Daniela Vadillo, Elliot Delahaye

Github Repository:
https://github.com/mdvadillo/ABM-Assignment-1
Word count: 565

Segregation in Two Dimensions:
A Development of the Schelling Model

Model Background

The model extends the classic segregation model of Schelling. It shares the original purpose of exploring the different mechanisms that may give rise to segregation between groups but extending the problem to include an additional dimension to agent preferences. It is designed for social scientists interested in understanding these social patterns as well as policy-makers focused on housing or civil rights.

The model contains one type of agent, which are distinguished by two binary characteristics/types (in contrast to Schelling's single binary characteristic). These characteristics are passive in the agent behavior and only affects how agents view each other. The proportions of the groups with each of the first characteristic are determined by an adjustable proportion parameter whilst the proportion of these two groups with each of the second characteristic is determined stochastically in line with an adjustable probability parameter.

At each model step, agents calculate how happy they are at their current location, given their current neighbors. This happiness is a function of whether they share the characteristics of their neighbors where there is equal marginal utility to sharing either characteristic. If agents' happiness meets a given threshold, they stay put. Otherwise, they move to a random location on the grid in the next step. The simulation stops once a given proportion of agents are happy (this ensures that simulations with extreme intolerance parameters can complete).

Design Concepts

In each iteration, agents make a decision to move given their happiness level. Where Schelling defines happiness by the number of adjacent neighbors that are of their type, our model generalizes the definition by giving a score of 0.5 for each characteristic that is shared with a neighbor. Hence a neighbor that shares both characteristics will continue to increase happiness by 1 but two non-exactly matching neighbors will increase it by 0.5. Each agent faces the same threshold intolerance, and moves if the threshold of intolerance is not met. (i.e. if they do not have enough matching neighbors, in the combined characteristic space, they move) .
The simulation stops when the sum of happiness across all agents reaches the percentage of happiness specified by the user.

Daniela Vadillo, Elliot Delahaye

Initial location and relocation of agents is random. In the first dimension, we specify the number of agents we want in each group. In the second dimension, we specify a probability of belonging to the majority/minority.
(165 words)

Implementation Details

The model was implemented in Python, using the ABM package Mesa.

At the start of the simulation, each agent is placed randomly on the grid, with agent's again being randomly scheduled for their positioning turn. The number of agents, the proportion of agents with the first characteristic and the probability that each will have a value of the second characteristic is set by user-defined inputs that can be altered across simulations.

Daniela Vadillo, Elliot Delahaye

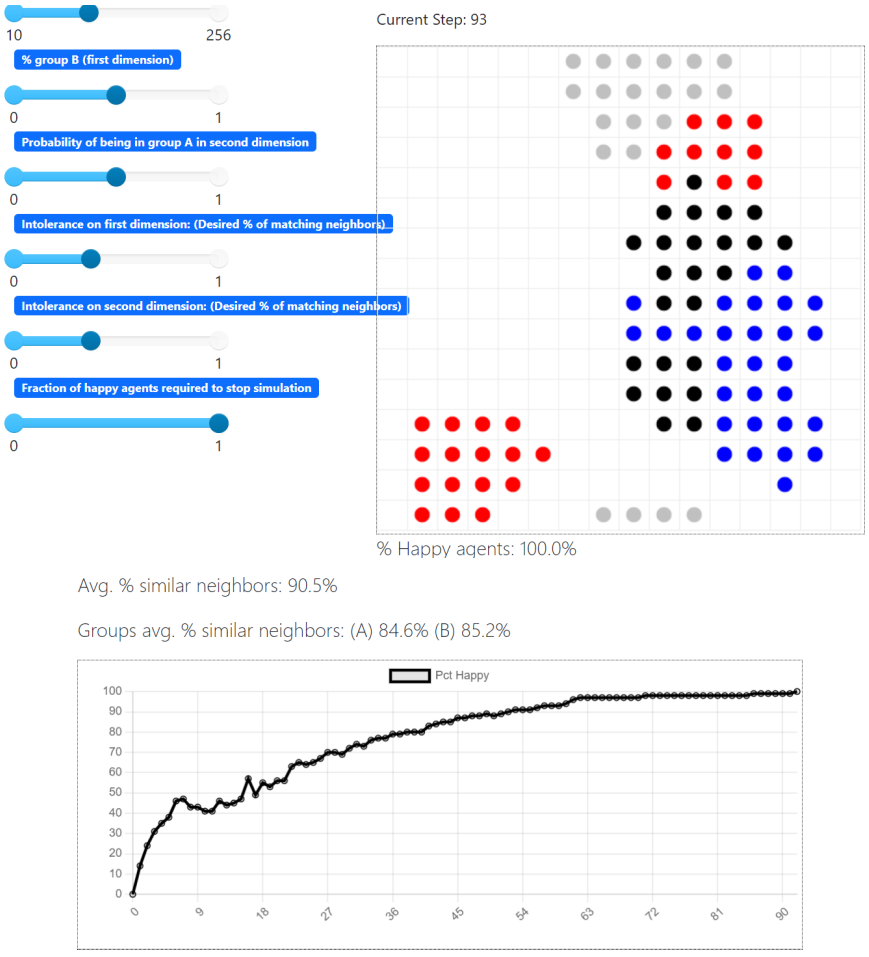
Results

As shown below in Figure 1, the changes lead to more complicated segregation patterns. A color-key is given in Table 1. It can be seen that the two opposite groups (Red-Blue), (Black, Grey), segregate fully, whereas the half-sibling agents (adjacent in Table 1), are happy to locate next to each other (whilst often clustering). This shows how more diverse groups can live together – trading off differences for similarities. If we make the agents too intolerant, happiness asymptotes at a level less than 100% – we implemented a parameter to set a stopping level of happiness to end the simulation in these odd cases.

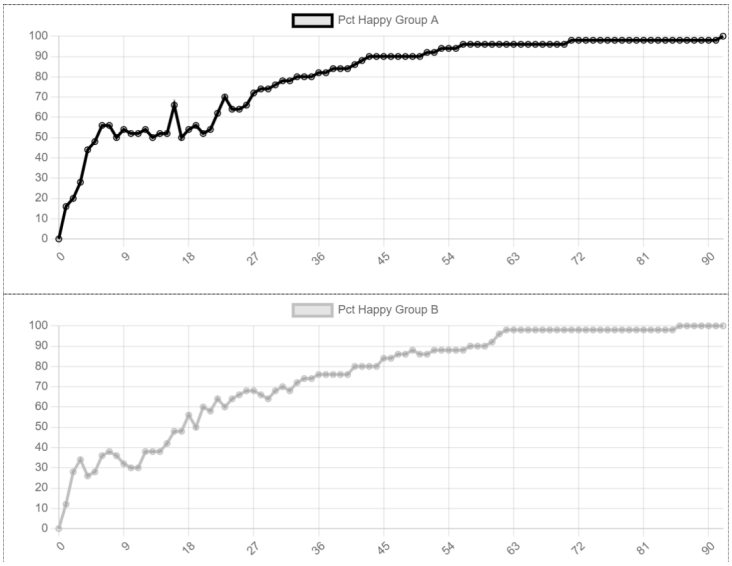
Table 1 - Taxonomy of Type Combinations

		Characteristic 2	
		0	1
Characteristic 1	0	Red	Grey
	1	Black	Blue

Figure 1 - A Simulation, Given Parameters



Daniela Vadillo, Elliot Delahaye



Daniela Vadillo, Elliot Delahaye

Appendix: Changes made to the model (code):

Agents.py

```
class SegAgent(Agent):
  def __init__(self, pos, model, agent_type_1, agent_type_2):  # agents and their characteristics
    super().__init__(pos, model)
    self.pos = pos
    self.type_1 = agent_type_1
    self.type_2 = agent_type_2
    self.similar = 0  # agent-specific measures of neighbor similarity
    self.similar0 = 0 #did not change this, still tracks similarity for type_1
    self.similar1 = 0
```

...

```
def step(self):
    self.similar = 0  # reset these counters each time step
```

...

```
# get neighbors and determine if your intolerance threshold is met
    for neighbor in self.model.grid.iter_neighbors(self.pos, True):
      self.neighbors_a += 1
      if neighbor.type_1 == self.type_1:
        self.similar += 1/2     # since we can be similar in two ways, weigh both markers (equal weights).
```

...

```
      if neighbor.type_2 == self.type_2: # look at second dimension of similarity, 50% weight of total similarity
        self.similar += 1/2          # adding additional if statement to not mess group tracking of first dimension


          if self.type_1 == 0:
      if self.similar < 8 * 1/2 * (self.model.intolerance_1 + self.model.intolerance_2):
        self.model.grid.move_to_empty(self)
```

calculate happiness just as before

```
    else:
      if self.similar < 8 * 1/2 * (self.model.intolerance_1 + self.model.intolerance_2):
        self.model.grid.move_to_empty(self)
```

Model.py

```
# set up the model and initialize the world
class SegModel(Model):
```

```
height = 16
width = height


# adding agents to the world
def __init__(self, width, height, num_agents, minority_pc_1, minority_pc_2, intolerance_1, intolerance_2, stopping_level):
    self.num_agents = num_agents  # we're allowing these values to be set at each run
    self.minority_pc_1 = minority_pc_1
    self.minority_pc_2 = minority_pc_2
    self.intolerance_1 = intolerance_1
    self.intolerance_2 = intolerance_2
    self.stopping_level = stopping_level


#### left the rest untouched, just added these new parameters
```

```
### Here we show how we selected the type of the second dimension
for i in range(self.num_agents):
        if i < self.num_agents1:
            self.agent_type_1 = 1
        else:
            self.agent_type_1 = 0


        # selecting the type for our new parameter
        self.agent_type_2 = np.random.binomial(n = 1, p = self.minority_pc_2)


        x = self.random.randrange(self.grid.width)
        y = self.random.randrange(self.grid.height)


        agent = SegAgent(i, self, self.agent_type_1, self.agent_type_2)
```

```
# And we changed the stopping condition to avoid infinite simulations with steady states where minorities keep moving but majorities are happy
if self.happy >= self.stopping_level * self.schedule.get_agent_count():
        self.running = False
```

Daniela Vadillo, Elliot Delahaye

Server.py

changed color scheme to account for new agent types

```
def schelling_draw(agent):
  if agent is None:
    return
  portrayal = {"Shape": "circle", "r": 0.5, "Filled": "true", "Layer": 0}


  if agent.type_1 == 0 and agent.type_2 == 0:
    portrayal["Color"] = "silver"
  elif agent.type_1 == 1 and agent.type_2 == 0:
    portrayal["Color"] = "red"
  elif agent.type_1 == 0 and agent.type_2 == 1:
    portrayal["Color"] = "blue"
  else:
    portrayal["Color"] = "Black"


  return portrayal
```

Added more user-settable parameters (intolerance level for both dimensions, fraction of minorities in both dimensions, and fraction of happiness required to stop the simulation)

```
model_params = {
  "height": SegModel.height,
  "width": SegModel.width,
  "num_agents": UserSettableParameter('slider', "Number Agents",
                          int(0.8 * SegModel.height ** 2), 10,
                          SegModel.height * SegModel.width, 10),
  "minority_pc_1": UserSettableParameter('slider', "% group B (first dimension)", 0.35, 0.00, 1.0, 0.05),
  "minority_pc_2": UserSettableParameter('slider', "Probability of being in group A in second dimension", 0.35, 0.00, 1.0, 0.05),
  "intolerance_1": UserSettableParameter('slider', "Intolerance on first dimension: (Desired % of matching neighbors)",
                            0.375, 0, 1, 0.125),
  "intolerance_2": UserSettableParameter('slider', "Intolerance on second dimension: (Desired % of matching neighbors)" ,
                            0.375, 0, 1, 0.125),
  "stopping_level": UserSettableParameter('slider', "Fraction of happy agents required to stop simulation" ,
                            1, 0, 1, 0.1),
```