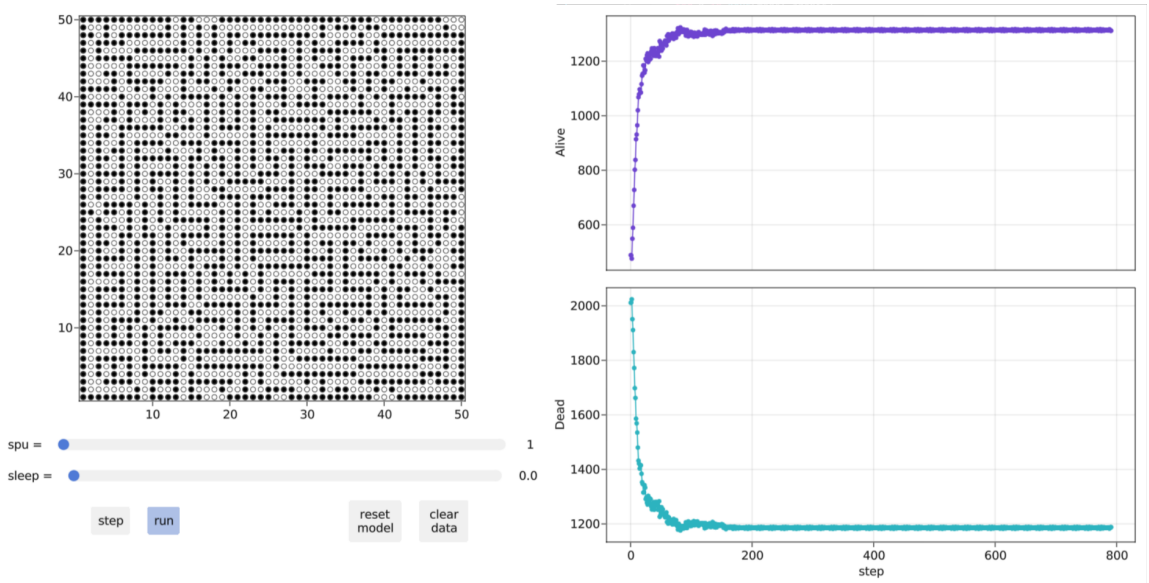
Initial Run



Working Step function

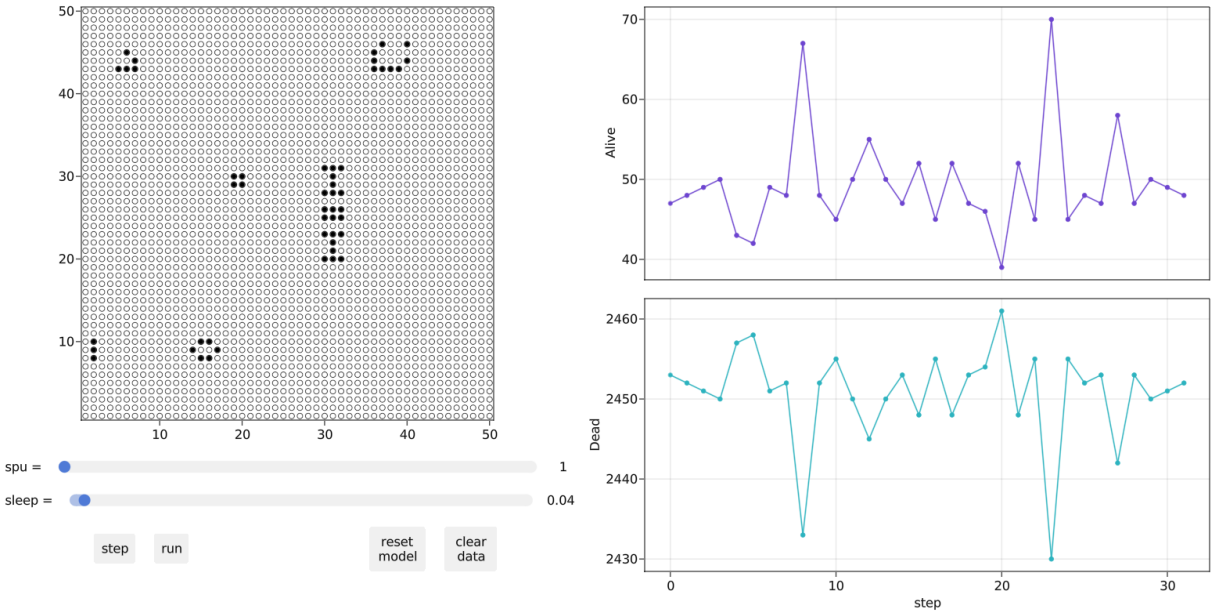
```
function step!(model::Agents.ABM)
    new_status = fill(false, Agents.nagents(model))
    for k in keys(model.agents)
        agent = model[k]
        n = sum(map((a) → a.status, Agents.nearby_agents(agent, model)))

        if agent.status
            if  (n <  2 || n > 3)
                new_status[agent.id]  = false
            else
                new_status[agent.id] = true
            end
        elseif !agent.status && (n == 3)
            new_status[agent.id]= true
        end
    end

    for k in keys(model.agents)
        model[k].status = new_status[k]
    end
end
```



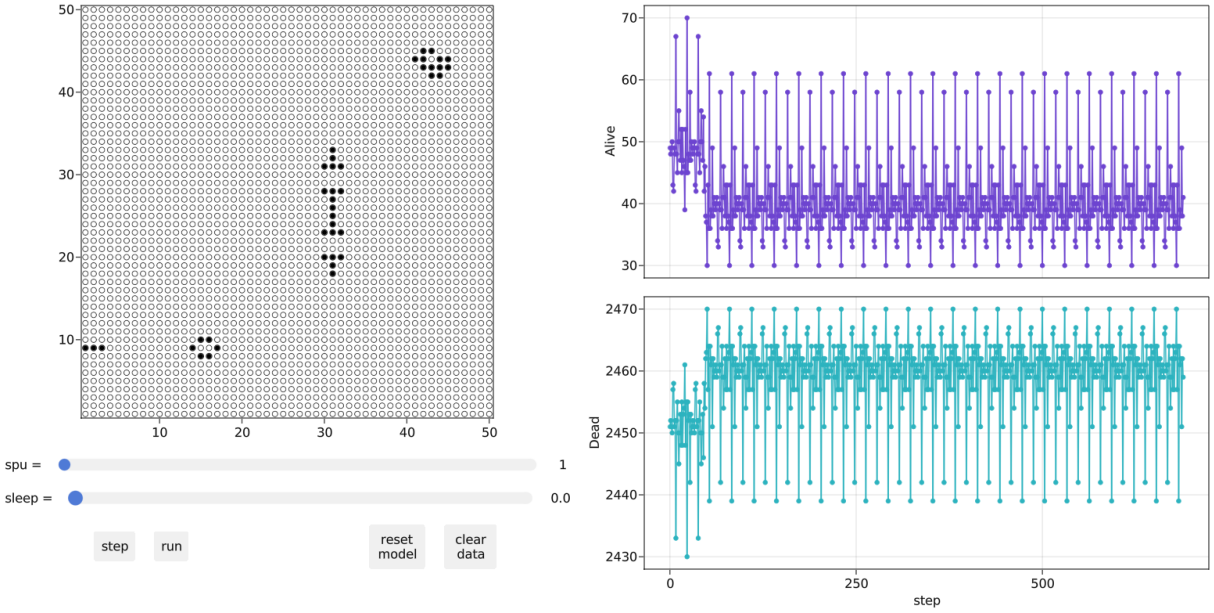
I found Julia pretty straightforward to work with. The most complicated part of the code is the visualization stuff, but that was provided so we didn't have to worry about it. Syntax wise, Julia is very similar to other high-level programming languages I've worked with like Python and Ruby. This made getting things up and running fairly intuitive. I was initially worried that the agents being stored in a 1-D array rather than a 2-D matrix was going to make it a bit more difficult to retrieve the neighboring agents, but then thankfully Agents provides a Agents.nearby_agents() function to handle this logic for you.

The time it took for the visualization to spin up the first time was a little frustrating, and made me unsure if it was working properly. However, it was really nice being able to simply rerun and it would re-render the visualization extremely quickly.

All The different shapes



All the shapes after ~ 500 iterations



The glider ran into the block and the annihilated each other :(