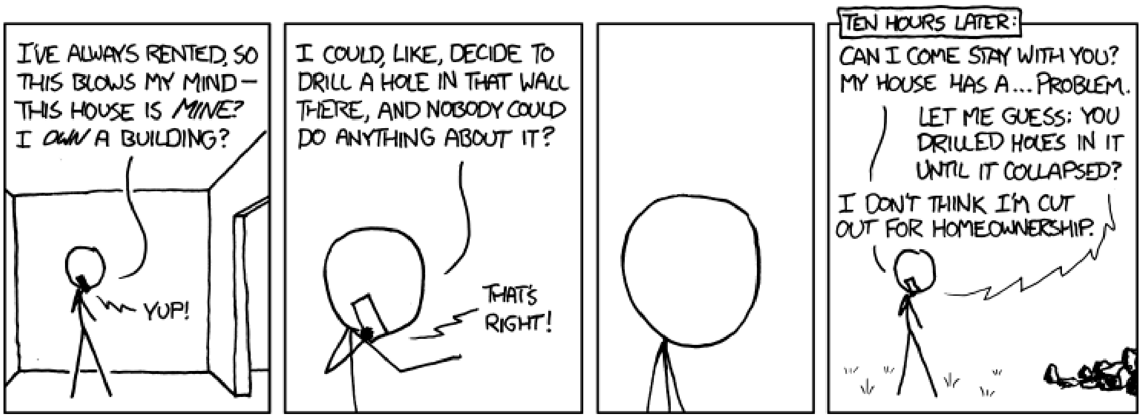
Case Study: Rental Prices

CITS4403 - Computational Modelling Assignment

Mitchell Pomery (21130887)

May 19, 2015


I'VE ALWAYS RENTED, SO THIS BLOWS MY MIND— THIS HOUSE IS MINE? I OWN A BUILDING?
YUP!
I COULD, LIKE, DECIDE TO DRILL A HOLE IN THAT WALL THERE, AND NOBODY COULD DO ANYTHING ABOUT IT?
THAT'S RIGHT!
TEN HOURS LATER:
CAN I COME STAY WITH YOU? MY HOUSE HAS A... PROBLEM.
LET ME GUESS: YOU DRILLED HOLES IN IT UNTIL IT COLLAPSED?
I DON'T THINK I'M CUT OUT FOR HOMEOWNERSHIP.

xkcd.com/905

1 Introduction

Finding a place to live in while renting can be very difficult. Expensive places all seem to be grouped together, right next to each other and all occupied while cheaper places are on the outskirts of town and relatively empty. There are many reasons why this could be, A good way to look at a possibly reason as to why this happens is by using agent based models[1]. These models simulate the behavior of agents in an environment. This allows us to see the overall effect of this behavior on the system.

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.

Simulating a city of people moving around to houses with changing rent prices is harder to implement. Several variables such as rental price, occupancy and rent history need to be maintained, and a system needs to be created to determine who is going to move into each house at the end of each month, rather than randomly reassigning them.

2 Implementation

To simulate a city with people moving around it we needed to create two agents, the houses themselves and the people who live in them. We then create a city for these houses to be built in and a population that live in this city. These two groups then interact with each other and act depending on the state of the other. At the end of the simulation there are two things we can look at, rental prices and income distribution. The simulation has been implemented in `rentals.py`.

2.1 Named Tuples

Since the system we are creating is complex, making the code easy to read and understand is important. One way to do this is through the use of pythons 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. It increases code readability by making it possible to refer to the x and y coordinates as `x` and `y`.

```
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 House and City

The `City` object manages all of the `House` objects. `House` objects are a simple data structure that holds a price and an occupant, while `City` implements all of the logic.

Every house belongs in `City` and is created by it with a default price. The methods do not pass `House` objects back to the caller but instead return coordinates. These coordinates can then be access by calling the `get_house` method.

At the start of every step the prices of each unoccupied rental property are updated to try and make the houses look as affordable as possible to anyone who might move. At the end of each step the price of each occupied rental that meets certain conditions is increased by a fixed amount.

2.3 People and Population

`People` in the simulation share similar properties to people in real life. They have an income, they pay rent every month and they can only leave a rental agreement at the end of it's term. However they are also dissimilar to real people, as they can't share houses, don't ever get pay increases and don't have sentience. Fortunately with agent based modeling you do not need to replicate what you are investigating perfectly, and can choose what you need.

Managing all of the `People` is the responsibility of the `Population` class. Each step it checks to see who out of the people who can move is unhappy about their current situation, and moves as many people as it can to unoccupied houses.

2.4 CityPrinter

`CityPrinter` creates colour heat maps of both rental prices and incomes using the `wx` Python Package. The methods for both of these heat maps are very similar. for the rental price heat map, price for each house is retrieved, a colour is generated for it and then it is added to a canvas. After every house has

```
def create_rent_map(self):
    min_price = self.city.min_price
    max_price = self.city.max_price
    for house in self.city.houses:
        x = house.x - self.city.size / 2
        y = house.y - self.city.size / 2
        col = self.color(self.city.get_house(house).price,
           min_price, max_price, \
                self.city.get_house(house).occupant == None)
        self.canvas_rent.AddPoint((x, y), Color = col)
    self.frame_rent.Show()
```

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, max, red=False):
  if not red:
      red_range = (0, 0.9)
      green_range = (0.25, 1.0)
      blue_range = (1.0, 1.0)
  else:
      red_range = (1.0, 1.0)
      green_range = (0, 0.9)
      blue_range = (0.25, 1.0)

  percentage_of_range = 1 - (value - min)/(max - min)

  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 `rentals.py`, understand how it works and extend it.

Exercise 1: *Obtain a copy of `rentals.py` and modify it so that it runs for 100 years. Be patient as it will take a while to run. Change it back to 10 years when you are done.*

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.

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.

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].