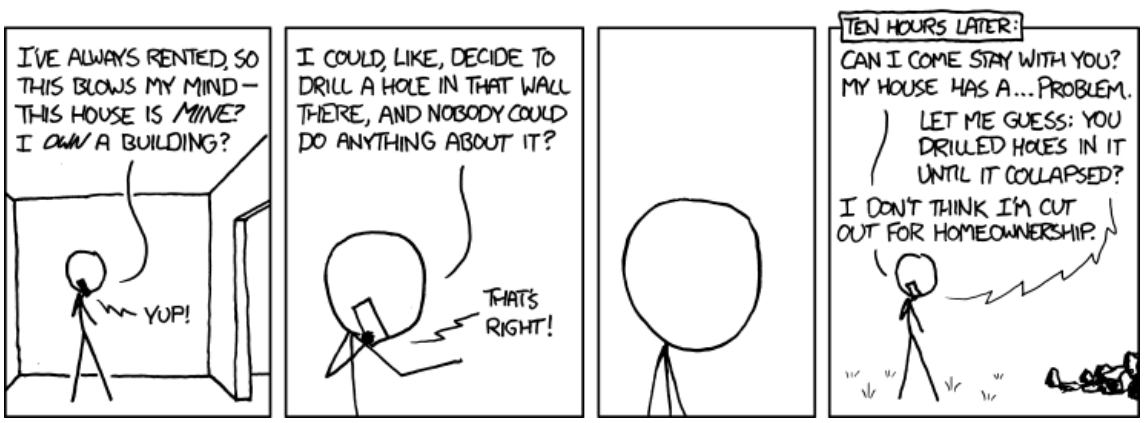
Case Study: Rental Prices

CITS4403 - Computational Modelling Assignment

Mitchell Pomery (21130887)

May 19, 2015



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. These models simulate the behavior of agents in an environment. This allows us to see the overall effect of this behavior on the system.

http://en.wikipedia.org/wiki/Agent-based_model

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 need 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.

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)
```

```
# Will print 4
print(location.x)

# Will print 5
print(location.y)
```

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.

Population

- Manages all the people
- Makes sure they are happy
- Make sure they move if they aren't
- Each step updates rent of old houses, moves people, then updates rent of occupied houses

3 Exercises

Exercise 1: *Randomly give people an income based off of Bureau of Statistic data.*

Exercise 2: *Modify the program to take city size and population size as arguments. See what happens when you increase and decrease the population.*

Exercise 3: *Modify the program so that there is a hole in the center of the City. Experiment with different City and population sizes and see how it changes.*