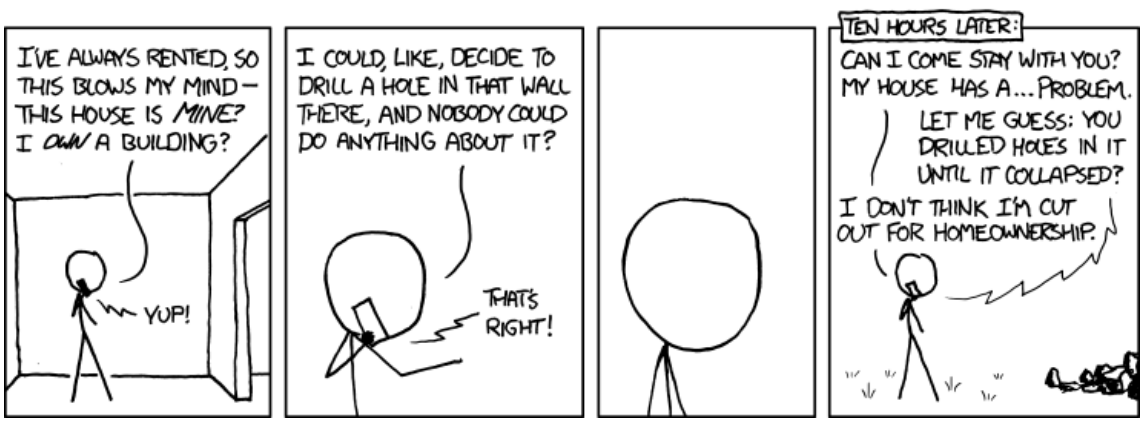
Case Study: Income Maps

CITS4403 - Computational Modelling Assignment

Mitchell Pomery (21130887)

May 20, 2015



xkcd.com/905

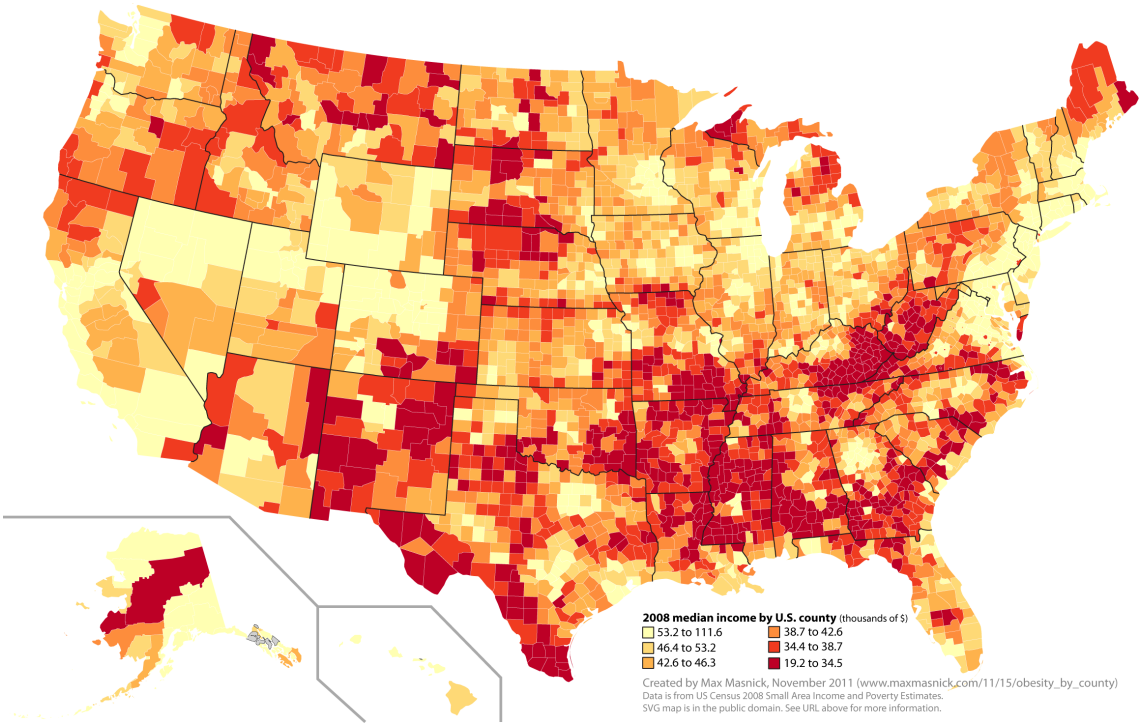
1 Introduction

Imagine you've just been given a massive raise and now earn significantly more than your neighbors. Is it be worth renovating your current property or should you move to a new place? How do expensive suburbs filled with people who earn more form?

Below is an infographic of the median household income per county in the United States. You can see that the high income areas are clumped together in groups with the lower income areas surrounding them.

There are many possible reasons for this clumping. A good way to look at a investigate these possible reasons is through the use of agent based models[1]. These models simulate simple behavior of agents in a simple environment to achieve a similar pattern to the real world.

Simulating population movements using agents is nothing new. Thomas C. Schelling's "Dynamic Models of Segregation", published in 1971, was among the first to do this. His model had two agents that both behaving the same way. They would move if they were "unhappy" because they were outnumbered by the other agent.



2 Implementation

To simulate a city with people moving around it we needed to create the environment and the agents. The simulation has been implemented in `income.py` and is based off of `rentals.py`, an incomplete agent based model. The model is of people moving around if their incomes are too different from the people in their neighborhood.

2.1 Named Tuples

In all programming, making the code easy to read and understand is important. One way to do this is through the use of Python's named tuples. They allow the creation of unchangeable tuples with properties that can be accessed by name instead of index. They easy to use and make code significantly more readable. The main named tuple used in the simulation is Coordinates, which are passed between several objects. Named tuples make it possible to refer to the x and y coordinates as `Coordinates.x` and `Coordinates.y` instead of `Coordinates[0]` and `Coordinates[1]`.

```
import collections

# Define a named tuple
Coordinates = collections.namedtuple('Coordinates', 'x y')
# Create a tuple
location = Coordinates(4, 5)
print(location.x) # Will print 4
print(location.y) # Will print 5
```

2.2 City

The `City` object keeps track of where every person is and who live in their neighborhood. Determining who lives in a neighborhood is achieved by iterating through all cells around the node and returning the coordinates of any cell that contains another person.

```
def get_neighbourhood(self, coords):
    neighbours = []
    for x in range(coords.x - self.neighbourhood_size, coords.x +
   self.neighbourhood_size + 1):
        for y in range(coords.y - self.neighbourhood_size,
   coords.y + self.neighbourhood_size + 1):
            if coords.x != x and coords.y != y:
                neighbour_coords = Coordinates(x, y)
                neighbour = self.get_house(neighbour_coords)
                if neighbour != None:
                    neighbours.append(neighbour)
    return neighbours
```

2.3 People and Population

`People` in the simulation share properties with real life people, but have everything unimportant to the simulation stripped away. They have an income, know what city they live in and know what their address is. Everything else such as name, age, gender, height, life goals, ambitions and sentience is unimportant to the simulation and so is left out. In agent based modeling you do not need to replicate every property of the being you are investigating, and can choose what you need.

Managing all of the `People` is the responsibility of the `Population` class. Each step `Population` gets a list of all the empty houses and all the people who want to move and moves as many people as it can to a new random position.

2.4 CityPrinter

`CityPrinter` is used to create a colour heat map of the incomes of the population and where they live. It uses the `wx` Python Package to create the graphics. To create this image, a colour is generated for every cell on the map that a person lives on, which is then added to a canvas. Once every point has been added to canvas, the frame is shown and the application runs until the frame is closed.

```
def create_heatmap(self):
    min_income = self.population.min_income
    max_income = self.population.max_income
    print("Lowest Income: " + str(min_income))
    print("Highest Income: " + str(max_income))

    for house in self.city.houses:
        x = house.x - self.city.size / 2
        y = house.y - self.city.size / 2
        person = self.city.get_house(house)
        if person != None:
            col = self.color(person.income, min_income, max_income)
            self.canvas.AddPoint((x, y), Color = col)

    self.frame.Show()
    self.app.MainLoop()
```

Determining the colour for a specific income is done using the `color` method. This method approximates the sequential colour scheme from the Department of Geography at the University of Oregon[2]. It does this by transferring the value from the scale provided to the red, green and blue scales in the method. It returns an object that can be used right away in the heat map creation methods.

```
def color(self, value, min_val, max_val):
    # Approximating
       http://geog.uoregon.edu/datagraphics/color/Bu_10.txt on the
       fly
    red_range = (0, 0.9)
    green_range = (0.25, 1.0)
    blue_range = (1.0, 1.0)

    percentage_of_range = 1 - (value - min_val)/(max_val - min_val)

    red = (((red_range[1] - red_range[0]) * percentage_of_range) +
       red_range[0]) * 255
    green = (((green_range[1] - green_range[0]) *
       percentage_of_range) + green_range[0]) * 255
    blue = (((blue_range[1] - blue_range[0]) *
       percentage_of_range) + blue_range[0]) * 255

    return wx.Colour(red, green, blue, 1)
```

3 Exercises

Below are a list of exercises that will help you to explore `income.py`, understand how it works and extend it.

Exercise 1: *Obtain a copy of* ***`income.py`*** *and run it. Modify it so that it runs for 100 steps and compare the two outputs. Be patient as it will take a while to run.*

Exercise 2: *Run the program with a different city size using the command line arguments and compare the outputs.*

Exercise 3: *Determine the Big-O notation for each step.*

Exercise 4: *Give people an income at random, weighted so that some people are low income earners, most people are medium income earners and a few people are high income earners.*

Exercise 5: *Determine if using '@property' causes significant performance degradation by removing it from classes where possible.*

Exercise 6: *Modify the city so that there is a hole in the center. Run the simulation and see how this changes the output.*

4 Analysis

Choosing to attempt to implement the complex social system that is rental properties and a society that lives in them was possibly not the best idea. In the real world there are several factors that affect the price of rental properties such as age of the

house, and distance to public transport, shopping centers, workplaces, schools and major population centers. People choose houses based on more than how far away it is from their current place and how much the rent is, as they look for community groups, where their friends and family live, and how many bedrooms, bathrooms and toilets it has.

Some fatal flaws exist in the simulation, such as the fact that rental properties can be driven all the way down to $0. However, the simulation does display some interesting things however. High cost rental properties start to cluster and get filled by high income earners. The further away you move from these clusters, the lower the rent and incomes become.

One way to simplify and improve the simulation is to remove the aspect of rental prices altogether and have people moving based one income. A mock up of this idea can be seen in `income.py`.

Bibliography

[1] Agent-based model - Wikipedia, the free encyclopedia. 2015. Agent-based model - Wikipedia, the free encyclopedia. [ONLINE] Available at: http://en.wikipedia.org/wiki/Agent-based_model. [Accessed 19 May 2015].

[2] Data Graphics Research. 2015. Data Graphics Research. [ONLINE] Available at: http://geog.uoregon.edu/datagraphics/color_scales.htm. [Accessed 19 May 2015].