Xiaofan Liang
Minerva Schools at KGI
Capstone Project Status

Index[1][2][3]

[1] #organization: organize my project status in the form of a draft capstone paper and break down different components for clear communications.
[2] #professionalism: follow APA format for the paper and carefully presented my work with articulations and details.
[3] #qualitydeliverables: 28 pages of report with substantial literature review, model implementation, and data analysis.

Introduction[4]

The spatial distribution of a city's population has long been the focus for urban sociologists, geographers, and economists. The factors that determine population distribution can be complex and interconnected. Understanding where different income groups, especially urban poor, are located in cities allows us to identify areas that have high social risks and design policies accordingly. In my capstone, I want to explore using agent-based modeling to simulate the residential dynamics of different income population given the economic factors (e.g., land price, housing price, density, etc.) in a monocentric city and experiment with the mechanisms that drive spatial differentiation of different income groups. Furthermore, I want to experiment whether differentiating low-income population into the subtle subgroups (e.g., urban migrants, college students, and local poor) will lead to spatially different residential choices.

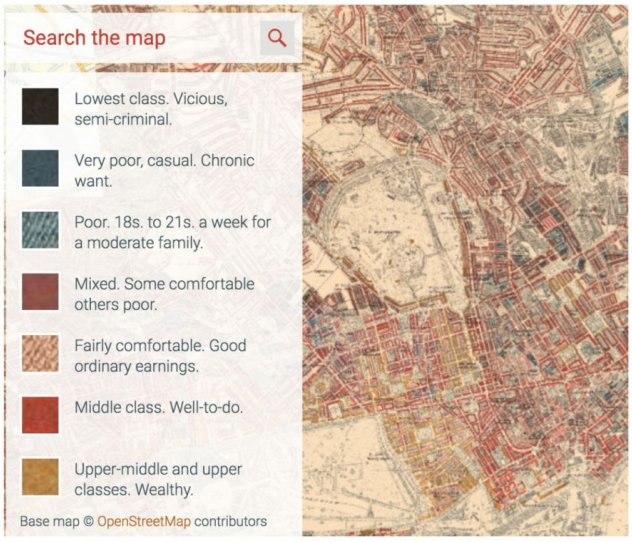
Previous Efforts to Model Residential Dynamics and Spatial Distribution of Urban Poor[5][6]

Urban Sociology and Economics Theories

Charles Booth's London Poverty map in 1889 was credited as the first modern map of social division and one of the earliest example of a large-scale social survey (Ball & Petsimeris, 2010). Without any commissions, Booth orchestrated decade-long efforts to construct the map as a comprehensive overview of poverty distribution in London. Every street in the map is colored

[4] #thesis: Formulate a well-defined thesis, justify its significance, and outline my capstone.
[5] #evidencebased: cited an abundance amount of literature to support the background and context of my work. Also cited literature for my model assumptions.
[6] #IL181_urbantheory: discussed urban theories that address my research question and outlined how my agent-based model may contribute to the field.

to reflect a different combination of employment and wealth. Booth's work influenced a generation of urban sociologists, the legacies of which include the establishment of Hull-House to record wages and nationalities data in Chicago, Du Bois's spatial examination of the black population in Philadelphia, and the spatial distribution of social characteristics in urban environments by researchers in the Chicago school (Vaughan, 2018). Burgess, who is often cited as the precursor of the Chicago school, theorized the spatial form of cities in a concentric model (1925). He identified the low-income neighborhoods as the "working men's zone" and "transition zone" that surrounded the centers of business development (CBDs).



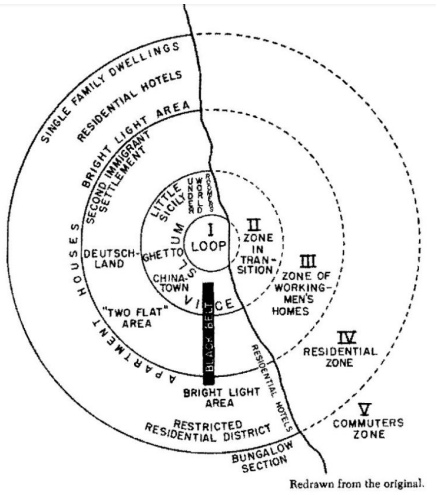
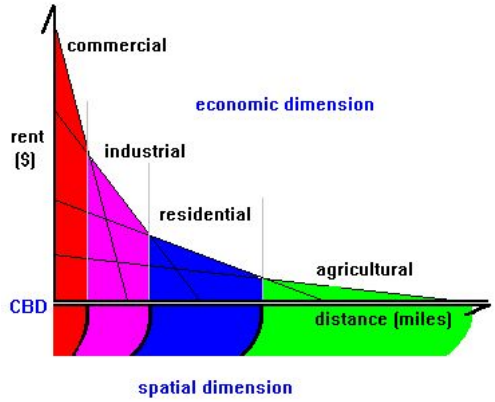


Chart 2—Urban areas.

Figure 1 (left): A Screenshot of Booth's Poverty Map on the LSE website (Map | Charles Booth's London, 2016).

Figure 2 (right): Illustration of Burgess's Concentric Model (Burgess, 1925)

Building on Burgess's monocentric assumption of cities, economists Alonso (1964) came up with the bit-rent curve (see Figure 3) that mathematicized the declining gradient of land prices, rents, and density away from the city center. Muth (1969) and Mills (1972) later used this distribution to predict the locations of various land uses and the living locations of diverse groups of people within the cities. Furthermore, Smith (1979) explained inner-city slums through the rent-gap theory that negative externalities (e.g., environment, industrial parks, degrading building values over time) can create concaves in the rent curve that makes certain locations close to city center affordable to the urban poor.



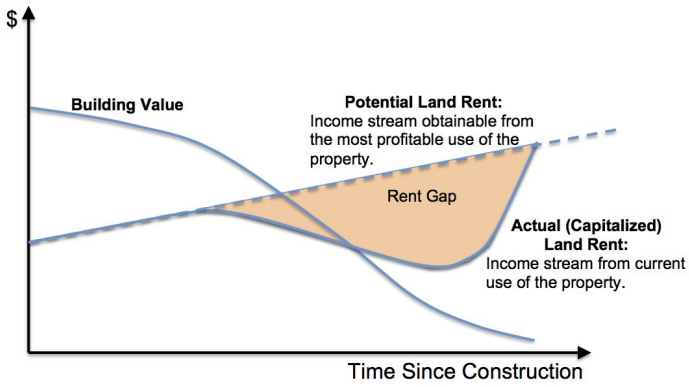


Figure 3: Bit-rent curve illustration (Alonso, 1964)

Figure 4: Rent-gap theory illustration (Smith, 1979)

However, the monocentric model of cities did not fully represent the realities of every case. For example, public transportation was not fully developed yet in Chicago when Burgess came up with his model. Hoyt (1939) complemented the Burgess model by proposing a fan-shaped model that emphasized the distribution of different income groups along transportation lines. Later on, the multi-nuclei model by Harris and Ullman (1945) further

attacked the mono-nuclei assumption in Burgess and Hoyt's work, and argued that the location of the low-income neighborhoods and industrial parks should be moved to the city periphery as top-down urban planning slowly came into place.

As cities experienced different phrase of development, the spatial distribution of its population can also be changed. In *Sprawl: A Compact History*, Bruegmann (2006) demonstrated the patterns of the density gradient of a city experiencing industrialization: at first, the population will heavily concentrated around city centers due to urbanization. As the city developed and the economic network and public transportation expanded, wealthier class tend to move toward the car-dependent suburbs, as a result of the urban sprawl.

Agent-based Modeling Approach to Residential Dynamics

The majority of Chicago School's models have treated the spatial forms of cities as a static phenomenon, while where people choose to locate is a mainly dynamic process. Agent-based modeling (ABM) provides an ideal playground to study such dynamic processes and experiment with parameter values to test out the effects of different interventions. Schelling's model of segregation (1971) first attempts to illustrate how complex segregated neighborhood can emerge from simple rules of choosing a location where one-third of the neighbors are of similar race. Benenson (1998) built upon Shelling's model and incorporated the dynamics of cultural dissonance and the cost of moving. More recent work tended to include more than one agent and populated the parameters with contextualized values. The multi-agents modeling of the slum formation (Patel, Crooks & Koizumi, 2013) and Liu's model of residential

dynamics in Haizhu, China (2010), in which each land patch has a vector containing metrics for education, transportation, public service, environment and so on, are examples in both directions.

However, most of the previous efforts have focused on the coarse-grained differentiation of agents (human) by income or ethnic identity and only a few have focused their narratives on the low-income neighborhood. In particular, they all assume the population is local or at most use population growth to brush over the existence of migrants in the cities. By far I have not found an agent-based model or multi-agent system that conceptualize the role of migrants and their residential choices in such frameworks. Will agglomerate spatial differentiation the specific groups of low-income population correspond to the spatial distribution of the entire low-income population? Will the subgroups of different low-income population distribute differently in space? Though some qualitative interviews and quantitative surveys have revealed the preliminary spatial difference for the subgroups (Zhong, 2017), the simulation method can provide a more dynamic explanation to the phenomena and thus worth exploring in my capstone.

Wu (2002) pointed out in his paper *Complexity and Urban Simulation: Towards a Computational Laboratory* that there are two kinds of agent-based models. One serves the function to conceptualize the model to build theories and explanations, while the other one is to calibrate the model according to real life data (very often GIS data) to make posterior predictions. My capstone aims to formulate the agent-based model aligning with the first type. The goal and the value is to provide a simulation framework that considers migrants and explores the residential dynamics within the seemingly homogenous low-income population, under the urbanization context in China.

Model Introduction

My model is conceptualized as a spatial agent-based model of a city in a confined space with people agents and patches interacting through space and time. The key model dynamics relies on affordability, job opportunities, and density as the key factors that motivate people to move their residences in the cities.

Human Agent

Every human agent has two functional attributes -- 1) income and 2) density sensitivity. Income is a categorical variable with values ranging from 1 to 3, representing the low, middle, and high level. Density sensitivity is a number arbitrarily assigned based on the income level. Rich people only move to places that they can occupy alone, while middle and poor people can tolerate three and five people at the same patch respectively.

Patch Agent

Each patch agent has three functional attributes -- 1) land price, 2) job probability, 3) density and 4) housing price. Similar to income, land price is also a categorical variable with values ranging from 1 to 3. Job probability is static at each patch with a fixed continuous value ranging from 0 to 1. The value is assigned when the model is initialized. Density is updated every step to calculated the number of human agents on top of the patch. Housing price is a composite variable, derived as the land price per person at a certain patch.

Model Assumptions[7]

The classical theories and prior modeling efforts in the previous section provide a basis for my assumptions regarding the justification and distribution of agent attributes. All of the variables mentioned (e.g., land price, housing price, density, etc.) are well-researched and traditional economic factors in the field of urban economics. The concentric distribution of land price and job probabilities is an assumption taken from the bit-rent curve. I have not found any literature directly support density sensitivity as a motivating factor for people yet, but I used density sensitivity as an indicator to represent the effects of other factors that are typically associated with density. For example, middle and upper-middle class people, in general, prefer independent housing and tend to live in car-dependent suburbs (Bruegmann, 2006), which corresponds to low density and thus can be translated as high-density sensitivity in my model. It is worth noticing that where people locate may vary in time, intensity, and context depending on the locations of the cities and which part of the life cycle they are experiencing. Recent reports in London observed a reversed suburbanization trend as the poor are increasingly pushed to the suburbs while the city centers are further gentrified (Jaffe, 2015). So far, my model has not considered transportation, and thus constrain the interpretation to an early period. Nonetheless, most of the assumptions still hold today, as many cities remain monocentric (e.g., Chicago) and the fundamental dynamic stays the same.

Programming Framework

[7] #IL181_urbanmodeling: combined with the literature section on agent-based modeling gap analysis, I discussed the advantages of ABM, my model assumptions, and the literature that support my assumptions.

I coded my agent-based model in Python with Mesa library (Masad & Kazil, 2015). Mesa is a newly developed Python library that provides a framework to do agent-based modeling in Python. Compared with NetLogo, programming in Mesa takes more time to build up the basic features of a model, but once it was built, it is easier to debug in Mesa, and it is more flexibility to customize the model as well as the visual outputs. For example, I was able to insert Javascript codes to Mesa framework so that three model canvas with different layouts can run at the same time, which is impossible in NetLogo. Moreover, Mesa can directly output data collected during the model runs as panda data frames which can be quickly connected to all the visualization and analytical tools in Python. To run the py or ipynb file, people need to install Mesa library, and the codes will automatically pop up a new browser window that shows the simulation.

Model Interface[8]

The model interface contains three main components: 1) model parameters, 2) model canvas (visuals), and 3) model metrics. Model parameters can only be set before the model starts running and changing the values while the model is running will not be factored into the model. The number of agents controls the amount of agents placed on the canvas and the rich, middle, and poor people are in an arbitrary 1: 3: 5 ratio. The three model canvas represent land price, job probability, and density respectively, and will update at the same time with the same model. Model metrics also update every step dynamically, calculating the percentage of unhappy (need to move) poor, middle or rich people within their groups.

[8] #ABMviz: Work beyond the library and wrote my own Javascript codes to adjust the visuals of model canvas. Uses seaborn library to make python matplotlib graphs more aesthetics.

The red color represents the poor, the yellow represents the middle-income group, while the green represents the rich, with the dot size decreasing accordingly. Therefore, if different types of agents overlap, they will be shown as layered circles.

Lighter colors refer to higher values, which can be higher density, job probability, or land price. As you can see, land price layout only have three color gradient, corresponding to the categorical nature of the variable. Job probability is continuous, and thus the layout shows more varieties in color.

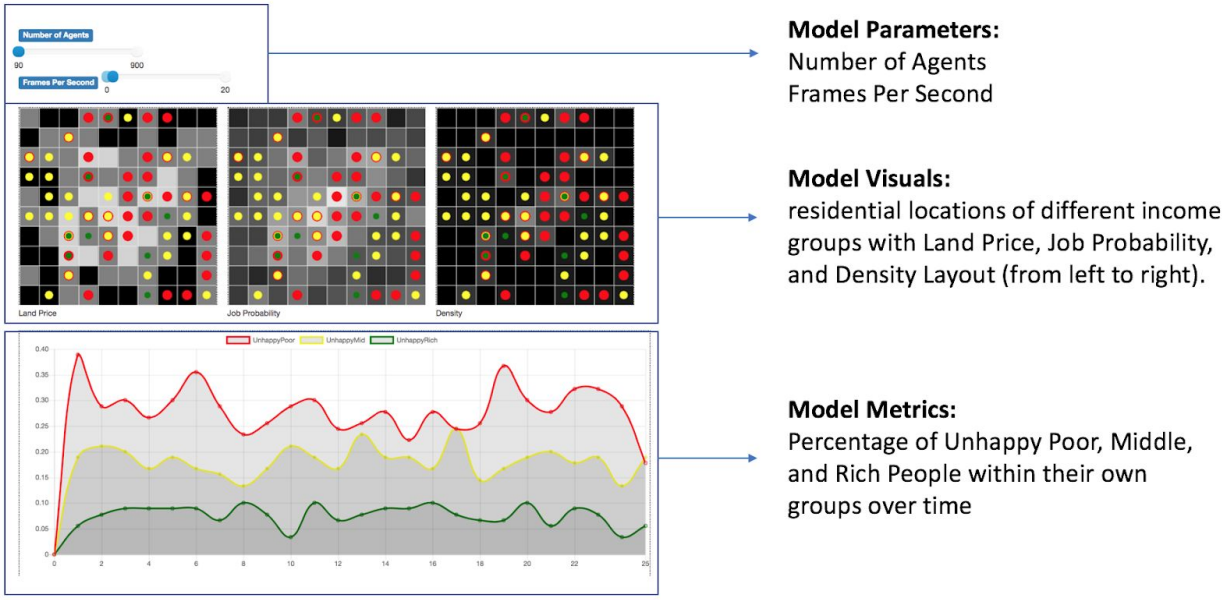


Figure 5: Model Interface

Model Logics and Behavioral Rules

At each time step, each human agent will be activated in random order and follow the decision tree to decide on its action (see Figure 6). The model will first evaluate whether the human agent is “happy”, which means its income can afford the patch’s housing price, the

density of the patch is lower than its threshold, and it can secure a job according to the patch's job probability.[9] If all these evaluations are satisfied, the human agent can stay at the current spot and wait for the next step. If not, then the human agent will pick a random neighbor within the radius of five and evaluate it against the same standard. If this random neighbor failed, then the human agents will choose another one until all neighbors ran out. In that case, if the human agent still cannot find an option within three steps, it will be removed from the model forever. If the human agent finds a satisfactory patch, it will move to the target patch and wait for all other agents to finish.

For patch agents, the model logic is simple: they only need to update the density after all agents have moved in each step.

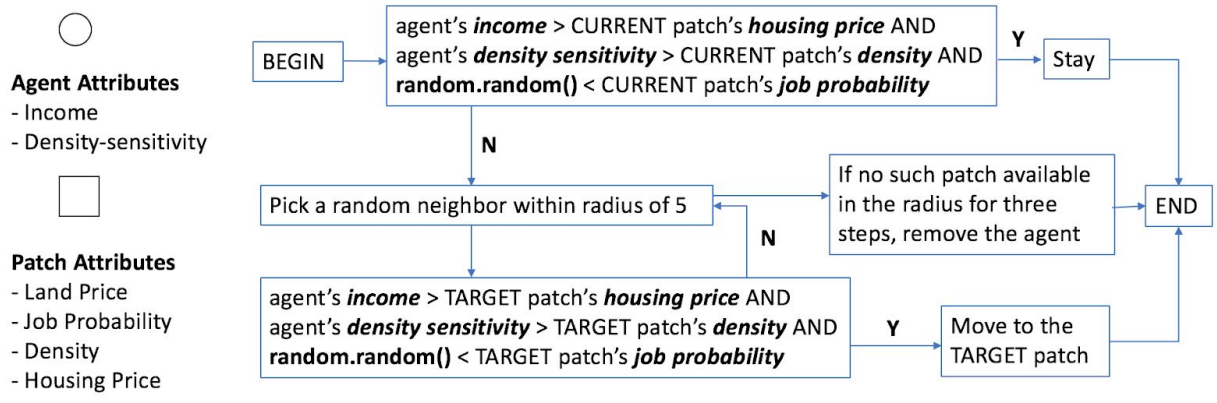


Figure 6: Decision tree diagram of the behavioral rule of one time-step.

[9] This part can be improved in the future because rolling a dice every step makes staying at one place really difficult. For example, even if the job probability of the patch is 0.8, which is very high, the chance to consecutively stay at the patch for five steps is only 0.8^5.

Model Metrics and Outcomes

To better understand how each factor impact model behaviors, I decided to add factors one by one and observe changes in model outputs. The four models below show the progression of model outcomes under different sets of variables, from simple to complex. The two key model metrics I discussed in this section are 1) the percentage of unhappy agents in each income groups over time, and 2) the percentage of agents in each income group over distances away from the city center. Due to time constraints, the second metric has only applied to the fourth model and may still contain bugs.

1) Model with only Income and Land Price

In the model with only income and land price as factors, agents will just move to one of the neighbors (within a radius of 5) that has affordable land price relative to their income. Since the searching range is large and no other mechanism is in place to stir the market, agents just search for land that they can pay for and find their best spots within one step (see Figure 7).

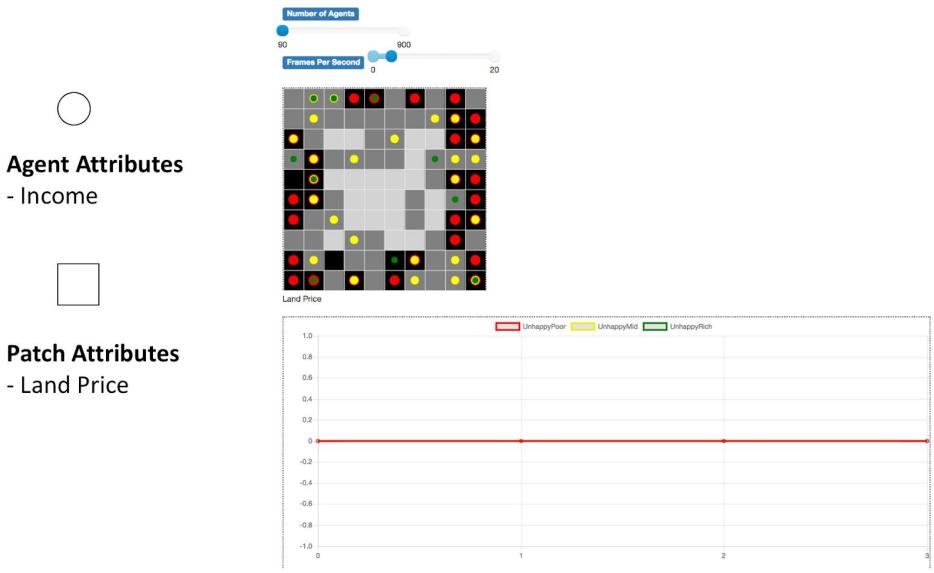


Figure 7: Model with only income and land price.

2) Model with Income, Job Probability, and Land Price Interaction

Adding job probability to the model destabilizes the interaction between income and land price and thus enables human agents to move beyond one step (see Figure 8). In the long run, we should expect the spatial differentiation of income groups. Rich people will move toward city centers as they can afford the price and have more job stability, while middle-income and poor people will occupy places further away from the CBD. This hypothesis can be confirmed (or disproved) quantitatively by running 100 iterations of models over a long period while tracking the changing percentages of rich people in the central patches.

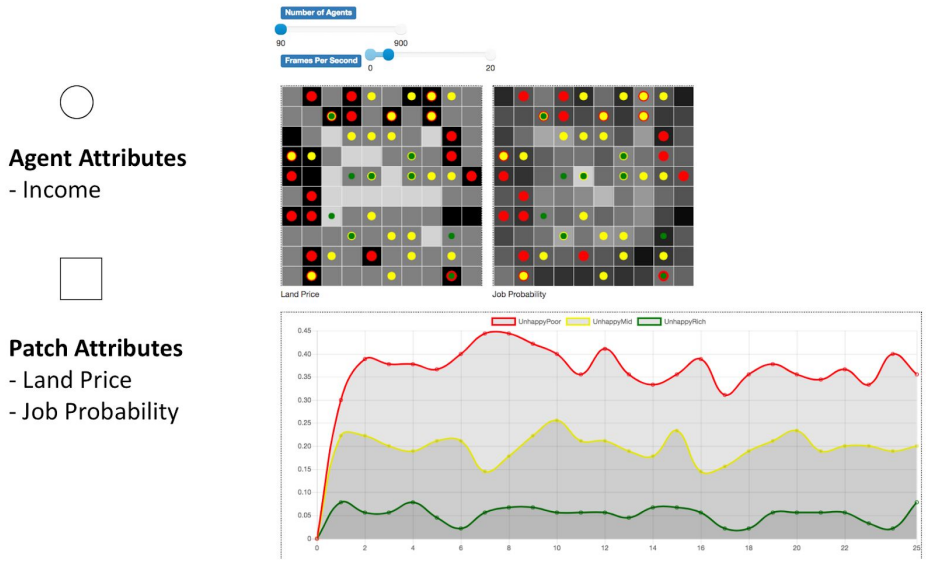


Figure 8: Model with income, land price, and job probability.

3) Model with Income, Density-sensitivity, Job Probabilities, Land Price Interaction

After adding density sensitivity to the model, rich people now prefer to move away if they overlap with others. However, they still seem to occupy the center of the city because they are the only people that can afford the land price (see Figure 9). This alerts me to differentiate land price and housing price in the fourth model because the latter can be compensated with density, which is the strategy poor people use to live close to the city center.

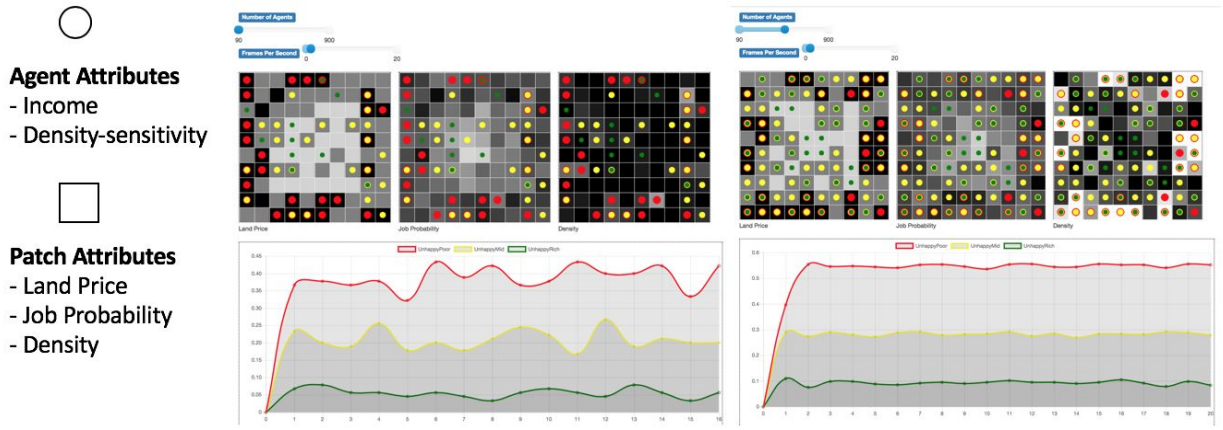


Figure 9: Model with income, land price, job probability, density-sensitivity. The model on the left has 90 agents, while the model on the right has 504 agents.

4) Model with Income, Density-sensitivity, Job Probabilities, Land Price, and Housing Price Interaction

Housing price is calculated as land price divided by density. In Figure 10, we can see that green dots (rich people) got pushed away from city center and red dots (poor people) can finally enter the city center by lowering housing price through density. The spatial segregation of different income groups is barely detectable in this model.

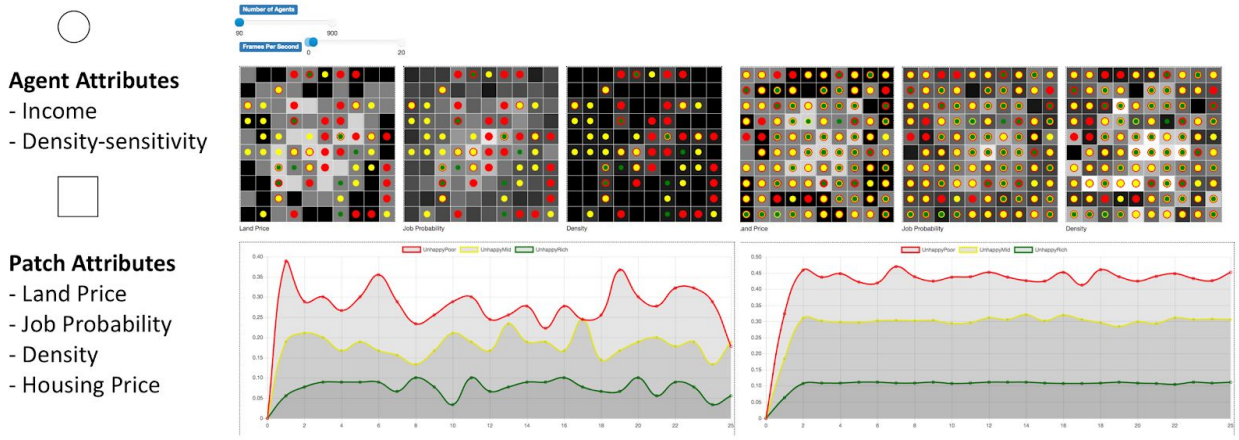


Figure 10: Model with income, land price, job probability, density-sensitivity, and housing price. The model on the left has 90 agents, while the model on the right has 504 agents.

5) Model Sensitivity[10][11]

To ensure what we observed in the fourth model is not due to the idiosyncrasy of the initial state, I tried to average over model outcomes (metric on the percentage of unhappy agents) across different initial states and visualize the results in the box plots (see Figure 11, 12, 13, 14). We can see that, over time, the confidence intervals of any income group's boxes do not seem to decrease, which means that within 50 steps, these models with 100 separate initializations, do not seem to reach a static equilibrium or aim for a steady state. The mean percentage of unhappy agents within each income group also does not vary in a statistically significant way over time. However, at each time step, the average percentage of the rich population does seem to be

[10] #responsibility: followed through last meeting's suggestions, and later proactively presented my work at Foster + Partner to receive external feedback.
[11] #accountability: followed through last meeting's suggestions, and later proactively presented my work at Foster + Partner to receive external feedback.

generally statistically significant from the middle and the poor, while the latter two cannot be completely separated from each other as their confidence intervals overlap.

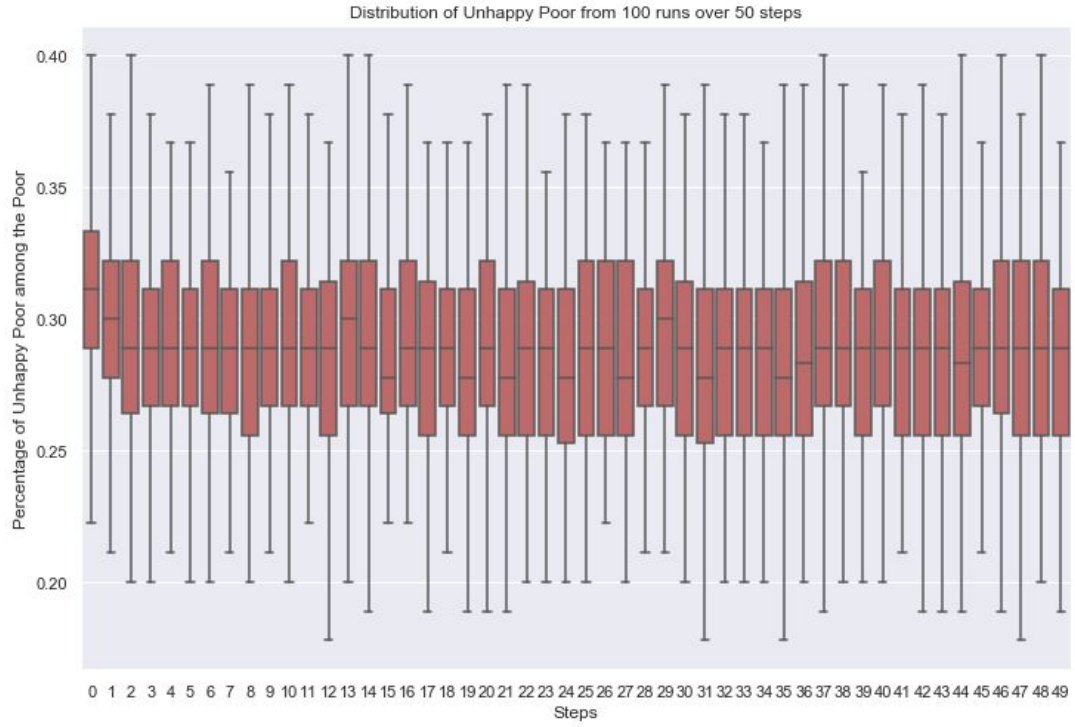


Figure 11: Box plot of Percentages of Unhappy Poor over 50 steps in 100 runs

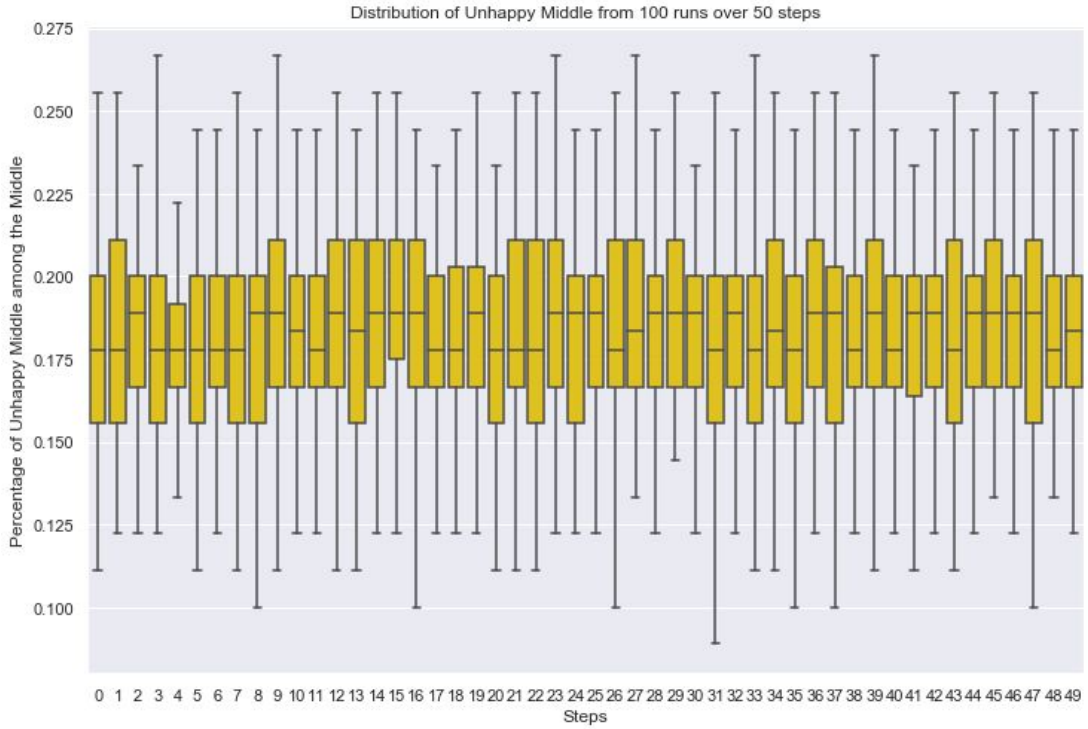


Figure 12: Box plot of Percentages of Unhappy Middle over 50 steps in 100 runs

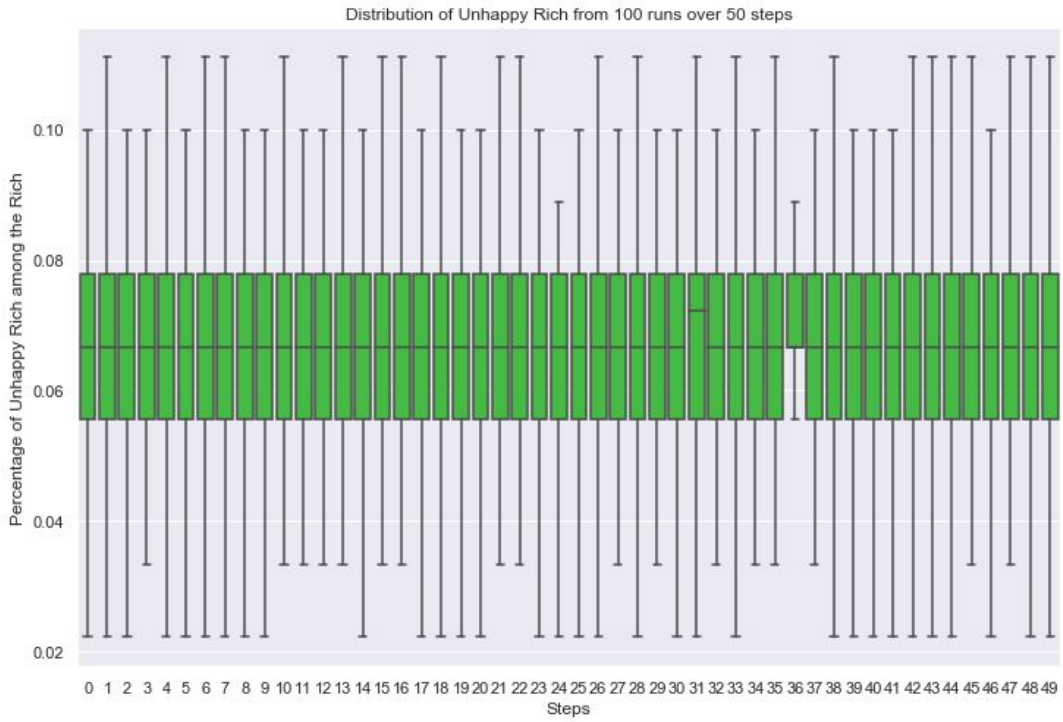


Figure 13: Box plot of Percentages of Unhappy Rich over 50 steps in 100 runs

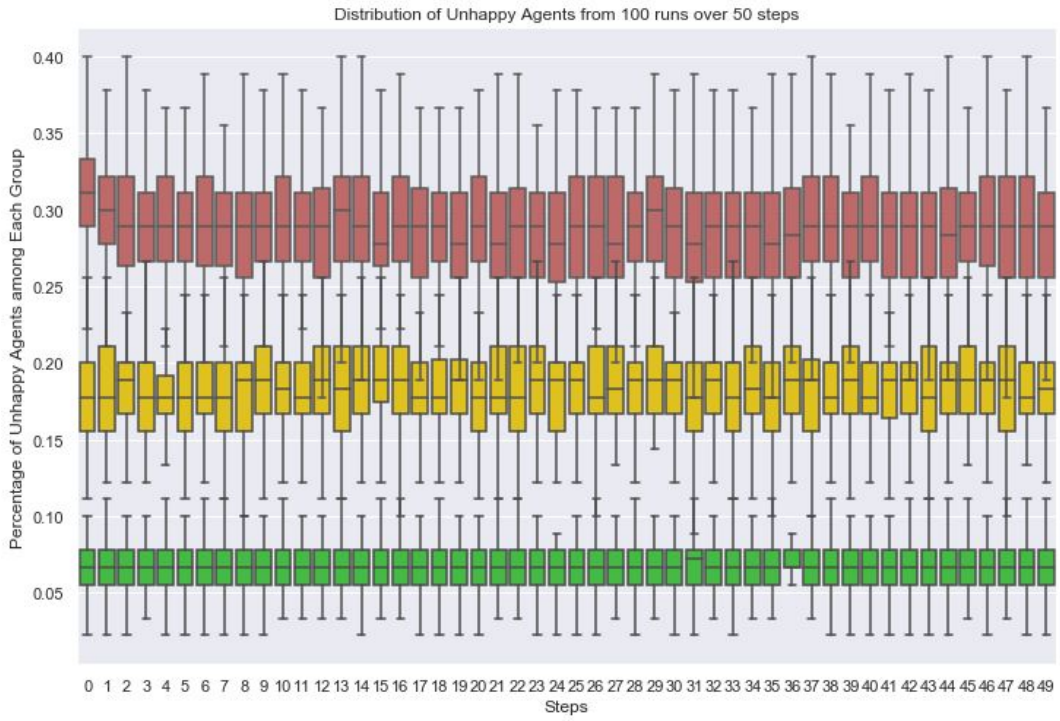


Figure 14: Figure 11, 12, 13 combined in one plot.

6) Spatial Metric

Figure 15, 16, 17 showed the spatial metric that I used to track the changes of different income groups in the space quantitatively. It is plotted in a similar format as the bit-rent curve. Each line in the plot represents the spatial distribution of the percentage of an income group in one step. A decreasing line means that the percentage of this population drops when it moves away from the city center. The lighter the line color, the earlier the time step. A standard model canvas is 10 times 10. Therefore, patches at position [4, 4], [4, 5], [5, 5], and [5, 4] are considered to have 0 unit of distance away from CBD.

For some reasons, the outermost ring (patches have 4 units of distance away from CBD) of the model always has a constant and equal amount of people in each income group (see Figure 18). Due to time constraint, I was not able to debug it before the submission.

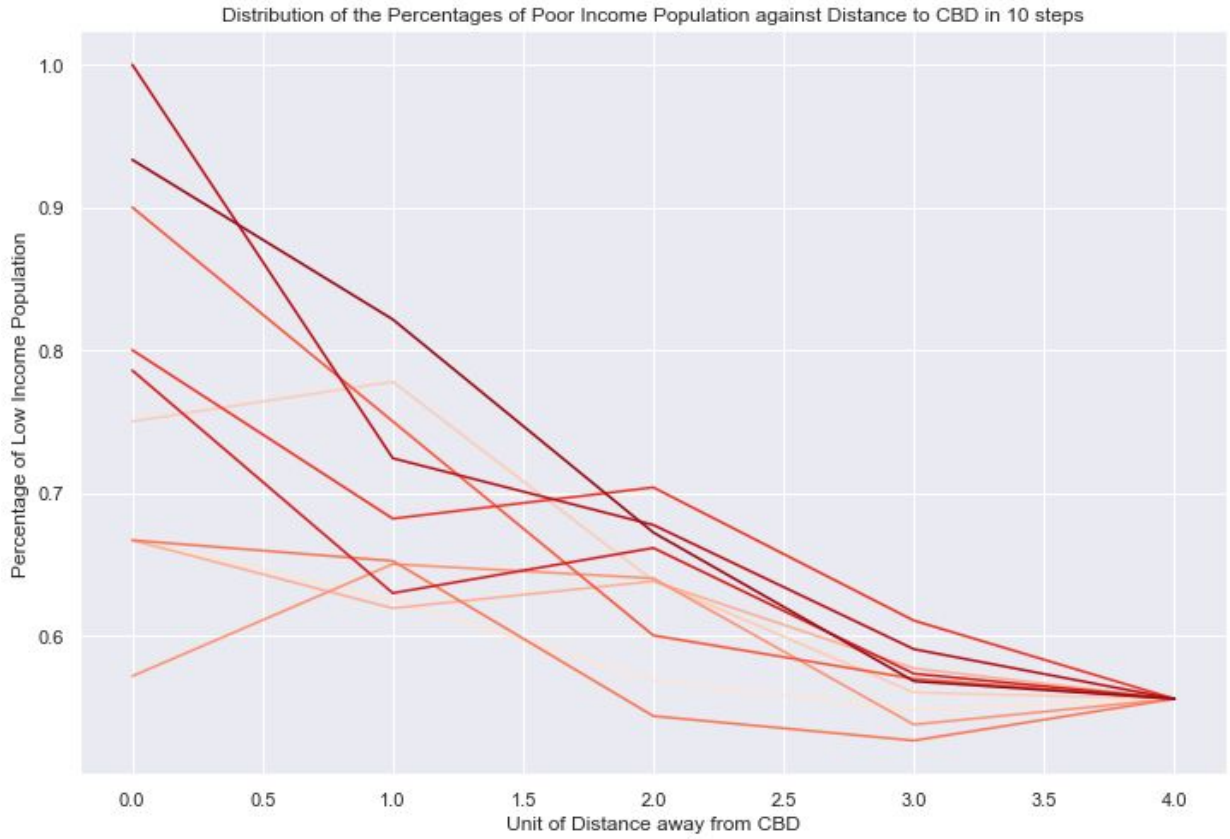


Figure 15: Distribution of low-income population percentage against distance away from CBD in 10 steps.

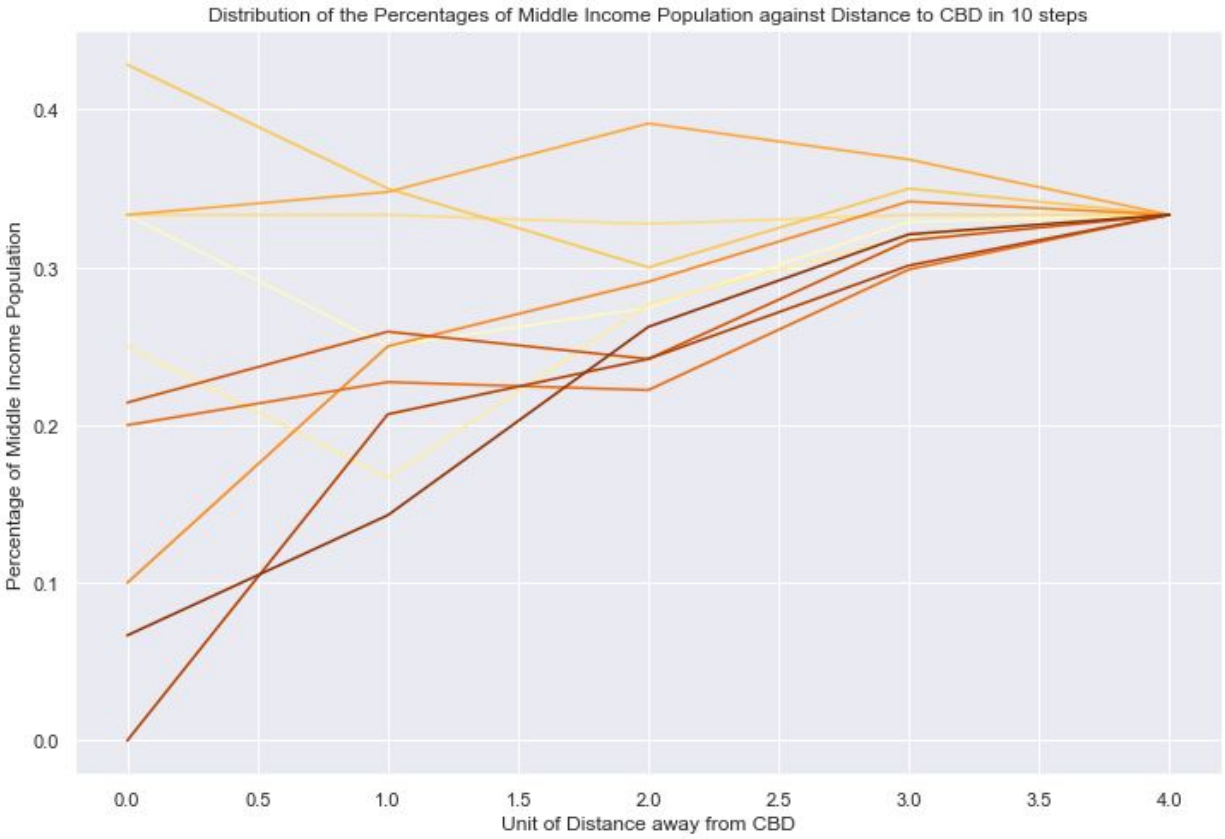

Distribution of the Percentages of Middle Income Population against Distance to CBD in 10 steps
Percentage of Middle Income Population
0.0
0.1
0.2
0.3
0.4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Unit of Distance away from CBD

Figure 16: Distribution of middle-income population percentage against distance away from CBD in 10 steps.

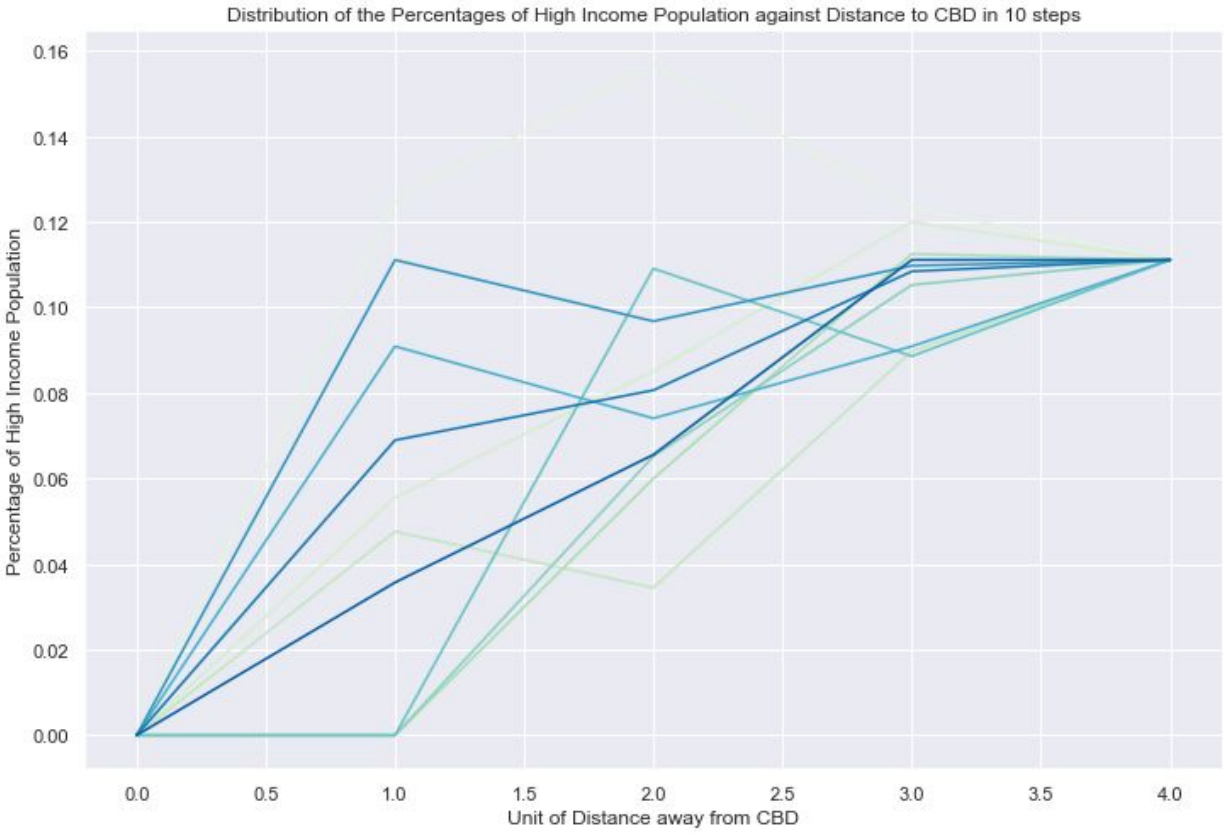


Figure 17: Distribution of middle-income population percentage against distance away from CBD in 10 steps.

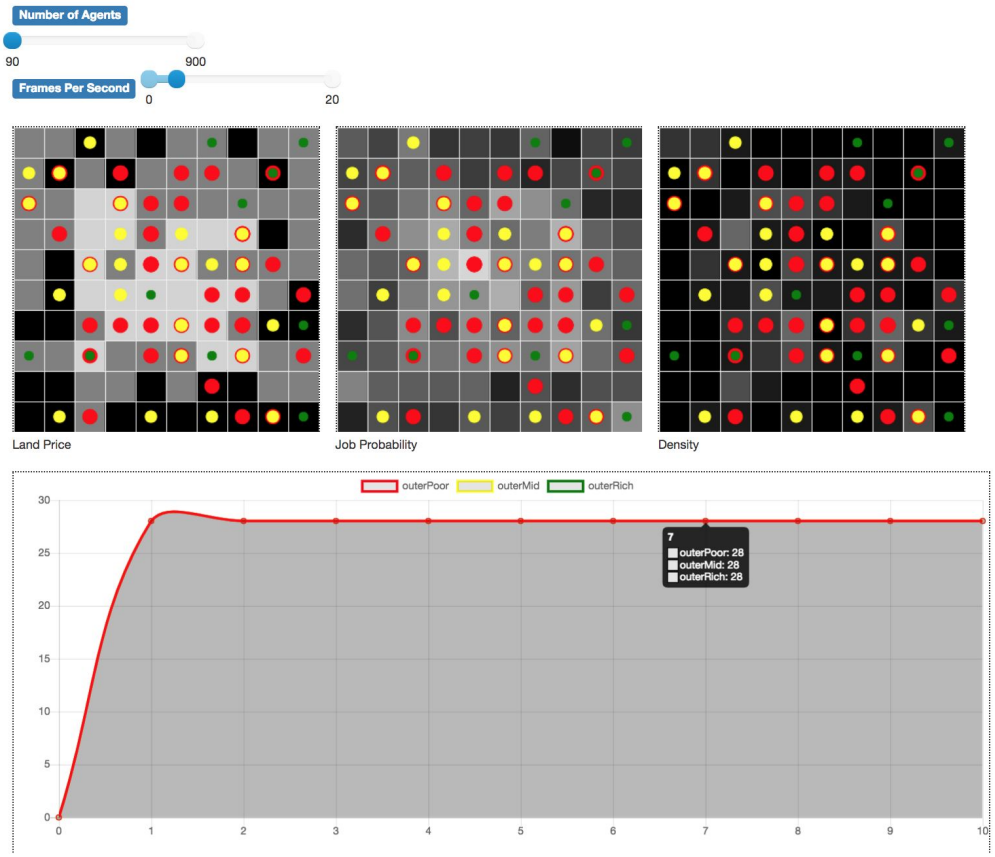


Figure 18: Proof of debugging effort. The metric only captures the percentages of poor, middle, and rich in the outermost ring and still show constant numbers over time.

Model Interpretation and Experts Feedback

I have not put much time into interpreting the models, especially outcomes from the metrics I just devised for this write-up. However, to better understand the model in real-world context, I participated in the presentation with Minerva's civic partner Foster + Partners and presented my work in front of a group of professional urban planners. After the presentation, I have received the following feedback:

- 1) The research question I am trying to answer still seems too broad, and the significance, applications, and audience of this research are also unclear. I believed once I settle down the final project direction (see next section), it will become clearer and I can focus more on interpreting the outcomes for next semester.
- 2) There are a few urban economic theories that address my question (spatial distribution of different populations), which I added them into my literature review.
- 3) Some question that my model is too deterministic, which I am also concerned about.
- 4) One person mentions that the term urban poor is stigmatizing that he would rather see a model that conceptualize people based on their contributions to cities, rather than their objective incomes. This is an interesting idea, but right now I do not plan to incorporate it into my project.
- 5) One person suggested that if I am looking for a specific city to validate my model, Chicago is still the biggest monocentric city in the U.S.A. and thus may fit my constraint.

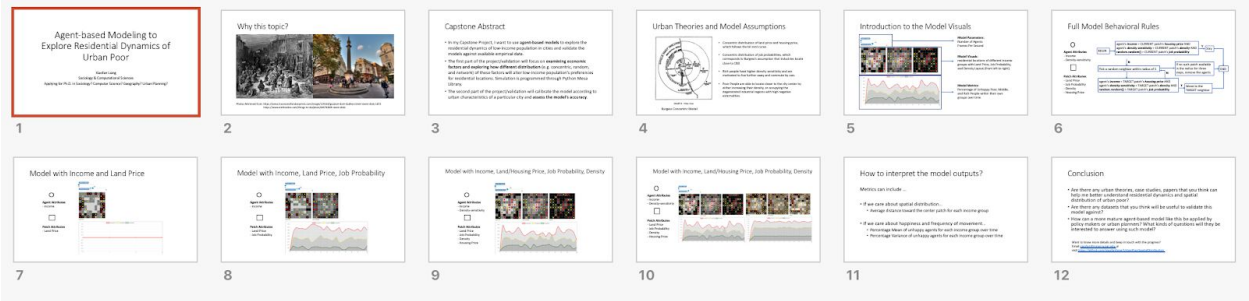


Figure 19: Slides that I used to present at Foster + Partners to receive feedback, in a Pacha Kucha formate (20 seconds * 12 slides)

Next Steps and Timeline[12]

Given the current progress of my capstone, I suggest two directions to go in-depth in the Spring semester before the submission. One direction is to go with what the current abstract describes and try to focus on contextualizing and analyzing the residential dynamics of different kinds of urban poor. The other direction is to stay on the theoretical ground and see if I can replicate the flow of people with different wealth that corresponds to different life cycles of cities, including urbanization/industrialization, sprawl/suburbanization/urban decay, and the reversed flow as the rich returns to the cities while the poor go to the suburbs. To achieve the second one, I will add the macro factors (e.g., introduction of cheap public transportation or cars) that trigger the changes of demography flow into the model for each lifecycle paradigm. The advantage of the second direction is that many of these factors are well-proven in theory and should be able to apply as general patterns to multiple cities.

For the first direction, the final goals before the submission will be to

- 1) fine-tune the metrics analysis of the current and future model
- 2) contextualize the model's parameters and the low-income population's search behaviors in a Chinese city, and
- 3) interpret the model outcomes and the significance to real-world audiences.

[12] #planningarchitecture: demonstrate planning for the next semester. In addition, throughout the Fall semester, Rinad, Skye, Parthiv, Jonathan and I have formed accountability groups that work on capstone together on Friday. Thanks to Parthiv's suggestion, I switched to Python early on this semester that allows me to move my progress forward much faster and had an easier time to debug.

For the second direction, the final goals before the submission will be to

- 1) fine-tune the metrics analysis of the current and future model
- 2) adding three factors into the model, commuting cost, commuting time, and differentiated preference for different income groups. The introduction of cheap public transportation and increasing ownership of cars were the keys to urban expansions, while CityLab cited a decreasing tolerance to commuting time recently as the major factor for the well-off young professionals to return to living in the city centers.
- 3) interpret the model outcomes and the significance to real-world audiences.

Tentative Timeline Table

CP194 Week	Milestones
1 (check-in)	Confirm the final direction, clarify feedback, strategize and finalize the to-do list
3 (asynchronous)	Presented model progress and at least made a few improvements on the second goal above.
5 (check-in)	Presented the final model logics, behavioral rules, and the running outcomes. Finished the second goal above.
7 (asynchronous)	Created final metrics and interpretations of model outcomes
9 (check-in)	Drafted the final paper and final outcomes
10 (capstone project and write-up due)	DUE

Reference[13]

Alonso, W. (1964). Location and land use.

Ball, S., & Petsimeris, P. (2010, May). Mapping urban social divisions. *In Forum qualitative sozialforschung/forum: Qualitative social research* (Vol. 11, No. 2).

Benenson, I. (1998). Multi-agent simulations of residential dynamics in the city. *Computers, Environment and Urban Systems*, 22(1), 25-42.

Bruegmann, R. (2006). *Sprawl: A compact history*. University of Chicago press.

Jaffe, E. (2015). An Intriguing Explanation for the 'Roots of Gentrification': A Hatred of Long Commutes. Retrieved from https://www.citylab.com/transportation/2015/11/why-the-wealthy-have-been-returning-to-the-city-center/416397/

Harris, C. D., & Ullman, E. L. (1945). The nature of cities. *The Annals of the American Academy of Political and Social Science*, 242(1), 7-17.

Hoyt, H. (1939). *The structure and growth of residential neighborhoods in American cities.*

Liu, X., Xia, L., Chen, Y., Liu, T., & Li, S. (2010). 基于多智能体的居住区位空间选择模型. *地理学报*, 65(6). Retrieved from https://wenku.baidu.com/view/f0e9de2aed630b1c59eeb50f.html

Map | Charles Booth's London. (2018). Retrieved from https://booth.lse.ac.uk/map/13/-0.1093/51.5234/100/0

Masad, D., & Kazil, J. (2015). MESA: an agent-based modeling framework. *In Proceedings of*

[13] #sourcequality: high-quality sources.

the 14th Python in Science Conference (SCIPY 2015) (pp. 53-60).

Mills, E. S. (1972). Welfare aspects of national policy toward city sizes. *Urban Studies*, 9(1), 117-124

Muth, R. (1969). *Cities and housing: The spatial patterns of urban residential land use*. University of Chicago, Chicago, 4, 114-123.

Schelling, T. C. (1971). Dynamic models of segregation. *Journal of mathematical sociology*, 1(2), 143-186.

Smith, N. (1979). Toward a theory of gentrification: a back to the city movement by capital, not by people. *J. Am. Plan*. Assoc, 45, 538-548.

Vaughan, L. (2018). *Mapping Society: The Spatial Dimensions of Social Cartography*. UCL Press.

Wu, F. (2002). Complexity and Urban Simulation: Towards a Computational Laboratory. *Geography Research Forum*, 22. Retrieved from http://raphael.geography.ad.bgu.ac.il/ojs/index.php/GRF/article/view/242/234

Zhong, Y., & Feng, J. (2017). 城市迁移人口居住空间分异 ——对深圳市的实证研究. *地理科学进展*, 36(1), 125-135. doi: DOI: 10.18306/dlkxjz.2017.01.012

Appendix - Model Implementation[14]

[14] #AMBimplementation: Substantial amount of coding for an agent-based model and had spent time improving the speed and the structure of the model to accommodate for a large number of agents. Optimize the search behaviors of the agent when finding next neighbors.

Scheduler.py

In [38]:
```
# scheduler.py
from mesa.time import RandomActivation
from collections import defaultdict


# Copy from WolfSheep Example.
# In Mesa, multiple agents including the patch, need to written as bree
d.
# Otherwise, a method is needed for each agent to identify their type.

class RandomActivationByBreed(RandomActivation):
    """
    """
    def __init__(self, model):
        super().__init__(model)
        self.agents_by_breed = defaultdict(dict)

    def add(self, agent):
        self._agents[agent.unique_id] = agent
        agent_class = type(agent)
        self.agents_by_breed[agent_class][agent.unique_id] = agent

    def remove(self, agent):
        del self._agents[agent.unique_id]
        agent_class = type(agent)
        del self.agents_by_breed[agent_class][agent.unique_id]

    def step(self, by_breed=True):
        if by_breed:
            for agent_class in self.agents_by_breed:
                self.step_breed(agent_class)
            self.steps += 1
            self.time += 1
        else:
            super().step()

    def step_breed(self, breed):
        agent_keys = list(self.agents_by_breed[breed].keys())
        random.shuffle(agent_keys)
        for agent_key in agent_keys:
            self.agents_by_breed[breed][agent_key].step()

    def get_breed_count(self, breed_class):
        return len(self.agents_by_breed[breed_class].values())
```

Model.py

In [39]:

```
# model.py
from mesa import Agent, Model
from mesa.space import MultiGrid
from mesa.datacollection import DataCollector
import random
import math
from math import hypot

#calculate percentage of unhappy agents or absolute number of agents in
 each income groups
def datagenerator(income, percentage):

    def compute(model):
        Number = len([agent.happy for agent in model.schedule.agents
                    if type(agent) is PeopleAgent and agent.income == in
come and agent.happy == False])
        if percentage:
            return Number/model.schedule.get_breed_count(PeopleAgent)
        else:
            Number = len([agent.happy for agent in model.schedule.agents

                    if type(agent) is PeopleAgent and agent.income == in
come])
            return Number
    return compute

#calculate a list of percentage of each income group
def datagenerator2(income):

    def compute2(model):

        halfwidth = int(model.width/2)
        center = halfwidth-0.5

        thresholds = [math.hypot(halfwidth + j - center, halfwidth + j -
 center) for j in range(halfwidth)]

        pctlst = []

        for i in thresholds:
            totalsubgrp = []
            totaldensity = [0.0001]
            for agent in model.schedule.agents:
                if type(agent) is PatchAgent and math.hypot(agent.pos[0]
 - center, agent.pos[1] - center) <= i:
                    if income == 1:
                        totalsubgrp.append(agent.Npoor)
                    elif income == 2:
                        totalsubgrp.append(agent.Nmid)
                    else:
                        totalsubgrp.append(agent.Nrich)
                    totaldensity.append(agent.density)
            pctlst.append(sum(totalsubgrp)/sum(totaldensity))

        return pctlst
```

```
    return compute2

def testOuterRing(income):

    def compute3(model):
        counter = 0
        for agent, x, y in model.grid.coord_iter():
            if type(agent) is PeopleAgent and agent.income is income and
 x is 0 or x is 9 or y is 0 or y is 9:
                counter += 1
        return counter

    return compute3


class BurgessModel(Model):


    def __init__(self, N, width, height):
        #super().__init__(seed)
        self.width = width
        self.height = height
        self.num_agents = N
        self.grid = MultiGrid(width, height, True)
        self.schedule = RandomActivationByBreed(self)
        self.running = True

        # Add the agent to a random grid cell
        def createagent(peopletype):
            x = random.randrange(self.grid.width)
            y = random.randrange(self.grid.height)
            a = PeopleAgent(id, peopletype, (x, y), self)
            self.schedule.add(a)
            self.grid.place_agent(a, (x, y))

        # Creat Patches
        id = N
        for agent, x, y in self.grid.coord_iter():
            id += 1
            patch = PatchAgent(id, (x, y), self)
            self.grid.place_agent(patch, (x, y))
            self.schedule.add(patch)

        # Create People Agents
        id = 0
        for i in range(N - (N//9 + N//9*3)):
            id += 1
            createagent("poor")
        for i in range(N//9*3):
            id += 1
            createagent("middle")
        for i in range(N//9):
            id += 1
            createagent("rich")

        #create datacollector
        self.datacollector = DataCollector(
```

```
            model_reporters={"UnhappyPoor": datagenerator(1,True),
                             "UnhappyMid": datagenerator(2,True),
                             "UnhappyRich": datagenerator(3,True)
                            })
        self.computeN = DataCollector(
            model_reporters={"NPoor": datagenerator(1,False),
                             "NMid": datagenerator(2,False),
                             "NRich": datagenerator(3,False)
                            })
        self.computePctDistance = DataCollector(
            model_reporters={"PctDistancePoor": datagenerator2(1),
                             "PctDistanceMid": datagenerator2(2),
                             "PctDistanceRich": datagenerator2(3)
                            })
        self.computeTest = DataCollector(
            model_reporters={"outerPoor": testOuterRing(1),
                             "outerMid": testOuterRing(2),
                             "outerRich": testOuterRing(3)
                            })
    def step(self):
        self.schedule.step()
        self.datacollector.collect(self)
        self.computeN.collect(self)
        self.computePctDistance.collect(self)
        self.computeTest.collect(self)
```

Agents.py

In [40]:

```
#agents.py

def matchpatch(pos, model):
    for agent in model.grid.grid[pos[0]][pos[1]]:
        if type(agent) is PatchAgent:
            return agent

class PeopleAgent(Agent):

    def __init__(self, unique_id, category, pos, model):
        super().__init__(unique_id, model)
        #added a parameter of category
        self.category = category
        self.model = model
        self.happy = False
        self.overdue = 0
        self.pos = pos

        if category == "poor":
            self.income = 1
            self.dens_sens = 5
        elif category == "middle":
            self.income = 2
            self.dens_sens = 3
        elif category == "rich":
            self.income = 3
            self.dens_sens = 1
        else:
            raise Exception("category not found")

    def move(self):
        # for each step of one agent, find possible neighbors within cer
tain radius
        possible_neighbors = self.model.grid.get_neighborhood(
            self.pos,
            True,
            radius = 5,
            include_center=False)

        #pick one of the possible steps that has income >= Lprice, dens_
sens >= density, (later job_probs)
        #the patch that has the same location as step

        new_position = False

        while possible_neighbors:
            neighbor = random.choice(possible_neighbors)
            possible_neighbors.remove(neighbor)
            patch = matchpatch(neighbor, self.model)
            if self.income >= patch.Hprice and random.random() <= patch.
jobs_prob and self.dens_sens >= patch.density:
                new_position = neighbor
                break
        if new_position:
            self.model.grid.move_agent(self, new_position)
            self.pos = new_position
```

```
            self.overdue = 0
        else:
            self.overdue += 1
            self.model.schedule.remove(self)
            if self.overdue >= 3:
                self.model.schedule.remove(self)

    #update agent happiness
    def updatehappiness(self):
        patch = matchpatch(self.pos, self.model)
        self.happy = self.income >= patch.Hprice and random.random() <=
patch.jobs_prob and self.dens_sens >= patch.density

    def step(self):
        # check if income still higher than Lprice and job maintains
        self.updatehappiness()
        if not self.happy:
            self.move()
        self.updatehappiness()
        pass


class PatchAgent(Agent):

    def __init__(self, unique_id, pos, model):
        super().__init__(unique_id, model)
        self.model = model
        self.pos = pos
        self.Npoor = 0
        self.Nmid = 0
        self.Nrich = 0

        #assume the range of CBD, urban, suburban is 2:4:4 (arbitrary)
        width = self.model.width
        lt_exp = width/2 - 1 #4
        rt_exp = width/2 #5
        rt_mid = rt_exp + width/5 #7
        lt_mid = lt_exp - width/5 #2
        x = self.pos[0]
        y = self.pos[1]

        #-------- Assign job probability and land price distribution ---
-------.
        #high
        if rt_exp >= x >= lt_exp and rt_exp >= y >= lt_exp:

            #high job distribution
            self.jobs_prob = random.normalvariate(0.7, 0.1)

            #80% chance of having the most expensive price (arbitrary)
            if random.random() >= 0.2:
                self.Lprice = 3
            else:
                self.Lprice = 2
        #low
        elif ((lt_mid > x >= 0 or width -1 >= x > rt_mid) and (width - 1
 >= y >= 0)) or ((lt_mid > y >= 0 or width - 1 >= y > rt_mid) and (width
```

```
 - 1 >= x >= 0)):
            self.jobs_prob = self.jobs_prob = random.normalvariate(0.3,
0.1)
            self.Lprice = random.randint(1,2)

        #middle
        else:
            self.Lprice = random.randint(2,3)
            self.jobs_prob = random.normalvariate(0.5, 0.1)

        # ---- Caculate the amount of agents on each patch ---------.
        self.density = len(self.model.grid.grid[self.pos[0]][self.pos[1
]])-1
        #self.density = 0 #in case we do not want Housing price as a fac
tor. add this line.

        # ---- Caculate the Housing price given land price and density -
----. (This is quite arbitrary right now)
        if self.density == 0:
            self.Hprice = self.Lprice
        else:
            self.Hprice = self.Lprice/self.density

    def step(self):
        #count number of people on the same patch and update density
        #self.density = 0 #in case we do not want density as a factor. a
dd this line.
        agentsonpatch = self.model.grid.grid[self.pos[0]][self.pos[1]]
        self.density = len(agentsonpatch)-1
        self.Npoor = len([agent for agent in agentsonpatch if type(agent
) is PeopleAgent and agent.income == 1])
        self.Nmid = len([agent for agent in agentsonpatch if type(agent)
 is PeopleAgent and agent.income == 2])
        self.Nrich = len([agent for agent in agentsonpatch if type(agent
) is PeopleAgent and agent.income == 3])
```

Test run

In [41]:
```
test = BurgessModel(90,10,10)
```

In [42]:
```
for i in range(20):
    test.step()
```

Server.py

In [6]:

```
#server.py

from mesa.visualization.modules import CanvasGrid
from mesa.visualization.ModularVisualization import ModularServer
from mesa.visualization.modules import ChartModule
from mesa.visualization.UserParam import UserSettableParameter

def layout(layout_choice):
    def agent_portrayal(agent):
        if type(agent) is PatchAgent:
            portrayal = {"Shape": "rect",
                         "Filled": "true",
                         "Layer": 0,
                         "w": 1,
                         "h": 1,
                         "Color": "white",
                        }
            if layout_choice == "land_price":
                if agent.Lprice == 3:
                    portrayal["Color"] = "LightGrey"
                elif agent.Lprice == 2:
                    portrayal["Color"] = "Grey"
                else:
                    portrayal["Color"] = "black"

            elif layout_choice == "job_prob":
                portrayal["Color"] = "hsl(0, 0%, "+str(agent.jobs_prob *
 100)+"%)"

            #a better density calculation can be designed.
            elif layout_choice == "density":
                portrayal["Color"] = "hsl(0, 0%, "+str(agent.density * 1
0)+"%)"

            else: raise Exception("Agent Layer not found")


        elif type(agent) is PeopleAgent:
            portrayal = {"Shape": "circle",
                         "Filled": "true",
                         "Layer": 1,
                         "r": 0.5}
            if agent.category == 'poor':
                portrayal["Color"] = "red"
            elif agent.category == 'middle':
                portrayal["Color"] = "yellow"
                portrayal["Layer"] = 2
                portrayal["r"] = 0.4
            else:
                portrayal["Color"] = "green"
                portrayal["Layer"] = 3
                portrayal["r"] = 0.3

        else: raise Exception("Agent Portrayal not found")

        return portrayal
```

```
    return agent_portrayal

#the width and height can only be multiple of 10 because the proportion
 of low, middle, high job ring and land price ring is distributed accord
ing 2:4:4
width = 10
height = 10

land_price = CanvasGrid(layout("land_price"), width, height, 350, 350)
job_prob = CanvasGrid(layout("job_prob"), width, height, 350, 350)
density = CanvasGrid(layout("density"), width, height, 350, 350)

pctunhappychart = ChartModule([{"Label": "UnhappyPoor",
                      "Color": "Red"},
                     {"Label": "UnhappyMid",
                      "Color": "Yellow"},
                     {"Label": "UnhappyRich",
                      "Color": "Green"}
                    ],
                    data_collector_name='datacollector')

# nunhappychart = ChartModule([{"Label": "NPoor",
#                       "Color": "Red"},
#                      {"Label": "NMid",
#                      "Color": "Yellow"},
#                      {"Label": "NRich",
#                      "Color": "Green"}
#                     ],
#                     data_collector_name='computeN')

testChart = ChartModule([{"Label": "outerPoor",
                      "Color": "Red"},
                     {"Label": "outerMid",
                      "Color": "Yellow"},
                     {"Label": "outerRich",
                      "Color": "Green"}
                    ],
                    data_collector_name='computeTest')

n_slider = UserSettableParameter("slider", "Number of Agents", 90, 90, 9
00, 9)

# layout_choice = UserSettableParameter("choice", "Layout Choices", valu
e = "land_price",
#                                       choices = ["land_price", "job_pr
ob", "density"])

userparameters = {"N": n_slider,
                  "width": width,
                  "height": height,
                 }

from mesa.visualization.ModularVisualization import VisualizationElement

#personalized web layout in Javascript
class Reorganize(VisualizationElement):
    def __init__(self):
```

```
        self.js_code = """
        canvasbox = document.querySelector('#elements')
        canvasbox.style.width = '1150px';
        allcanvas = document.querySelectorAll('canvas:not(.chartjs-rende
r-monitor)');

        var namelist = ["Land Price", "Job Probability", "Density"]
        allcanvas.forEach(
            function (elem, index) {
            var div = document.createElement('div');
            div.style.display = 'inline-block';
            div.style.marginRight = '20px';
            div.style.marginBottom = '20px';
            div.style.marginTop = '20px';
            div.style.width = '350px';
            canvasbox.insertBefore(div, elem);
            div.appendChild(elem);
            div.appendChild(document.createTextNode(namelist[index]))
            }
        )
        """

JScode = Reorganize()

server = ModularServer(BurgessModel,
                       [land_price, job_prob, density, testChart, JScode
],
                       "Burgess Model",
                       userparameters)
```

In [7]:
```
server.port = 8521 # The default
server.launch()
```

```
Interface starting at http://127.0.0.1:8521
```

Data Analysis

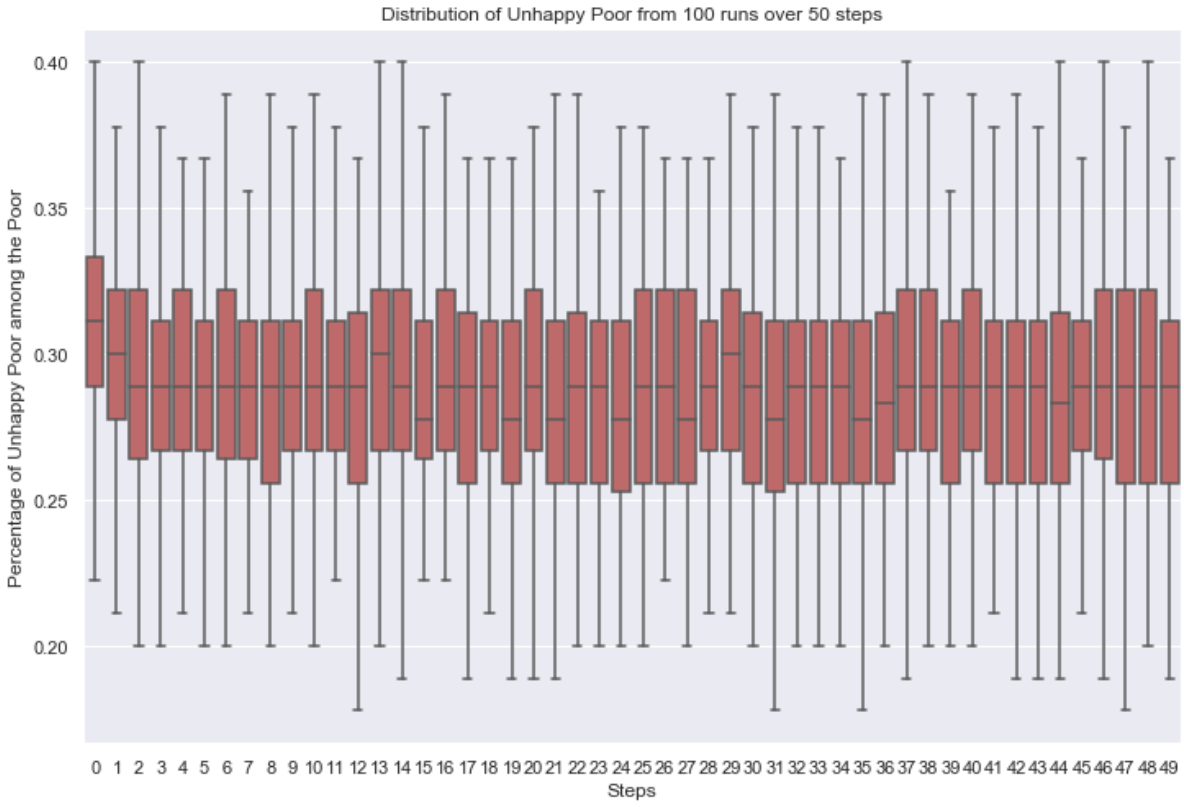
In [43]:
```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import time
start = time.time()
#import seaborn as sns

meanmid = []
meanpoor = []
meanrich = []
varmid = []
varpoor = []
varrich = []
midBoxData = pd.DataFrame()
poorBoxData = pd.DataFrame()
richBoxData = pd.DataFrame()

#100 runs and 50 steps per run
for i in range(100):
    model = BurgessModel(90,10,10)
    for t in range(50):
        model.step()

    model_df = model.datacollector.get_model_vars_dataframe()

    mean_mid = np.mean(model_df["UnhappyMid"])
    var_mid = np.var(model_df["UnhappyMid"])
    midBoxData["run"+str(i+1)] = model_df["UnhappyMid"]

    mean_poor = np.mean(model_df["UnhappyPoor"])
    var_poor = np.var(model_df["UnhappyPoor"])
    poorBoxData["run"+str(i+1)] = model_df["UnhappyPoor"]

    mean_rich = np.mean(model_df["UnhappyRich"])
    var_rich = np.var(model_df["UnhappyRich"])
    richBoxData["run"+str(i+1)] = model_df["UnhappyRich"]

    meanmid.append(mean_mid)
    meanpoor.append(mean_poor)
    meanrich.append(mean_rich)
    varmid.append(var_mid)
    varpoor.append(var_poor)
    varrich.append(var_rich)

end = time.time()
print(end - start)
```

```
81.16215205192566
```

In [44]:
```
#percentage over distance metric.
```

In [45]:
```
sns.set(rc={'figure.figsize':(12,8)})
plt.title('Distribution of Unhappy Poor from 100 runs over 50 steps')
plt.xlabel('Steps')
plt.ylabel('Percentage of Unhappy Poor among the Poor')
sns.boxplot(data=poorBoxData.T, color="indianred", showfliers=False)
```

Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x10a986b00>

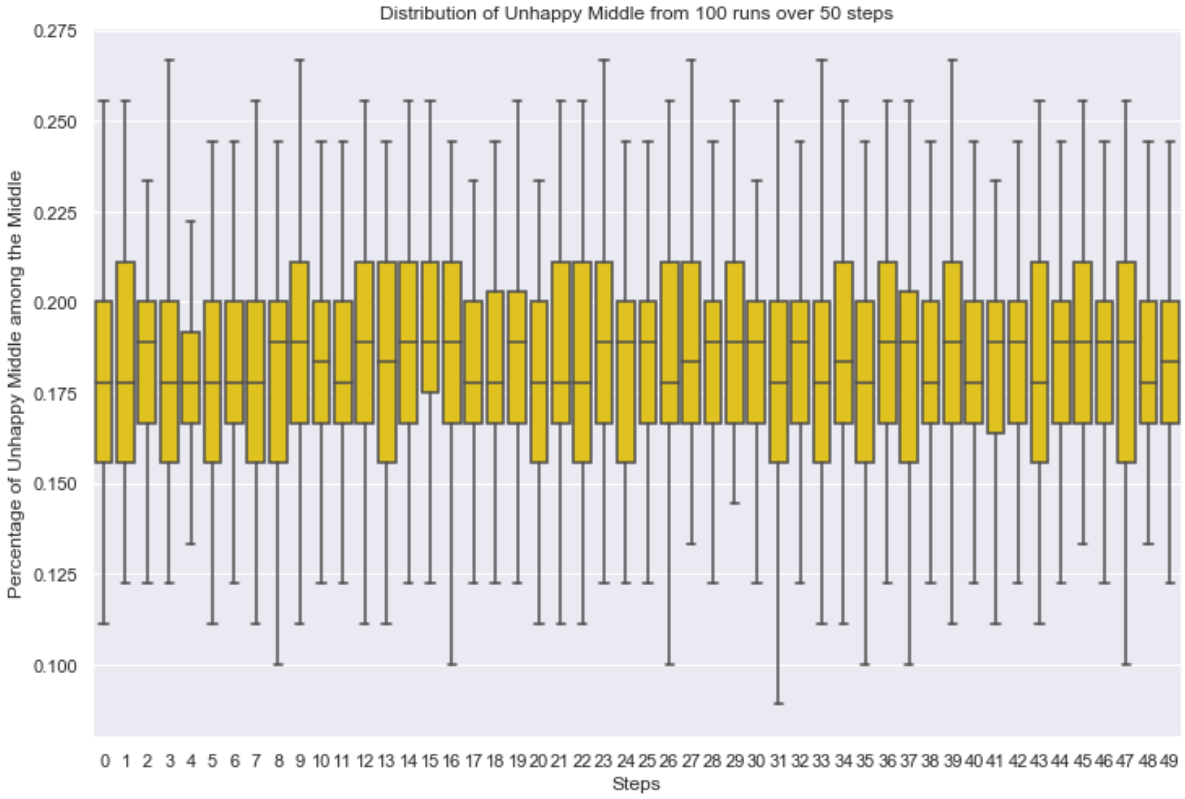


In [46]:
```
#example of one unhappy poor percentage over time.
sns.lineplot(data=model_df["UnhappyPoor"])
```

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x10ac93470>

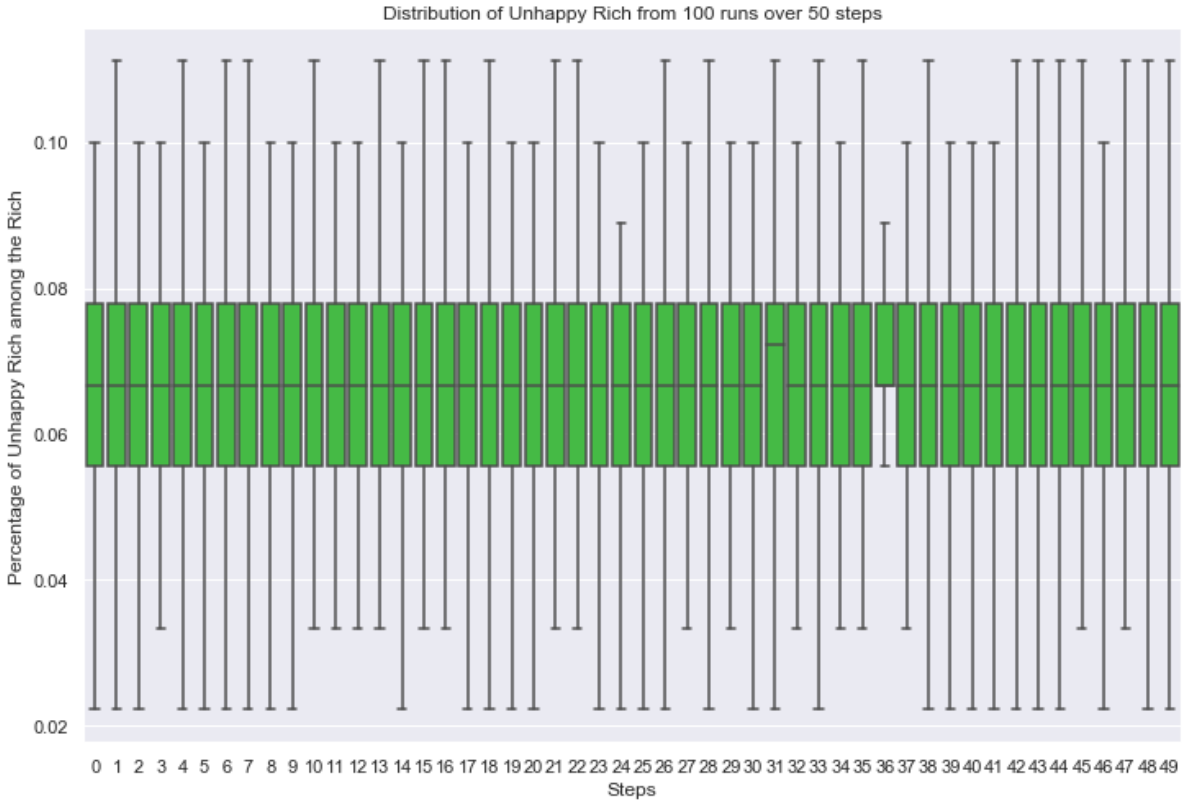
In [47]:
```
plt.title('Distribution of Unhappy Middle from 100 runs over 50 steps')
plt.xlabel('Steps')
plt.ylabel('Percentage of Unhappy Middle among the Middle')
sns.boxplot(data=midBoxData.T, color="gold", showfliers=False)
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x10b07dba8>



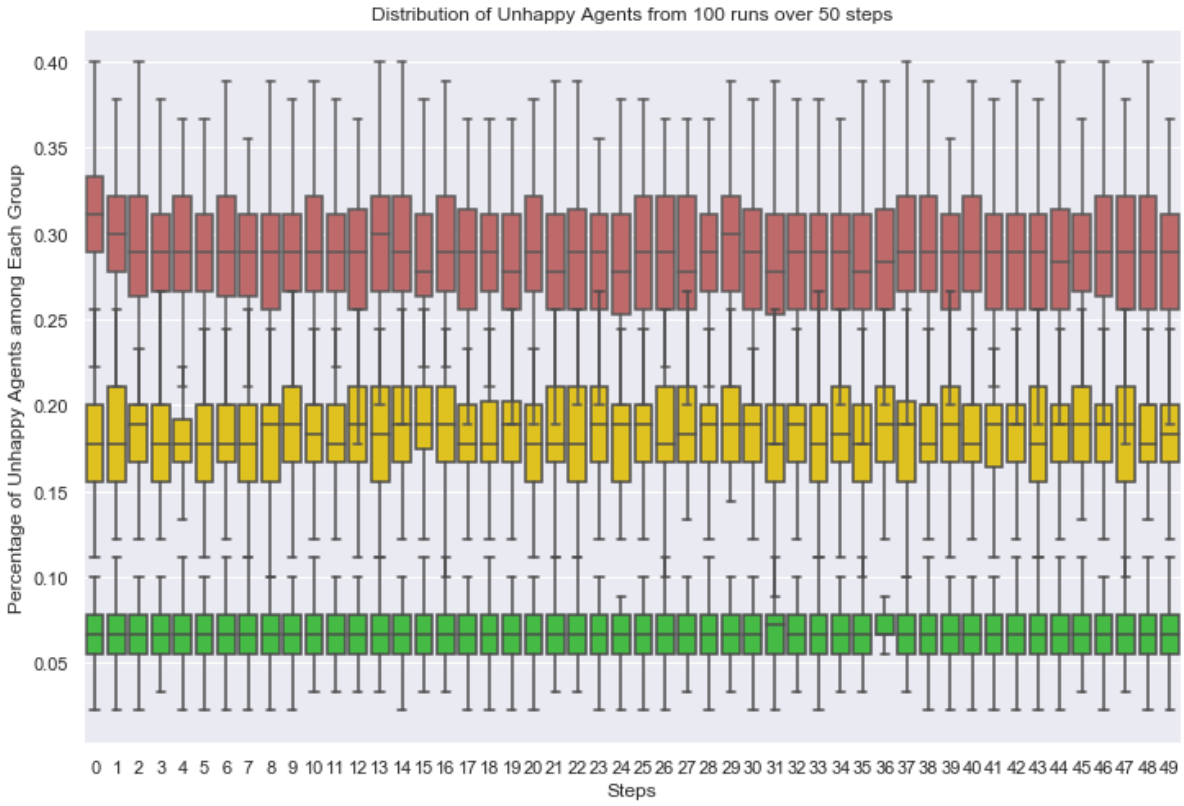
In [55]:
```
plt.title('Distribution of Unhappy Rich from 100 runs over 50 steps')
plt.xlabel('Steps')
plt.ylabel('Percentage of Unhappy Rich among the Rich')
sns.boxplot(data=richBoxData.T, color="limegreen", showfliers=False)
```

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x10da13b00>



In [56]:
```
plt.title('Distribution of Unhappy Agents from 100 runs over 50 steps')
plt.xlabel('Steps')
plt.ylabel('Percentage of Unhappy Agents among Each Group')
sns.boxplot(data=poorBoxData.T, color="indianred", showfliers=False)
sns.boxplot(data=midBoxData.T, color="gold", showfliers=False)
sns.boxplot(data=richBoxData.T, color="limegreen", showfliers=False)
```

Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x10e125978>



In [49]:
```
model2 = BurgessModel(90,10,10)
for t in range(10):
    model2.step()

model2_df = model2.computePctDistance.get_model_vars_dataframe()
model2_df
```

Out[49]:

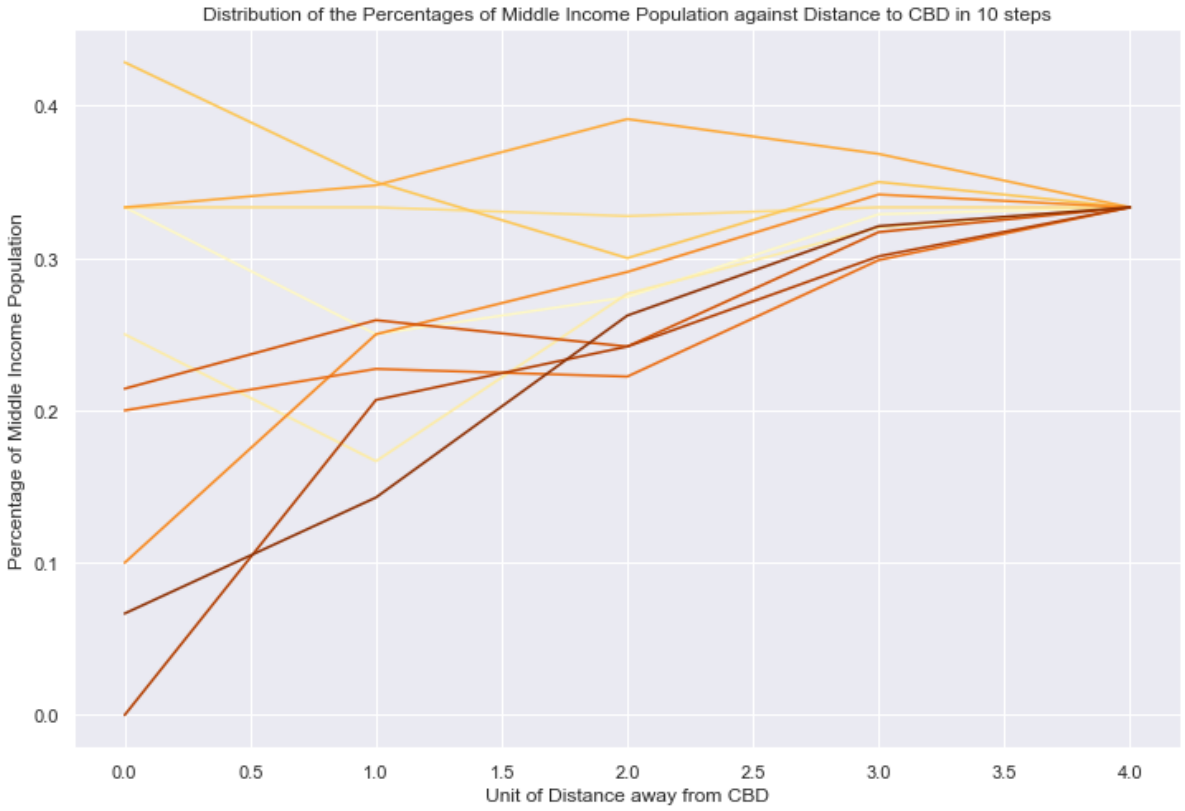
	PctDistanceMid	PctDistancePoor	PctDistanceRich
0	[0.33332962967078145, 0.24999843750976558, 0.2...	[0.6666592593415629, 0.6249960937744139, 0.568...	[0.0, 0.12499921875488279, 0.1568624375246323,...
1	[0.24999375015624611, 0.16666574074588475, 0.2...	[0.7499812504687383, 0.7777734568141288, 0.638...	[0.0, 0.05555524691529492, 0.08510620190169807...
2	[0.3333277778703688, 0.3333317460393046, 0.327...	[0.6666555557407376, 0.6190446712158514, 0.637...	[0.0, 0.047618820862757795, 0.0344826991677600...
3	[0.4285653062099113, 0.34999825000874996, 0.29...	[0.5714204082798817, 0.6499967500162499, 0.639...	[0.0, 0.0, 0.059999880000239994, 0.11249985937...
4	[0.33332962967078145, 0.3478245746757623, 0.39...	[0.6666592593415629, 0.6521710775170543, 0.543...	[0.0, 0.0, 0.06521724952771842, 0.105263019390...
5	[0.09999900000999991, 0.2499989583376736, 0.29...	[0.8999910000899991, 0.7499968750130208, 0.599...	[0.0, 0.0, 0.10909071074416228, 0.088607482775...
6	[0.19999800001999982, 0.22727169421957172, 0.2...	[0.7999920000799993, 0.6818150826587152, 0.703...	[0.0, 0.09090867768782869, 0.07407393690011685...
7	[0.21428418368440225, 0.25925829904333686, 0.2...	[0.7857086735094749, 0.6296272976766752, 0.661...	[0.0, 0.11111069959000151, 0.0967740374612299,...
8	[0.0, 0.2068958382902128, 0.24193509365307475,...	[0.9999923077514788, 0.7241354340157448, 0.677...	[0.0, 0.06896527943007093, 0.08064503121769158...
9	[0.06666622222518517, 0.14285663265488338, 0.2...	[0.9333271111525924, 0.8214256377655794, 0.672...	[0.0, 0.035714158163720845, 0.0655736629939950...

In [50]:
```
for i in range(5):
    print(model2_df["PctDistanceMid"][i])
```

```
[0.33332962967078145, 0.24999843750976558, 0.2745092656681065, 0.328766
67292236583, 0.33333296296337445]
[0.24999375015624611, 0.16666574074588475, 0.2765951561805187, 0.319999
5733339022, 0.33333296296337445]
[0.3333277778703688, 0.3333317460393046, 0.3275856420937205, 0.33333290
598345383, 0.33333296296337445]
[0.4285653062099113, 0.34999825000874996, 0.29999940000119996, 0.349999
56250054687, 0.33333296296337445]
[0.33332962967078145, 0.3478245746757623, 0.39130349716631047, 0.368420
56786767385, 0.33333296296337445]
```

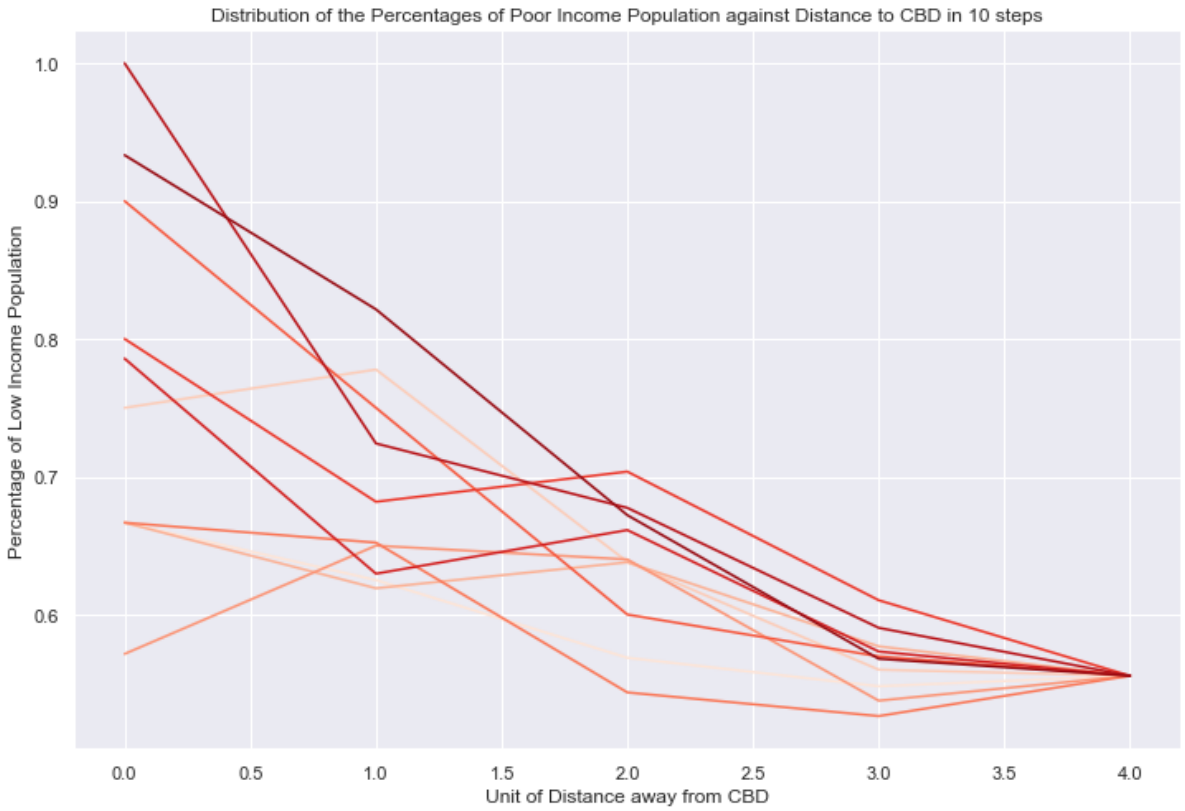
In [51]:
```
plt.gca().set_prop_cycle('color', [plt.cm.YlOrBr(i) for i in np.linspace
(0.1, 0.9, 10)])

for t in range(10):
    sns.lineplot(x=range(5), y=model2_df["PctDistanceMid"][t])

plt.title('Distribution of the Percentages of Middle Income Population a
gainst Distance to CBD in 10 steps')
plt.xlabel('Unit of Distance away from CBD')
plt.ylabel('Percentage of Middle Income Population')
```

Out[51]: Text(0,0.5,'Percentage of Middle Income Population')



In [52]:
```
plt.gca().set_prop_cycle('color', [plt.cm.Reds(i) for i in np.linspace(
0.1, 0.9, 10)])

for t in range(10):
    sns.lineplot(x=range(5), y=model2_df["PctDistancePoor"][t])

plt.title('Distribution of the Percentages of Poor Income Population aga
inst Distance to CBD in 10 steps')
plt.xlabel('Unit of Distance away from CBD')
plt.ylabel('Percentage of Low Income Population')
```

Out[52]:
```
Text(0,0.5,'Percentage of Low Income Population')
```



In [53]:
```
plt.gca().set_prop_cycle('color', [plt.cm.GnBu(i) for i in np.linspace(
0.1, 0.9, 10)])

for t in range(10):
    sns.lineplot(x=range(5), y=model2_df["PctDistanceRich"][t])

plt.title('Distribution of the Percentages of High Income Population aga
inst Distance to CBD in 10 steps')
plt.xlabel('Unit of Distance away from CBD')
plt.ylabel('Percentage of High Income Population')
```

Out[53]:
```
Text(0,0.5,'Percentage of High Income Population')
```