

# Schelling's Segregation Model in Second Life - Technical Details

Joel Dearden

Centre for Advanced Spatial Analysis, University College London

## Overview

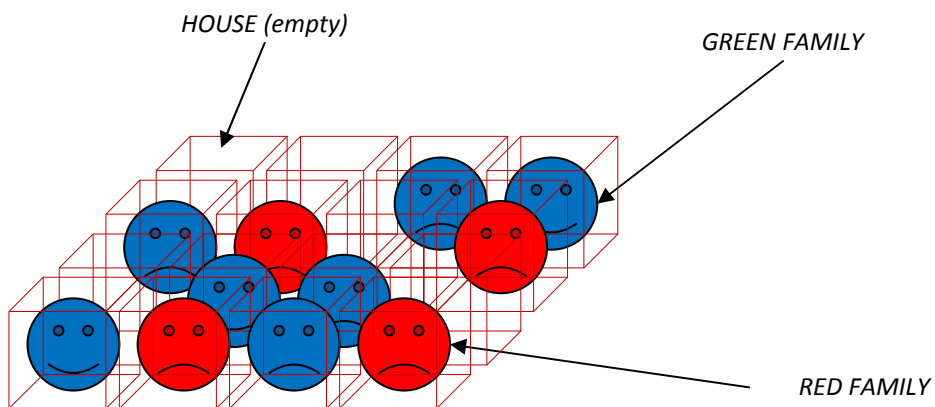
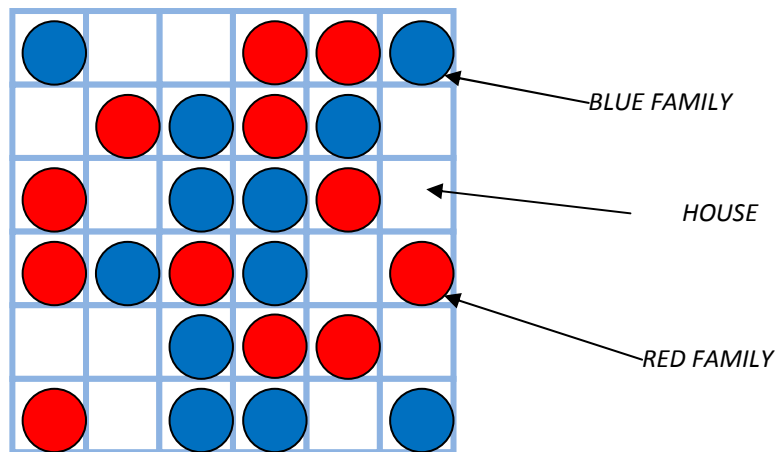
This document is a collection of rough notes about how the CASA's Second Life version of Schelling's Segregation model works.

**The notes were made as part of the development process and details were inevitably changed during implementation and bug fixing so there may be some differences between this document and the real code.**

## Rules

- Generate a grid of houses
- We have two types of agents: red and blue. Fill a third of houses with red and third with blue.
- This leaves a proportion of the available space empty
- Agents determine whether they are satisfied with their current location/house based on the colour of the agents in neighboring houses (defined by a 3x3 Moore neighborhood). We can set the percentage of the neighbors that an agent requires to be the same colour as itself to make it happy. This can be varied to give different results.
- The model is run and each time we update the model we prompt one agent to decide if it is satisfied with its current location. If the agent is happy with its current house it doesn't move. If the agent is unhappy with its current house it looks to all the other houses that are empty and moves to the nearest one that will make it happy.
- **If an agent can't find a house that makes it happy then it stays where it is**

## Mock-up



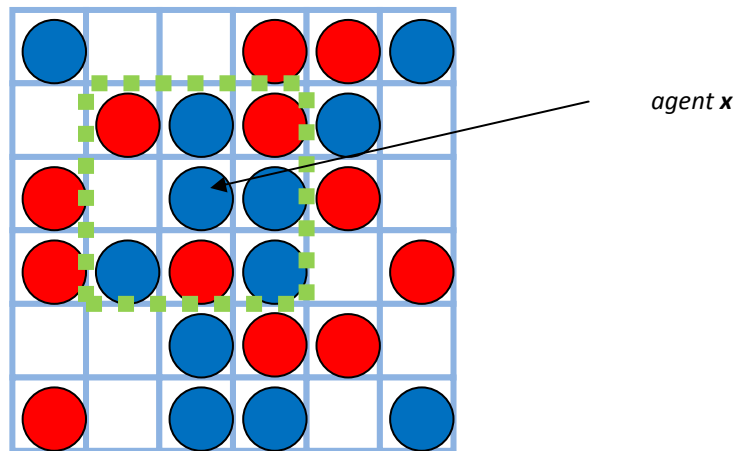
We can indicate which families are happy through colour, texture, glow or labeling.

### Preferences for neighbourhood composition

**Neighbourhood size:** this can be varied, e.g. immediate neighbours (eight total), neighbours two steps away (twenty four total), etc.

We will start with a neighbourhood that is only the immediate neighbours around an agent's house.

An agent is happy if the number of neighbours like itself is at least  $q$  percent. For example:



In the grid above we can set  $q$  to 25% we can check if agent  $x$  is happy or not. It is happy if at least 25% of its neighbours are the same colour as itself. If we count the neighbours we have:

- Three red
- Four blue

...giving a total of seven neighbours.

We can work out the percentage neighbours who are the same colour:

7 neighbours = 100%

1 neighbour =  $100 / 7$  %

4 neighbours =  $400 / 7$  % = **57.14 %**

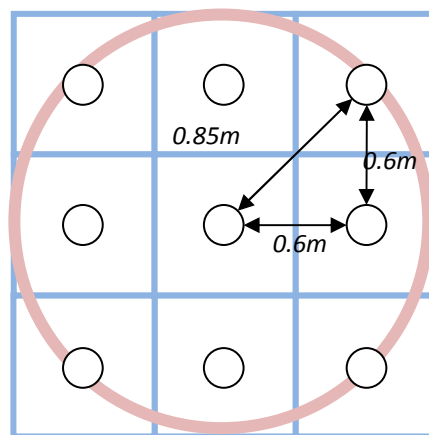
So the agent is happy – and does not want to move.

### Neighbors checking implementation

Red agents are called "red" and blue agents are called "blue". We can use *ISensor* once for each colour of agent with a set range to find out the number and colour of any neighbours, i.e.:

- *ISensor*: red
- *ISensor*: blue

Combine the results.



Given the size of the houses we need to set the range of the *ISensor* scan to 0.85m. This is shown in the diagram above.

### Triggering agent moves

The controller doesn't know anything about the state of the grid. How does it pick a random agent and get it to move?

The controller gives each agent a unique integer ID as the *on\_rez* parameter. It just loops through this contiguous range of IDs reading each one out in turn and waiting for a confirmation from each agent that it has done its thinking and moved if it was not happy.

The *on\_rez* parameter then can't be used to send the segregation preference parameter so this is broadcast through a chat message to all agents.

### Choosing a new house

Once an agent has decided to move it sends a chat message to all houses saying it wants to move. The chat message includes its colour so houses can check if they are suitable, e.g.:

*"look\_new\_house:red"*

All houses that receive the message check if they are empty.

If they are not empty they don't reply.

If they are empty then they scan the neighbourhood for red and blue agents. Only if the mix is suitable (e.g. for a red agent in this case) then they reply with *"new\_house\_ok: <vector pos>"*.

In a grid there will always be a fixed number of houses empty. E.g. in a 12x12 grid we will always have 48 free houses (i.e.  $144 / 3$ ). So an agent looking for a new house only needs to wait for 48 messages from houses to say either *"not\_ok"* or *"new\_house\_ok: <vector pos>"*. It then knows it has all the information it needs to choose the closest house that will make it happy. If it doesn't receive any *"new\_house\_ok: <vector pos>"* messages then it knows it can't move anywhere that will make it happy so it stays where it is.

Once it has calculated the closest house it uses the position to set its own position. It replies with the message *"accept\_new\_house: <vector pos>"* and so the chosen house knows it is not empty anymore. The old house is notified of the exit with the message *"abandon\_house: <vector pos>"*.

## Entities

Types:

- House
- Red Agent
- Blue Agent
- Controller

For a square grid  $n \times n$  we would have:

- $n^2$  houses
- $n^2 / 3$  Red Agents
- $n^2 / 3$  Blue Agents
- 1 controller

Which gives a total number of objects =  $1 + 5n^2/3$

So some example grids would be:

10x10 = 167 objects

