## Strip the physical proximity layer out and base everything on network connections

Currently:

function agent\_step!(agent, model)

    count\_neighbors\_same\_group = 0

    # For each neighbor, get group and compare to current agent's group

    # and increment `count\_neighbors\_same\_group` as appropriately.

    # Here `nearby\_agents` (with default arguments) will provide an iterator

    # over the nearby agents one grid point away, which are at most 8.

    for neighbor in nearby\_agents(agent, model)

        if agent.group == neighbor.group

            count\_neighbors\_same\_group += 1

        end

    end

    # After counting the neighbors, decide whether or not to move the agent.

    # If count\_neighbors\_same\_group is at least the min\_to\_be\_happy, set the

    # mood to true. Otherwise, move the agent to a random position, and set

    # mood to false.

    if count\_neighbors\_same\_group ≥ agent.seg

        agent.mood = true

    else

        agent.mood = false

        move\_agent\_single!(agent, model)

    end

    return

end

function model\_step!(model)

    for agent in allagents(model)

        #check whether the agent has a graph edge with its neighbours, and if not add an edge.

        for neighbor in nearby\_agents(agent, model)

            if has\_edge(model.social, neighbor.id, agent.id) == false

                add\_edge!(model.social, neighbor.id, agent.id)

            end

        end

    end

end

* Needs to be changed from “nearby\_agents” to find the number of *network* neighbours in the same group and compare to agent.seg
* Possible agent\_seg needs to be changed to a percentage proportion, because a small raw number will probably be washed out by a large number of graph connections
* Run with step!(model,agent\_step!,model\_step!)

## Build in a mechanism for pruning graph edges

* There are going to be too many graph edges if we run this for any significant number of steps