BOĞAZİÇİ UNIVERSITY
DEPARTMENT OF MANAGEMENT INFORMATION SYSTEMS

Economixim: Simulating a National Economy with Agent-Based Modeling

YUSUF COŞKUN

June, 2025

ABSTRACT

Agent-based models (ABMs) have made economic policy-making significantly easier and have brought the discussion to an unbiased plane. However, most existing models are either (a) using representative agents and are macroeconomics focused or (b) focused on a specific aspect of an economy (e.g. tax policy). To bridge this gap, this study introduces Economixim, an agent-based economic simulation built on top of the Mesa framework, designed to allow policy-makers to make micro-level adjustments in an economy and capture microeconomic and macroeconomic emergent behaviors. The goal of this paper and tool is to simulate an economy involving firms, households, individual persons and a government that are equipped with adaptive processes.

The simulation facilitates the emergence of economic phenomena like market equilibrium, social welfare, government fiscal sustainability, and labor market dynamics. By modeling the simulation with heterogeneous agents, the model reveals the impact of economic policies and micro-level decisions that would otherwise be observed weaker in a homogeneous-agent model.

ÖZET

Aktör Tabanlı Modeller (ABM'ler), ekonomi politikalarının belirlenme sürecini önemli ölçüde kolaylaştırmış ve bu sürecin tarafsız bir sahaya çekilmesini sağlamıştır. Ancak mevcut ekonomik ABM'lerin çoğu ya makroekonomiye odaklanmış, ya temsili aktörler kullanmış, ya da ekonominin daha küçük bir kesitine (ör. vergi sistemi) odaklanmıştır. Bu çalışma, bu soruna daha dengeli bir çözüm sunma amacıyla geliştirilen "Economixim" ekonomi simülasyonunu açıklama amacıyla yapılmıştır. Economixim, ekonomi politikaları üzerine tartışan ekonomistlerin ekonomide mikro düzeyde ayarlamalar yapabilmesine ve hem mikroekonomik hem de makroekonomik ölçekte ortaya çıkan davranışları gözlemleyebilmesine olanak sağlayacak şekilde tasarlanmıştır. Bu aracın temel amacı, şartlara uyum sağlayarak karar verme yetisine sahip şirketler, haneler, bireyler ve bir devlet yapısından oluşan bir ekonomiyi simüle etmektir.

Simülasyon, arz-talep dengesi, refah, ülke ekonomisi ve işgücü piyasası gibi ekonomik olguların ortaya çıkışını mümkün kılmaktadır. Simülasyondaki aktörlerin heterojen şekilde modellenmesi, homojen şekilde modellenen aktörlerin bulunduğu bir simülasyonun aksine, ekonomik politikaların ve mikro düzeyde alınan kararların etkisinin güçlü bir şekilde gözlemlenebilmesini sağlamaktadır.

1. INTRODUCTION

The study of understanding and developing complex economic systems has led to the creation of various modeling methods. While traditional macroeconomic models, such as DSGE, have been used for macroeconomic analysis for a long time, their reliance on representative agents, rational expectations, and market clearing assumptions have been criticised for not capturing real-world complexities such as heterogeneous behavior enough (Dawid & Delli Gatti, 2018; Heckbert et al., 2010). Agent-Based Modeling (ABM) has emerged as a solution to this criticism, with a "bottom-up" approach to simulate economic systems by focusing on the interactions of autonomous, diverse, and granular entities (Macal & North, 2009; Tesfatsion, 2006). This allows for the exploration of emergent macroeconomic behavior arising from micro-level decisions, creating a sort of digital laboratory to test different policies and techniques in (Farmer & Foley, 2009). The flexibility of ABM allows the implementation of behavioral rules, changing models from having only strict utility maximization to including heuristics, social influence, and learning. Foundational works in the ABM field, such as Schelling's (1971) model of segregation and Epstein and Axtell's (1996) "SugarScape," showcase the power of ABMs to demonstrate how simple individual interactions can generate complex, often counterintuitive, patterns. These counterintuitive results of ABMs were something that we also witnessed during the development of Economixim.

The application of ABM in economics has grown thanks to advancements in computational power and the development of modeling toolkits such as Mesa, NetLogo, and Repast. These tools allow researchers to design virtual economies where they can explore the consequences of different behavioral rules and policy interventions.

In the area of macroeconomics, ABMs have been developed to simulate national and global economies, with heterogenous agents that interact with each other. For example, frameworks like BeforeIT.jl (Glielmo et al., 2025) simulate national economies with millions of heterogenous households, firms, and financial institutions, calibrated with detailed real-world data. Delli Gatti et al. (2018) provides a comprehensive toolkit for such projects. Earlier influential works like Assenza, Delli Gati, and Grazzini (2015) explored emergent behavior in macroeconomic ABMs with detailed capital and credit mechanisms. These models aim to reproduce empirical facts, and to provide economic forecasts.

A recent and important advancement in economic ABMs is the addition of learning and adaptation. Rather than relying only on predefined rules, more and more ABMs are implementing agents with the ability to learn from their environment and adjust their strategies. Reinforcement Learning (RL) has become a leading technique for this purpose. For example, Osoba et al. (2020) explored the use of RL agents in a policy-focused ABM, and demonstrated that RL agents can learn reward-maximising behaviors in situations such as minority games and disease transmission models. Brusatin et al. (2024) substituted a variable number of Firms with RL agents in a macroeconomic ABM to study the impact of varying degrees of firm rationality on economic outcomes, finding that RL firms could learn complex profit-maximizing strategies. "The AI Economist" by (Zheng et al., 2020) uses multi-agent RL to design optimal tax policies by simulating an economy where both AI agents and a social planner learn at the same time. Dwarakanath et al. (2024)'s

"ABIDES-Economist" is a multi-agent simulator featuring heterogeneous households, firms, a central bank, and government agents that can learn. These studies show the potential of learning agents to discover more realistic and adaptive economic behaviours.

ABMs are also powerful tools for policy analysis, allowing for the evaluation of policy impacts in a controlled computational environment (*ex-ante analysis*, meaning measuring the impact of policies before they are done in the real world). Salle et al. (2013) utilized an ABM to assess inflation targeting regimes within a learning economy, showing the importance of central bank credibility. The ability to model heterogeneous responses to policy changes is an advantage of ABMs over representative-agent models.

While the application of ABM in economics has grown, there is still a need for models that can integrate diverse agent types and their interactions to provide insights for both theoretical exploration and policy analysis. Many existing models either focus on general macroeconomic results with simplified agent behaviors or specific micro-aspects, such as tax policy (Zheng et al., 2020) or market microstructure (Yang et al., 2025) in isolation.

This paper introduces **Economixim**, an agent-based economic simulation developed using the Mesa framework. Economixim is designed to model a mixed economy comprising Firms (differentiated by product type and operational area), Households (that aggregate individual Person agents and consume Firm products), Persons (characterized by skills and employment status), an Intermediary Firm (representing input suppliers), and a Government agent (overseeing fiscal policy and economic monitoring). These agents make adaptive decisions related to consumption, savings, production, pricing, inventory management, employment, wages, and public spending. The primary objective of Economixim is to create a base platform for exploring how granular interactions and policy interventions lead to emergent macroeconomic results like market equilibrium dynamics, price and inventory adjustments, labor market fluctuations, shifts in social welfare, and the evolution of government fiscal balances. By adhering to the ODD (Overview, Design Concepts, Details) protocol (Grimm et al., 2006, 2020), Economixim aims to provide a replicable framework for exploring the dynamics of a multi-agent economy.

The model code can be found here:
https://github.com/yyusufcoskun/Economyxim

2. MODEL DESCRIPTION

The model description attempts to follow the ODD (Overview, Design Concepts, Details) protocol for describing individual and agent-based models (Grimm et al. 2006), as updated by Grimm et al. (2020).

2.1. OVERVIEW

2.1.1. PURPOSE AND PATTERNS

Purpose

The primary purpose of the **Economixim** model is to simulate an economy involving firms, households, persons, and a government agent to explore emergent economic outcomes resulting from micro-behaviors. Specifically, it explores how granular agent decisions regarding consumption, savings, production, pricing, inventory management, employment, wages, taxation, and public spending lead to system-level phenomena like market equilibrium, price dynamics, labor market dynamics, inventory fluctuations, and overall economic indicators like GDP and inflation. The model serves as a virtual laboratory for theoretical exploration of economic interactions and is designed to help better understand economic theories, in an unbiased and agent-driven simulation environment.

Population characteristics such as birth, death, and marriage are not included in this model. The creation of new product lines from firms, supply chain dynamics and bankruptcy are also excluded from the model. A national banking system and the modeling of international trade were not designed due to such designs pushing the project beyond its scope.

Patterns

The evaluation of the model’s success is done by way of comparing results to multiple emergent patterns commonly observed or expected in real-world economies:

1. **Market Equilibrium:** If prices and production levels adjust over time in response to supply and demand. Firms are expected to raise prices when demand is high relative to inventory and lower them when inventory is high or sales are poor. Successful reproduction of this pattern indicates that the agent decision rules for pricing and production are capable of leading to market-clearing.
2. **Inventory Dynamics:** If firm inventory levels fluctuate based on demand and production decisions, differentiating between necessity and luxury goods. It is expected that firms aim to avoid excessive stockouts or overstocking through production adjustments
3. **Household Income:** The emergence of different income brackets among households based on agent skill, employment and wage.
4. **Consumer Spending Behavior:** If household spending patterns change based on income brackets and available goods. This will show the model's ability to represent the impact of wealth on consumption structure.
5. **Firm Profitability:** The distribution of profits across different types of firms (necessity vs. luxury) and firm areas (physical, service etc.). This pattern will validate the model's ability to simulate competitive dynamics and the financial performance of heterogeneous businesses.

6. **Inflationary Pressure:** If the general price level in the economy changes over time in response to aggregate demand, supply metrics, and firm production adjustments.
7. **Labor Market Dynamics:** If the unemployment rate fluctuates based on firms' hiring and firing decisions driven by profitability and labor needs, and if a person's skill contributes to the nature of the labor market. This will measure the model's representation of the labor market, which is crucial for understanding income and in turn demand generation.
8. **Welfare:** How income, consumption, saving and public spending levels affect social welfare.

2.1.2. ENTITIES, STATE VARIABLES, AND SCALES

Entities & State Variables

The following agents are included in the model: government, firms, households and persons. Banks, unions, supply chain members, and international trade mechanics are not included in the model as their impact would only be realised outside of the project scope.

The state variables of entities are defined as "variables that are not calculated from other state variables and do not include variables that are readily calculated from other variables" according to the ODD protocol. The protocol also states that a state variable defines "how an entity's state varies over time or a distinguishing value of entities of the same type". The following descriptions are made to match this definition of a state variable.

Government Agent is a single entity that represents the governmental body responsible for fiscal policy. Although its name suggests purely governmental affairs, this agent also functions as the country's central bank in the model. It exists to simulate the impact of government taxation and spending on the economy, firm behavior and household welfare. It also serves as a collector of aggregate economic statistics such as GDP, inflation, and unemployment. Almost all variables this agent possesses are calculated from other variables.

Variable Name	Variable type and units	Variable range	Represents
reserves	Currency, Float, Dynamic	Real numbers	Government's financial reserves

Firm Agents are multiple entities representing individual firms that engage in production and sales. These firms also interact with the labor market by hiring/firing *Person Agents* (as described in the following entity descriptions). Firms can be of two types: necessity and luxury. Each firm type also has "business areas" (e.g. physical, service, technical), which are abstractions created to complement *Person Agent skill* and provide richer labor market strategies. They exist to model the supply of the economy, labor demand and the dynamics of different market sectors. Their adaptive behaviors are crucial for price information and market equilibrium.

Variable Name	Variable type and units	Variable range	Represents
firm_type	String, Static	Necessity - Luxury	Firm's type
firm_area	String, Static	One of: physical, service, technical, creative, social, analytical	Business area of firm
capital	Currency, Float, Dynamic	Real numbers	The firm's accumulated financial wealth
inventory	Units, Integer, Dynamic	>= 0	Current stock of unsold goods
production_capacity	Integer, Dynamic	>= 0	Maximum production per step
production_level	Percent, Float, Dynamic	0.1-1.0	Proportion of capacity used for production
production_cost	Currency, Float, Static	>= 1.0	Cost of producing one unit of a product
product_price	Currency, Float, Dynamic	> *min_price*	Selling price per unit
markup	Float, Dynamic	>= 0.5	Multiplication added to cost per unit to determine selling price
initial_employee_target	Integer, Static	Different for every firm	Firm's employee count at the beginning of the simulation run
num_employees	Integer, Dynamic	>= 1	Number of *Persons* employed
entry_wage	Currency, Integer, Static	> 0	Base wage for entry-level employees

Intermediary Firm Agent is a firm that represents the "raw material" producers that *Firm Agents* buy from. It receives demand from *Firm Agents*, and distributes their revenue as wages to their employees. It exists to simulate the "raw materials" sector and to connect the circular cash flow by keeping *Firm* production costs inside the economy. It has no variables that suit the definition previously made for state variables, all variables are a result of a calculation, which will be discussed in the Submodels section.

Household Agents are multiple entities that represent households in an economy. Each household is composed of one or more *Person Agents*. Households aggregate *Person* income, and make consumption and saving decisions. The aggregation of *Person* data combined with specific household variables also result in a *social welfare* calculation made on a per house basis. They exist to simulate households in an economy, where income is aggregated and consumption decisions are made, and to serve as a unit for measuring social welfare.

Variable Name	**Variable type and units**	**Variable range**	**Represents**
num_people	Integer, Static	1 - 5	Number of people/number of *Persons* to be generated per household
income_tax_rate	Percent, Static	0.15, 0.20, 0.27 (based on income bracket)	Income tax rate applied to household income, set by *Government*

Person Agents are multiple entities that each belong to a *Household Agent*. Persons represent individuals in the labor force, and possess skill levels in a certain area (i.e. areas that are matched one-to-one with *Firm Agent* business areas), seek employment, and earn wages. Persons can also improve their skill level if below the entry-level threshold. They exist to model individual-level characteristics like skill, employment status, and wage, which in turn affect *Household* calculations.

Variable Name	**Variable type and units**	**Variable range**	**Represents**
household	Object Reference, Static	N/A	*Household* to which the Person agent belongs to
employer	Object Reference, Dynamic	N/A	*Firm* to which the Person agent works for
skill_type	String, Static	One of: physical, service, technical, creative, social, analytical	Skill area of the Person agent
skill_level	Integer, Dynamic	1-100	Skill level of the Person agent in relevant *skill_type*
job_seeking	Boolean, Dynamic	True/False	Whether the person

	(initially True by default, however *Firm* initialization makes sure hired Persons set it as False)		is actively looking for a job
wage	Float, Dynamic	>= 0	Current wage, 10000 if unemployed (*Government* unemployment benefit)
labor	Float, Static	skill_level/random(3, 5)	Labor addition to *Firm* production capacity
work_hours	Integer, Dynamic (although in its current state, every Person has the same static value - 40)	30 - 45	Preferred weekly work hours
job_level	String, Dynamic	One of: entry, mid, senior	Seniority in *Employer Firm*, None if unemployed

Economy Model is the environment of the simulation. It manages the collection of agents, holds the *data collector,* and controls the simulation.

Variable Name	**Variable type and units**	**Variable range**	**Represents**
current_step	Integer, Dynamic	0 - 60	The current step count of the simulation run

Scales

The model runs at a quarter time step (i.e. 3 months) as economic reports are generally described in quarters. Simulations were run for 60 quarters which amount to 15 years. The model does not explicitly represent geographic space. Interactions (like purchasing or hiring) are not based on spatial proximity. The model is spatially abstract to simplify the model and focus on the core economic interactions between agents, rather than the complexities of spatial economics.

2.1.3. PROCESS OVERVIEW AND SCHEDULING

This section will describe what the model does as it executes: which entity or entities, execute which process, that update which aspects of the simulation, and the order in which the entities execute the process.

The sequence of actions taken during initialization are discussed in the relevant chapter.

The scheduling of processes within a single simulation step is as follows:

1. **Government Agent - Fiscal and Economic Monitoring Phase**
   i. The `GovernmentAgent` executes its `step` submodel.
   ii. Inflation is calculated via the `_calculate_inflation_rate` submodel.
   iii. Welfare payments (unemployment benefits and low-income transfers) are calculated and distributed via the `_calculate_and_distribute_unemployment_payments` and `_calculate_and_distribute_low_income_transfers` submodels. Government reserves are updated.
   iv. Government spending on necessity goods is executed via the `_execute_government_necessity_spending` submodel. Government reserves and records of purchases are updated.
   v. Taxes (household income and corporate) are collected via the `_collect_taxes` and `_collect_corporate_taxes` submodels, respectively. Both functions collect taxes from the values *Households* and *Firms* provide in the previous step. Government reserves are updated.
   vi. Key economic indicators (unemployment rate, GDP, Gini coefficient) are calculated via the `_calculate_unemployment_rate`, `_calculate_gdp`, and `calculate_gini_coefficient` submodels, respectively.

2. **Household Agents - Consumption Phase**
   i. All `HouseholdAgent`s execute their `step` submodel.
   ii. Each household calculates its income, determines income and wealth brackets
   iii. Based on their wealth bracket, *Household* makes consumption decisions for necessity and luxury goods. This involves interactions with `FirmAgent`s via the `_calculate_cost_and_buy` and `_spend_on_luxuries` submodels (which in turn call `FirmAgent.fulfill_demand_request`).
   iv. Household financial metrics (expenses, savings, debt) and welfare indicators (health, overall welfare) are updated.

3. **Firm Agents - Production and Adaptation Phase**
   i. All `FirmAgent`s execute their `step` submodel.
   ii. Firms update historical price records.

iii. Production operations occur: firms produce goods based on their current *production_level*, adding to *inventory*.
iv. Total labor and wage costs are calculated, and production material costs are sent (as demand) to the IntermediaryFirmAgent via its receive_firm_demand submodel.
v. Financial calculations are performed: *revenue* (from sales made to households and government), *profit* (after accounting for costs and corporate taxes paid to the government), and *capital* are updated. *revenue_per_employee* is calculated.
vi. Historical metrics (*demand_history*, *profit_history*, *average_demand*) are updated, and step-specific counters are reset.
vii. Firms adapt their operations:
   a. *product_price* is updated, via the adjust_price submodel.
   b. *production_level* is updated, via the adjust_production submodel.
   c. Workforce size (*employees*) is adjusted (hiring or firing), via the adjust_employees submodel (which calls hire_new_employee or fire_least_productive).
viii. *units_sold_this_step* and *total_requested_this_step* are reset.

4. Intermediary Firm Agent - Revenue and Wage Distribution Phase

i. The IntermediaryFirmAgent executes its step submodel.
ii. Demand accumulated from Firms via the receive_firm_demand submodel is aggregated.
iii. Aggregated demand is turned directly into *revenue*.
iv. *revenue* is divided by *num_employees* and distributed as wages to employees.
v. Demand is reset.

5. Person Agents - Skill and Status Update Phase

i. All PersonAgents execute their step submodel.
ii. If their skill_level is below the minimum skill required to get hired for an entry position in their skill_type (all *Persons* of this type are naturally unemployed):
   a. *job_seeking* is updated as False.
   b. They study to increase their skill by a marginal amount. (This cycle continues until they reach the minimum skill threshold.)

6. Data Collection:

- The model's DataCollector gathers specified agent-level variables.

After these 6 phases, the simulation advances 1 step.

The reasoning behind this scheduling order is to ensure each agent can interact with the necessary agents in a logical order.

The *Government* steps first to set the fiscal stage by setting tax brackets, collecting taxes based on the previous period's income/profit, and making initial expenditures (welfare spending and purchases). This provides a stable economic environment and initial conditions for the current step's household and firm decisions.

Households move next after determining their income including *Government* transfers, and generate a demand signal for *Firms*.

Firms then receive demand from *Households* and the *Government*. They fulfill this demand from their existing inventory, then produce to replenish their inventory based on past demand trends. Following production and sales, they calculate their financial statuses and adapt their pricing, production levels, and employment for the next period. This simulates businesses reacting to the market.

While *Firms* produce, they send their costs as demand to the *Intermediary Firm,* which acts as an abstraction for raw material purchases *Firms* make. This structure allows cash to not leak out of the economy, as *Intermediary Firms* channel the money back to the system as wages.

At the end, *Persons* update their skills, reacting to their *Household* status and labor market conditions shaped by *Firm* decisions.

This sequence provides decisions that are based on the most recent information available, while also maintaining a cause-and-effect chain typical of economic cycles.

2.2. DESIGN CONCEPTS

Basic Principles

The model is based on agent-based modeling principles applied to economics. It simulates an economy where market outcomes emerge from interactions between agents, which are: households that try to balance their welfare through spending patterns, profit-seeking firms and a government that keeps the balance of reserves and social welfare. The model puts forward the idea that system-level outcomes emerge from individual actions and interactions, and simulates this via *Person Agents*.

The model draws on established economic concepts like supply and demand, equilibrium theory, price-adjusting firms, labor market interactions, social welfare, and basic fiscal policy. The distinction between necessity and luxury goods, which influences firm inventory strategies and household consumption patterns, reflects basic consumer theory. The simulation is designed to validate (or invalidate, based on its results) these theories by providing a bias-free environment.

A key point of this model is the heterogeneity of agents. *Firms*, *Households*, and *Persons* differ from each other at the core by way of State Variables.

The model is implemented using the Mesa framework for agent-based modeling in Python.

Emergence

The primary results of the model can be viewed in two categories: emergent and imposed.

Behaviors that are a result of a combination of different interactions and decisions, and that are not based on singular rules (which generally have recurring or “looping” results) are classified as emergent behaviors.

Behaviors that result from strict rules that split values or create values are classified as imposed or rule-based behaviors.

The following are described as emergent behaviors:

1. **Pricing:** Pricing structures emerge from *Firms* reacting to inventory levels, costs, sales, and short and long term demand trends.
2. **Demand:** Pricing and the income level of *Households* directly impact how much demand is generated during the simulation. Low prices mean high demand, while high prices mean low demand.
3. **Market Equilibrium:** Due to the pricing and demand structures, market equilibrium emerges.
4. **Inflation:** Pricing decisions impact the inflation rate, which is the increase in price over time.
5. **Production:** Production levels, like pricing, emerge from *Firms’* decisions based on household and government demand trends. Also, as each employee adds a certain amount of `production_capacity` to the *Firm* via their `labor` variable, small changes can impact the production and sales of *Firms*.

6. **Firm Profitability:** The distribution of profits and whilst not coded, the “survival” or “failure” of firms, emerges from *Firms’* individual ability to manage costs, set prices, and manage their workforce.
7. **Employment:** Hiring and firing decisions emerge from *Firms* reacting to their own profits, the productivity levels of their employees and their interaction with the labor market which is created by *Persons* that seek jobs and their skill levels.
8. **Welfare:** The distribution of welfare emerges from individual employment statuses, wage, spending patterns, savings, preferred work hours, health levels, and government public spending.
9. **Gini coefficient:** The distribution of income based on employment and household size creates the base for a Gini coefficient calculation.
10. **Initial values:** The single most important aspect of this model’s emergent behavior is the initialization phase. Initial values such as household count, firm inventory, firm production capacity, government spending level; and values given to rules such as market pressure parameters (*how much* demand and inventory should affect pricing) completely change the simulation results even when a single change is made. The difficulty of initialization and the problems it brings will be discussed in the Initialization section.

The following are described as imposed behaviors:

1. **Income/Wealth Distribution:** While the distribution of income is based on skill, employment and household size, and is an emergent behavior, the discrete categorization of income and wealth brackets are imposed by calculations.
2. **Public Spending:** The *Government* is involved in public spending, which is calculated as a percentage of their reserves in every step. Depending on their reserves, less or more demand is generated in the economy, affecting all other aspects of the simulation.
3. **Tax Revenue:** The income bracket distribution discussed earlier splits *Households* into different tax rates, which in turn generate cash inflow for the *Government*.

Adaptation

Agents in Economixim display indirect objective-seeking adaptive behaviors. The following describes each **decision** an agent makes during its step, and should not be confused with each action an agent takes during a step, which was described in the Process Overview and Scheduling chapter.

Firm Agents

1. **Production Level Adjustment**
   Firms adjust their production level during this adaptation.

   In order to make this decision, *Firms* use the following inputs:
   - **Average Demand:** Is a weighted average of the demand received by the Firm over the last 5 steps. Demand from steps closer to the current step are heavier in weight, while demand from steps further back from the current step are lighter.
   - **Demand to Inventory Ratio:** Is calculated by dividing average demand by current inventory. This results in a value that describes how much inventory can cover the demand.
   - **Sell Through Rate:** Is calculated by dividing units sold in that step to units produced in that step. This results in a value that describes what percentage of newly produced units are sold.
   - **Firm Type:** *Firms* have different minimum production level thresholds and crisis management strategies based on their firm type - necessity or luxury.

   After this decision, the value of the `production_level` variable changes, ranging from the minimum threshold set by firm type to 1.

2. **Price Adjustment**
   Firms adjust their product price and markup levels during this adaptation.

   In order to make this decision, *Firms* use the following inputs:
   - **Demand History:** Is a list of the demand received over the last 5 steps.
   - **Short Term Trend:** Is calculated by dividing the last 2 steps' demand values to each other, using the demand history list. Represents the change in demand in the near past.
   - **Long Term Trend:** Is calculated by dividing the average of the first 2 entries in the demand history list by the average of the last 2 entries. Represents the change in demand over time.
   - **Inventory to Demand Ratio:** Is calculated by dividing current inventory to demand received this step. This results in a value that describes how much inventory there is compared to demand.
   - **Sell Through Rate:** Is calculated by dividing units sold in that step to units produced in that step. This results in a value that describes what percentage of newly produced units are sold.

- **Market Pressure:** Is a value between -1 and 1, and is calculated via aggregation after applying rules based on the previous 4 inputs. Represents a "push" in price depending on the results of the previous inputs.
- **Firm Type:** *Firms* have different crisis management strategies and market pressure multiplications based on their firm type - necessity or luxury.

After this decision, the value of the `markup` variable changes, which in turn affects the `product_price` variable due to how it's calculated.

3. **Employee Hiring and Firing**

Firms hire or fire employees during this adaptation.

In order to make this decision, *Firms* use the following inputs:

- **Profit History:** Is a list of profit that *Firms* have made over the last 4 steps. *Firms* monitor if they are at a loss or a profit. Then, they check if their profits or losses are stable, growing or shrinking. *Firms* only fire when they are at a loss, and only hire when they are profiting, subject to sub-decisions.

When firing, *Firms* use the following inputs:

- **Employee Skill:** Is derived from the `skill_level` variable that each *Person* possesses.
- **Employee Labor:** As mentioned in the State Variables chapter, each *Person* has a *labor* value which is `skill_level` / *random(3,5)*. This represents an additional *production_capacity* value to the value *Firm* already has.
- **Employee Productivity:** Is calculated as `skill_level` * `labor` / `wage`. Represents a virtual "productivity" score of employees.

When hiring, *Firms* use the following inputs:

- **Minimum Skill Levels:** Is a list of minimum skill levels required for each *job_level* of each `firm_area`. (e.g. "technical": "senior": 80, "mid": 60, "entry": 40)
- **Skill Mix:** Is a list of what percentage of each `job_level` there should be in a *Firm*, for each `firm_type`.
- **Skill:** Is derived from the `skill_level` variable that each *Person* possesses.

After making a hiring or firing decision, *Firms* update the value of `num_employees`, and in turn *costs* increase or decrease according to employee `wages`.

Household Agents

1. **Consumption Decisions**
Households select which *Firm* they will buy from and how much they will buy during this adaptation.

When selecting *Firms*, *Households* use the following inputs:
- **Necessity Target:** Every *Household* must buy from *Firms* with `firm_type` = "necessity". How much they will buy is determined via the `total_necessity_target` variable. It is calculated by multiplying the `necessity_spend_per_person` (whose value is 57750) and `num_people` variables.
- **Wealth Bracket:** Represents which wealth bracket a *Household* belongs to and is calculated on each step.
    - If a *Household* belongs to the "Low" wealth bracket, they use the *product_price* variable of *Firms* to find the lowest 25% priced firms and select randomly from among them. *Households* in the "Low" wealth bracket can only buy from *Firms* that have the `firm_type` = "necessity".

    - If a *Household* belongs to the "Middle" or "High" wealth brackets, they select a random *Firm* to buy from. These *Households* not only buy necessities, but can also buy from *Firms* with `firm_type` = "luxury".

      After necessity spending is complete, "Middle" wealth *Households* spend between 60%-100% of their remaining balance, while "High" wealth *Households* spend between 80%-100% of their remaining balance on luxury *Firms* which they randomly select, from 2 randomly selected luxury `firm_area's`.

After these consumption decisions, Households update their `necessity_fullfilment` and `household_step_expense` variables.

Person Agent

1. **Studying**
   Person's grow their skills during this adaptation.

   In order to make this decision, *Persons* use the following inputs:
   - Skill Level: Represents a *Person*'s skill in a specific `skill_type` / `firm_area`. *Person*'s calculate and monitor their own skill.
   - Minimum Skill Levels: Is a list of minimum skill levels required for each `job_level` of each `firm_area`.

   If a *Person*'s skill level is below their own `skill_type`'s minimum skill level, they decide to study to increase their skill up to that threshold, and update their `skill_level` after this decision.

Government Agent

The *Government Agent* makes no adaptive decisions during a step.

Intermediary Firm Agent

The *Intermediary Firm Agent* makes no adaptive decisions during a step.

Objectives

Agents in this model do not generally employ direct objective-seeking by maximizing a specific, mathematically defined objective function (like utility or long-term profit). Instead, their adaptive behaviors (see Adaptation) directly affect these objectives.

The ***Government Agent*** aims to maximise government reserves, maximise social welfare, and reduce the Gini coefficient.
Firm Agents aim to maximise profits, minimise excessive inventory or stockouts, achieve long-term growth, and balance employee costs and productivity. They do this by adjusting prices, adjusting production, hiring/firing employees, and keeping track of employee productivity.

Household Agents aim to maximise welfare. As each *Household* is composed of ***Person Agents***, *Person* decisions and outcomes directly affect *Household* objectives. While *Households* decide on how much to spend and what to spend it on, *Persons* get employed, earn money, and improve their skills. *Person*s aim to be employed and earn income. Welfare is calculated as:
`self.household_step_income_posttax`*0.3 + `self.household_step_expense`*0.2 + `self.total_household_savings`*0.2 + `self.health_level`*0.3

Learning

Learning is not implemented in this model.

Prediction

While prediction is not explicitly implemented as defined in the ODD Protocol, *Firms* taking previous short term and long term demand is meant to simulate future demand prediction. *Firms* also decide to hire/fire employees based on their profit history. Consistent profit or loss trends lead to hiring and firing decisions, as *Firms* internally assume that these trends will continue or that they need intervention.

The reason for the lack of explicit prediction with a technique such as regression is that such a function would expand the complexity of this model beyond its scope.

Sensing

While the ODD protocol states that the Sensing chapter should discuss how agents can sense *state variables*, this version will also discuss which non-state variables are sensed by other agents.

All agents are aware of their own variables, as a result, only variables that they can sense from other entities will be discussed.

Government Agent

Sensed variables from Firms: `product_price`, `price_two_steps_ago`, `firm_type`, `tax_paid_this_step`, `firm_area`, `produced_units`

How can they sense these variables: The *Government* is assumed to have perfect knowledge over the entire *Firm* industry as *Firms* "share" these values with the *Government* for legal documentation.

Sensed variables from Households & Persons: `income_bracket`, `household_step_income`, `employer`, `job_seeking`, `num_people`, `necessity_spend_per_person`, `total_household_savings`

How can they sense these variables: The *Government* is assumed to have a statistics institution (e.g. TURKSTAT) that collects this knowledge from *Households* and *Persons*.

Firm Agents

Sensed variables from Persons: `employer`, `skill_level`, `skill_type`, `job_seeking`, `job_level`, `produced_units`, `labor`, `wage`

How can they sense these variables: *Firms* are assumed to know *Person* information from portals similar to career websites. They are also assumed to know the variables of their own employees as they need them to calculate expenses.

Sensed variables from Households: `demand`

How can they sense these variables: *Firms* are able to sense demand from *Households* as they send demand during their own step.

Sensed variables from Government: `demand`, `corporate_tax_rate`

How can they sense these variables*: Firms* are able to sense demand from the *Government* as they send demand during their own step. Firms also sense the corporate tax rate the *Government* sets as they need to pay taxes.

Household Agents

Sensed variables from Persons*:* all variables

How can they sense these variables*: Households* know *Person* variables as *Households* are collectives composed of *Persons*.

Sensed variables from Firms*:* `firm_type`, `product_price`, `inventory`

How can they sense these variables*: Households* know *Firm* variables as they are assumed to see the type of item they're getting, the price of it, and if they can find it on the shelves of the supermarket (`inventory`).

Sensed variables from Government*:* `tax_rates`

How can they sense these variables*: Households* know their income tax rate as they need to pay taxes to the *Government* based on those rates.

Person Agents

Sensed variables from Households*:* `unique_id` (Households' internal ID number that Mesa attaches automatically)

How can they sense these variables*: Persons* naturally know which *Household* they belong to.

Sensed variables from Firms*:* `min_skill_level`, `wage`, `job_level`

How can they sense these variables*: Persons* are assumed to know the minimum skill level due to virtually searching through jobs from job portals that post job listings. They also know their wages due to the need to get paid and contribute to the *Household*.

Intermediary Firm Agent

Sensed variables from Firms*:* `production_cost`

How can they sense these variables*: Intermediary Firms* are able to sense demand from *Firms* as they send their costs as demand during their own step.

Interaction

There are 2 kinds of interactions in this model: direct and mediated.

The **direct** interactions in this model are as follows:

1. **Household ↔ Firm**
   - *Households* sense firm data to make purchasing decisions. Specifically, the *Household* identifies potential firms based on their `firm_area`, queries their `product_price`, and checks their `inventory` status.
   - *Households* send a purchase request to *Firms* with the `fulfill_demand_request` function. This function on the *Firm Agent* takes the number of units the household wishes to buy as an argument.
   - *Firms* directly sell products to *Households*. If the *Firm* has sufficient `inventory`, it reduces its `inventory` and increases its internal `units_sold_this_step` counter, completing the transaction.
2. **Person → Household**
   - Persons' `wage` directly contributes to the `total_household_income` of the *Household* they belong to.
3. **Firm ↔ Person**
   - *Firms* hire persons by finding a suitable candidate and directly modifying their state variables: `employer` is set to the firm object, `job_seeking` is set to `False`, and `wage` and `job_level` are assigned.
   - *Firms* fire persons by removing them from their `employees` list and setting the person's `employer` attribute back to `None`, their `wage` to 0, and `job_seeking` to `True`.
   - Employed *Persons* directly provide labor for *Firms*.
4. **Government ↔ Household**
   - The *Government* directly taxes *Households*.
   - The *Government* provides direct financial support to households. It identifies *Households* in need (i.e. that can't meet their necessity spending target) and directly increases their `total_household_savings` via a transfer payment. This in turn affects *Household* welfare.
   - Tax payments made by *Households* affect `reserves`, therefore affecting general fiscal policy.
5. **Government ↔ Firm**
   - The *Government* collects corporate taxes from *Firms* which affect `reserves`. *Firms* calculate their own tax paid which reduces their profit.
   - The *Government* makes direct purchases from necessity *Firms*, impacting their sales and inventory.
6. **Government → Person**
   - The *Government* identifies all unemployed and job seeking (i.e. the definition of unemployment in macroeconomics), and provides unemployment benefits by directly setting such *Persons'* `wage` attribute to the unemployment payment amount.
7. **Intermediary Firm → Person**

- The *Intermediary Firm* initializes by hiring *Persons*, and setting their wage attribute to their `revenue` / `num_employees`. They also set their `employer` to the firm object, `job_seeking` to `False`, and `job_level` to `Entry`.

8. **Firm → Intermediary Firm**
    - *Firms* calculate their `production_costs`, and send this value as a demand signal to the *Intermediary Firm*.
    - The *Intermediary Firm* converts demand into `revenue` and distributes it as `wage`. This way, circular cash flow between *Firms*, *Households*, *Government* and the *Intermediary Firm* is achieved.

The **mediated** interactions in this model are as follows:

1. **Household ↔ Firm:**
    - The supply-demand interaction between the two agents influence *Firms'* adaptive behaviors (i.e. pricing, production, hiring) and therefore affect the market for all *Households* and *Firms*.
2. **Person → Government:**
    - Employment status of *Persons* affect the `unemployment_rate` variable, which the *Government* senses.
3. **Firm → Government:**
    - Total production and pricing structure of *Firms* affects the `GDP` and `inflation_rate` variables, which the *Government* senses.
4. **Firm ↔ Firm**:
    - *Firms* compete for *Household* and *Government* demand, which is a shared resource. One *Firm's* pricing, production, and inventory levels indirectly affect the sales opportunities and profitability of all other firms in the same market segment.
    - *Firms* also indirectly compete for *Persons* in the labor market, although this primarily occurs because of the pool of job seekers and *Firms'* hiring criteria.
5. **Household ↔ Household:**
    - *Households* contribute to aggregate demand which creates competition among each other due to goods being limited.
    - They also interact with each other as each working *Household* member directly contributes to *Firm* performance, which affects pricing and production.
6. **Government ↔ Household:**
    - The Government and Households compete for Firm products, which are a shared resource.
7. **Person ↔ Person:**
    - *Persons* indirectly compete for available employment opportunities. One *Person's* successful hiring means that particular job opening is now closed for another *Person*.
8. **Firms ↔ Intermediary Firm Persons:**
    - The demand sent by the *Firm* to the *Intermediary Firm* determines their employee's wages. This means the production levels of *Firms* indirectly determine the income levels of *Persons* in the *Intermediary Firm*.

None of the interactions in this model are spatial.

Stochasticity

Stochasticity is used in several aspects of the model, primarily during initialization and in some agent decision processes.This method was utilised to further realise if patterns used to evaluate the model (see Patterns) evolved naturally, as it ensures heterogeneity across the agents, which is crucial for observing emergent dynamics and avoiding artificial uniformity. The other reason stochasticity is utilized is that modeling certain systems would add great complexity to the project, such as modeling production capacity of a *Firm* depending on its factory size. The stochastic nature of the initialization aims to create a complex economic landscape from which interactions and adaptations can unfold.

During initialization:

Household Agents

- `num_people`: The number of people variable is drawn from a random selection between 1 to 5 people, meaning each *Household* randomly starts the simulation run with 1 to 5 people assigned to them. The reason behind this choice is to have heterogeneity between *Household* populations without modeling the causes of variability (e.g. birth, death).

Firm Agents

- `production_capacity`: The amount of units a *Firm* can produce in one step is determined via a random selection between a given range. This is done to create different production powers between *Firms* and monitor their adaptation to their attributes.
- `production_cost`: The cost of producing one unit is determined via a random selection between a given range. The reason behind this is to simulate different costs *Firms* might incur during production.
- `entry_wage`: The wage of an entry level employee, which is also a base for the entire wage structure, is determined via a random selection between a given range. This is done to model how *Firms* with different levels of capital pay their employees.
- `initial_employee_target`: This variable represents the amount of employees a *Firm* starts the simulation off with. This is done to simulate different employment structures of *Firms*.
- **Initial Workforce Population:** While running the `_populate_initial_workforce` function which adds *Persons* to a *Firm* at initialization, *Firms* shuffle possible candidates to introduce randomness in who gets hired if multiple candidates are available for hire.

Person Agents

- `skill_type`: The skill area a *Person* has is determined randomly between 6 values: physical, service, technical, creative, social, analytical. The reason behind this is to create employees with different professions.

- `skill_level`: The skill level a *Person* has is determined via a random normal distribution between 0-100, with a mean of 50 and a standard deviation of 15. This is to simulate different natural aptitudes between people.

Intermediary Firm Agent

- **Initial Workforce Population:** Similar to *Firms*, the *Intermediary Firm* shuffles possible candidates to introduce randomness in who gets hired if multiple candidates are available for hire while running the `_populate_initial_workforce` function.

After initialization, during agent decisions:

Household Agents

- **Finding Cheapest Firm:** When “Low” wealth *Households* are selecting a necessity *Firm* to purchase from, they find the cheapest 25% of *Firms*, and select one randomly.
- **Selecting a Firm:** When “Middle” or “High” wealth *Households* are selecting a necessity or luxury *Firm* to purchase from, they select a random *Firm* from the list of *Firms*.
- **Selecting a Luxury Type:** When “Middle” or “High” wealth *Households* are selecting a luxury *Firm* to purchase from, they must select two of four luxury types. This selection is done randomly.
- **Luxury Spend Percentage:** When doing their luxury spend, “Middle” wealth *Households* spend 60 to 100% of their remaining funds, and “High” wealth *Households* spend 80 to 100% of their remaining funds. The exact percentage to be spent varies and is selected randomly between these ranges.

Firm Agents

- `person_to_hire`: When hiring if there are multiple equally suitable top candidates, one is chosen randomly from the top half of candidates sorted by skill level. This simulates the “personal” aspect of hiring, where factors other than skill are considered to determine the right candidate.

Government Agent

- **Selecting a Firm:** When purchasing necessity goods, potential *Firms* are shuffled randomly before the *Government* iterates through them to make purchases.

Collectives

The model includes 2 collectives: the *Households* and the employees.
Each *Household* is an aggregation of one or more *Persons*, and are explicitly represented. Employees are the list of *Persons* working for a *Firm*, but are not explicitly represented and do not have their own behaviors and state variables.

The *Household* entity has its own state variables (see State Variables) and behaviors. This collective is included in the model as real-world households are also composed of multiple members contributing to the household differently while making collective decisions for the benefit of the entire household. The *Household*'s state is affected by the states of its *Persons* (their wages). Conversely, the household's overall financial status and decisions affect its members (e.g., contributing to the `welfare` and `health_level` which are calculated at the household level).

Simulating income and creating a labor market is made much easier by utilising individuality of *Person* agents rather than creating stochastic values for *Household* "blocks".

Observation

The model produces various outputs that capture the conditions emerging from agent interactions. Data for analysis is systematically collected at the end of each simulation step using the *Mesa DataCollector*. All observations are collected at the agent level, although the code implementation of *Government* data collection is made model-wide due to the way the Mesa framework operates. Outputs include both measures of central tendency and variability. This captures not just the average conditions but also the heterogeneity and inequalities across agents.

Variables collected from the ***Government*** agent are as follows:

- `reserves`: Tracked to monitor the *Government's* fiscal health and capacity for spending over time.
- `step_public_spending:` Observed to measure the *Government's* spending in each step.
- `unemployment_rate`: A key macroeconomic indicator, calculated to assess the health of the labor market.
- `gdp`: Measured as the total value of firm production, this variable tracks the overall economic output of the simulation.
- `step_tax_revenue`: Total tax collected in a step from *Households*, tracked to analyze government income and the tax burden on the economy related to *Households*.
- `step_corporate_tax_revenue`: Total tax collected in a step from *Firms*, tracked to analyze government income and the tax burden on the economy related to *Firms*.
- `inflation_rate`: A critical macroeconomic indicator, calculated from firm price changes to monitor price stability.
- `gini_coefficient`: Recorded to measure income inequality across all *Persons*, a key metric for social welfare analysis.

Variables collected from the ***Firm*** agent are as follows:

- `firm_type` and `firm_area`: Observed to analyze differentiated market behavior, profitability, and resilience between necessity/luxury firms and across different business sectors.
- `profit` and `revenue`: Tracked to evaluate *Firm* performance over time.
- `inventory`: Tracked to observe *Firm* inventory levels over time.
- `produced_units`: Recorded to track production over time.
- `product_price` and `markup`: Recorded to assess pricing dynamics, *Firm* strategy, and their contribution to inflation.
- `num_employees` and `revenue_per_employee`: Observed to understand labor market dynamics.
- `production_level`: Recorded to understand whether *Firms* adapt to demand as designed.
- `production_capacity`: Observed to assess whether production capacity has a correlation with revenue.
- `demand_for_tracking`: Tracked to measure consumer demand over time.
- `costs` and `capital`: Recorded to see a complete picture of a *Firm's* financial state, including its operational costs and accumulated wealth.

Variables collected from the ***Household*** agent are as follows:

- `income_bracket` and `wealth_bracket`: Recorded to observe the distribution of *Households* across different economic statuses and to analyze social inequality.
- `household_step_income` and `household_step_income_posttax`: Tracked to observe the financial inflow and disposable income of *Households*, which drives consumption.
- `household_step_expense`: Observed to analyze consumption patterns.
- `total_household_savings`: Observed to analyze savings behavior and cash balance of *Households*.
- `health_level` and `welfare`: Recorded as key measures of *Household* well-being, which is a primary outcome for assessing the model's socio-economic state.
- `debt_level`: Observed to monitor *Household* financial distress
- `num_people`, `num_working_people`, `num_seeking_job` and `num_not_seeking_job`: Tracked to understand *Household* composition and its labor force participation, which influences income and welfare.

Variables collected from the ***Person*** agent are as follows:

- `skill_level`: Tracked to observe skill distribution in the population and changes in skill level over time.
- `skill_type`: Recorded to understand how different skill sets fare in the market.
- `employer`: Observed to calculate unemployment rates. If an `employer` exists, *Person* is regarded as employed.
- `job_seeking`: Recorded to provide the raw data for calculating unemployment rates. If a *Person* is unemployed but not `job_seeking`, they are disregarded in the unemployment calculations.

- `wage`: Observed to track individual earnings, and to provide the basis for calculating income inequality and *Household* income.
- `job_level`: Recorded to track career progression, understand employment structure and how that affects *Household* `welfare`.

No "virtual scientist" or other specialized observation techniques are used to simulate empirical data collection biases. The observation process consists of recording the true state of agent and model variables at each step.

2.3. DETAILS

2.3.1. INITIALIZATION

The model is initialized with:

- 1 `GovernmentAgent`,
- 75 `FirmAgent`'s (unevenly split into different *`firm_area`*'s),
- 1 `IntermediaryFirmAgent`
- 1,000 `HouseholdAgent`'s
- 30,000 `PersonAgent`'s, who are assigned to Households and cleaned up later on.

During initialization, a creation sequence is run at the start of the simulation and proceeds as follows:

1. **Creation of *Persons***
   30,000 `PersonAgent`'s are created and placed in a temporary global list called `available_persons`.

2. **Creation of *Government* and *Firms***
   - The `GovernmentAgent` is created, followed by `FirmAgent`'s and the `IntermediaryFirmAgent`.

3. **Initial Hiring**
   - During the creation of `FirmAgent`'s and the `IntermediaryFirmAgent`, both agents run their internal *`_populate_initial_workforce`* function that hires their starting employees, from the `available_persons` list.

4. **Creation of *Households***
   1000 `HouseholdAgent`'s are created, each with a randomly assigned target population size.

5. **Household Assignment**
   The *`_assign_persons_to_households`* function is called in the model. This procedure assigns *Persons* to *Households* in this sequence:
   - Model lists employed, unemployed *Persons* and all *Households*.
   - Shuffles all three lists randomly.
   - Iterates through each *Household* and places 1 employed *Person* into each *Household* until there are no employed *Persons* left to place
   - Once all employed *Persons* are placed, remaining slots in *Households* (that were determined by their *`num_people`* attribute) are filled up with unemployed *Persons*.

6. **Cleanup**
   Any `PersonAgent`'s remaining in the `available_persons` list after the assignment process are deleted from the simulation as the initial targets have been reached.

Most initial state variables of entities are determined stochastically (see Stochasticity). Apart from the values described in the Stochasticity section, the initial values given for each entity's state variables are as follows:

Government Agent

- The *Government* `reserves` are initialized to 100,000,000

Firm Agents

- *`firm_area`* distribution:
    - 25 Necessity - Physical *Firms*
    - 25 Necessity - Service *Firms*
    - 10 Luxury - Technical *Firms*
    - 5 Luxury - Creative *Firms*
    - 5 Luxury - Social *Firms*
    - 5 Luxury - Analytical *Firms*
- *Firm* `capital` is initialized with 1,000,000 currency units for all *Firms*.
- *Firm* *`inventory`* is initialized with *`production_capacity`* * 2.
- *Firm* `markup` initialization for each *`firm_area`*:
    - Physical: 2
    - Service: 3
    - Technical: 7
    - Creative: 6
    - Social: 5
    - Analytical: 6
- *Firm* `production_level` is initialized as 1.

Intermediary Firm Agent

- This firm is initialized with an *`initial_employee_target`* of 240 employees.

Household Agents

- Each Household is initialized with an *`income_tax_rate`* of whichever rate the Government assigned them in accordance with their *`income_bracket`*.

Person Agents

- Each Person has their *`employer`* and *`household`* initialized by the creation sequence described earlier.
- Each Person is initialized with a *`job_seeking`* status of `True`.
- Each Person is initialized with a *`work_hours`* value of 40.

With this model, the goal of the experiments are both to understand the effects of the initial conditions and the effects of processes happening after initialization. This creates a conflict of interest within the model design. On one hand, how the model is initialized affects the results greatly. On the other hand, where the results converge in later steps of the simulation is extremely important. The hardest part about this model is the initial value calibration. Between runs, changing the initial values and the ranges in which the random selection happens is very difficult to calibrate and would require (a) large amounts of time, (b) large amounts of computational power, and (c) machine learning capabilities to automate and find the best calibration result. Unfortunately, due to the scope and time limit of this project, this was not possible. Initial values and the ranges of stochastic selection are not perfect, and this imperfection has affected the numerical results discussed in the Results chapter greatly. While the system of this model is comprehensive and logical, the results are skewed by poor initialization.

The rationale behind the stochastic nature of the initialization is further discussed in the Stochasticity section.

The hiring of an initial workforce by *Firms* is done so the economy doesn't start with universal unemployment.
Each *Household* having at least one employed *Person* ensures every *Household* is part of the economy, and is capable of earning and consuming money.
Firms start with `capital`, *Households* with some savings, and the *Government* with `reserves`, providing initial liquidity.
Firms calculate an initial price based on estimated initial `costs` and `markup`, rather than an arbitrary value.

The reasoning behind the number of created agents does not depend on real-world data but rather the real-world experience of the project author.

As previously discussed, the model does not include an explicit spatial environment. Therefore, no initialization of agent locations or spatial variables occurs.

2.3.2. INPUT DATA

The model primarily uses internally generated parameters and stochasticity for its dynamics rather than relying on external time-series input data to drive changes during a simulation run.

All financial data inside the model, including relevant *Government* variables, are in Turkish Lira (TRY/₺) format.

While not set in source-backed real-world data, prices and wages inside of the model are also generated from the real-world experience of the project author, and are meant to simulate the Turkish economy.

2.3.3. SUBMODELS

This section provides detailed descriptions of the function that constitute the core logic and behaviors of the agents in the Economixim simulation.

Firm Agents Submodels

1. **Initial Workforce Population (`_populate_initial_workforce`)**

   Purpose: To hire the initial set of employees for the firm when it is first created. This submodel ensures that firms start with a workforce appropriate to their area and target size.

   Inputs:

   - `target_count`: Desired number of initial employees for the firm.
   - `self.model.available_persons`: Global list of all `PersonAgent`s available for hire.
   - `self.firm_area`: The business area of the firm.
   - `self.skill_mix_config`: Configuration defining the target proportion of senior, mid, and entry-level employees for the firm's area.
   - `self.skill_type_matching_config`: Configuration matching firm area to required `PersonAgent` `skill_type`.
   - `self.min_skill_levels_config`: Configuration defining minimum `skill_level` for each job level in the firm's area.
   - `self.entry_wage`: The base wage for entry-level positions at the firm.
   - `self.wage_multipliers`: Multipliers applied to the `entry_wage` for mid and senior levels.

   Process:

   1. Iterate through job levels (senior, mid, entry) based on `skill_mix_config`
   2. For each job level, calculate the number of employees to hire
   3. Filter `self.model.available_persons` to find candidates who are `job_seeking`, have no `employer`, match the `target_skill_type` for the firm's area, and meet the `min_skill_for_job_level`
   4. Randomly shuffle the list of possible hires
   5. Hire candidates sequentially until the target for that job level is met or the overall `target_count` for the firm is reached

   Outputs/State Variables Updated:

   - `self.employees`: List of `PersonAgent`s employed by the firm is populated.
   - `self.num_employees`: Updated with the count of hired employees.

- For each hired `PersonAgent`:
    - `employer` is set to this firm.
    - `job_seeking` is set to `False`.
    - `job_level` is assigned.
    - `wage` is calculated based on `entry_wage` and `wage_multipliers`.

2. **Demand Fulfillment (`fulfill_demand_request`)**

    Purpose: To process purchase requests from `HouseholdAgent`s or the `GovernmentAgent`.

    Inputs:
    - `units_requested`: The number of units the customer wishes to buy.
    - `self.inventory`: The firm's current stock of goods.

    Process:

    1. Determine the number of units that can be fulfilled (`can_fulfill`) by taking the minimum of `units_requested` and `self.inventory`
    2. If `can_fulfill` > 0:
        - Decrement `self.inventory` by `can_fulfill`
        - Increment `self.units_sold_this_step` by `can_fulfill`
    3. Increment `self.total_requested_this_step` by `units_requested` (regardless of fulfillment)

    Outputs/State Variables Updated:
    - `self.inventory`
    - `self.units_sold_this_step`
    - `self.total_requested_this_step`

    Returns: Number of units actually sold/fulfilled.

3. **Production Adjustment (`adjust_production`)**

    Purpose: To adapt the firm's production level based on market conditions (demand, inventory) and firm type.

    Inputs:

    - `sold_units`: Number of units sold in the current step.
    - `self.labor_added_production_capacity`: Current total production capacity.
    - `self.inventory`: Current inventory level.

- `self.average_demand`: Calculated average demand.
- `self.produced_units`: Units produced in the current step.
- `self.firm_type`: "necessity" or "luxury".
- `self.production_capacity`: Base production capacity.

Process:

1. Handle zero capacity case
2. Calculate `demand_to_inventory_ratio` and `sell_through_rate`
3. Set `min_production_level` based on `firm_type` (0.3 for luxury, 0.2 for necessity)
4. If no demand history or average demand is zero, set production level based on a simpler inventory threshold
5. Otherwise, adjust `self.production_level` incrementally based on `demand_to_inventory_ratio` and `sell_through_rate`. High demand or good sell-through increases production; poor sales with high inventory decrease it
6. Special handling for luxury firms in crisis (low demand and low production level) to maintain a minimum viable production
7. Ensure `self.production_level` stays within `min_production_level` and 1.0

Outputs/State Variables Updated:

- `self.production_level`

4. **Price Adjustment (`adjust_price`)**

Purpose: To adapt the product's selling price based on market conditions, costs, and demand trends.

Inputs:

- `sold_units`: Units sold this step.
- `produced_units`: Units produced this step.
- `cost_per_unit`: Current cost to produce one unit.
- `self.inventory`: Current inventory.
- `self.total_requested_this_step`: Total demand this step.
- `self.demand_history`: History of demand.
- `self.firm_type`: "necessity" or "luxury".
- `self.average_demand`: Calculated average demand.
- `self.production_capacity`: Base production capacity.
- `self.min_price`: Minimum allowable price (based on initial costs).
- `self.production_cost`: Base material cost per unit.

Process:

1. Handle zero production case
2. Calculate `inventory_demand_ratio` and `sell_through_rate`
3. Calculate `short_term_trend` and `long_term_trend` from `demand_history`
4. Determine `market_pressure` based on a combination of `inventory_demand_ratio`, `sell_through_rate`, and demand trends. Luxury firms have more aggressive responses to oversupply
5. Include a crisis intervention for luxury firms (aggressive price cutting if very low demand and sell-through)
6. Calculate `markup_change` based on `market_pressure` (luxury firms in crisis have more aggressive markup reduction)
7. Update `self.markup`, ensuring it's at least 0.5
8. Calculate new `self.product_price` as `cost_per_unit * (1 + self.markup)`
9. Ensure `self.product_price` is not below a `flexible_min_price` (which is very close to material cost for luxury firms in crisis, otherwise it's the initial `min_price`)

Outputs/State Variables Updated:

- `self.markup`
- `self.product_price`

5. **Employee Adjustment (`adjust_employees`)**

Purpose: To adapt the firm's workforce size based on recent profitability.

Inputs:

- `self.profit_history`: List of profits from the last few steps.
- `self.employee_adjustment_cooldown`: Cooldown timer.

Process:

1. Check `employee_adjustment_cooldown`. If > 0, decrement and return
2. Require at least `profit_history_length` (4 steps) of profit data
3. Calculate `avg_past_3_steps_profit` (average of profits from t-1, t-2, t-3)
4. Determine if current profit (`p0`) is stable relative to `avg_past_3_steps_profit`
5. Decision logic:

   a. If at a loss (`p0 < 0`):
      i. If losses are worsening or stable compared to `avg_past_3_steps_profit`, call `fire_least_productive`
      ii. If losses are improving, monitor
   b. If profitable (`p0 >= 0`):
      i. If profits are increasing or stable compared to `avg_past_3_steps_profit`, call `hire_new_employee`
      ii. If profits are falling, currently monitor (no firing action implemented for this case)
6. If an adjustment (hire/fire) was made, set `employee_adjustment_cooldown` to 1 (effectively a 2-step cooldown)

Outputs/State Variables Updated:

- `self.employee_adjustment_cooldown`
- (Indirectly, via called functions) `self.employees`, `self.num_employees`, and `PersonAgent` states.

6. **Fire Employee (`fire_least_productive`)**

Purpose: To remove the least productive employee from the firm.

Inputs:

- `self.employees`: Current list of employees.
- `self.num_employees`: Current number of employees.

Process:

1. Return `False` if no employees or only one employee
2. Calculate productivity for each employee as (`skill_level * labor`) / (`wage` + 1e-6)
3. Sort employees by productivity (lowest first)
4. Select the employee with the lowest productivity (`person_to_fire`)
5. Update `person_to_fire`'s state: `employer = None`, `wage = 0`, `job_seeking = True`
6. Remove `person_to_fire` from `self.employees` and decrement `self.num_employees`

Outputs/State Variables Updated:

- `self.employees`

- `self.num_employees`
- State of the fired `PersonAgent` (`employer`, `wage`, `job_seeking`).

Returns: `True` if an employee was fired, `False` otherwise.

7. **Hire Employee (`hire_new_employee`)**

Purpose: To recruit a new employee based on skill needs and availability.

Inputs:

- `self.employees`: Current list of employees.
- `self.num_employees`: Current number of employees.
- `self.firm_area`: The firm's business area.
- `self.min_skill_levels_config`, `self.skill_mix_config`, `self.skill_type_matching_config`.
- `self.entry_wage`, `self.wage_multipliers`.
- `self.model.available_persons` (implicitly, by iterating `self.model.agents` to find job seekers).
- `self.previous_employees`: A set to track recently hired individuals to avoid immediate rehire.

Process:

1. Calculate the current mix of job levels (entry, mid, senior) in the firm
2. Determine the target job level most needed to achieve the `skill_mix_config`
3. Identify `min_skill_level` and `matching_skill_type` for the target job level
4. Find `available_job_seekers` from all agents in the model
5. Filter candidates based on `min_skill_level`, `matching_skill_type`, and not being in `previous_employees`. If no exact matches, relax the skill requirement slightly
6. If suitable candidates exist, sort them by `skill_level` (descending) and randomly choose one from the top half
7. Calculate `offered_wage` based on `entry_wage`, `wage_multipliers` for the job level, and a skill bonus
8. Update the hired person's state: `employer = self`, `wage = offered_wage`, `job_seeking = False`, `job_level`
9. Add the person to `self.employees`, increment `self.num_employees`, and add to `self.previous_employees`

Outputs/State Variables Updated:

- *self.employees*
- *self.num_employees*
- *self.previous_employees*
- State of the hired `PersonAgent` (*employer*, *wage*, *job_seeking*, *job_level*).

Returns: `True` if an employee was hired, `False` otherwise.

8. **Production Operations (Part of *step* method)**

Purpose: To produce goods based on the firm's production decisions and capacity.

Inputs:

- *self.production_capacity*: Base capacity.
- *self._calculate_total_labor()*: Calculated added capacity from employees.
- *self.production_level*: Current utilization percentage of total capacity.
- *self.production_cost*: Material cost per unit.
- *self.calculate_total_wage_cost()*: Total wage expenses.

Process:

1. Calculate *labor_added_production_capacity* by adding *_calculate_total_labor()* to *production_capacity*
2. Calculate *produced_units* by multiplying *labor_added_production_capacity* by *production_level* and rounding
3. Increase *self.inventory* by *produced_units*
4. Calculate total *costs* as the sum of *wage_costs* and *production_costs* (material costs for *produced_units*)
5. Send *production_costs* to the `IntermediaryFirmAgent` via its *receive_firm_demand* method

Outputs/State Variables Updated:

- *self.labor_added_production_capacity*
- *self.produced_units*
- *self.inventory*
- *self.costs*
- (Indirectly) `IntermediaryFirmAgent`*.demand_received_from_firms*

9. **Financial Calculations (Part of `step` method)**

Purpose: To calculate revenue, profit, and update capital.

Inputs:

- `self.product_price`: Current selling price.
- `sold_units` (derived from `self.units_sold_this_step` which is reset after this calculation).
- `self.costs`: Total costs calculated in production.
- `self.model.government_agent.government_purchases_from_firms_step`: Dictionary of purchases made by *Government*.
- `self.model.government_agent.corporate_tax_rate`: Corporate tax rate.

Process:

1. Calculate `self.revenue` as `self.product_price` * `sold_units`.
2. Calculate `pre_tax_profit` as `self.revenue` - `self.costs`
3. Determine `tax_paid_this_step`: If `pre_tax_profit` is positive, subtract revenue from direct government purchases (`amount_from_gov`) to find the `true_taxable_profit`. If this is positive, calculate tax as `true_taxable_profit` * `corporate_tax_rate`
4. Calculate final `self.profit` as `pre_tax_profit` - `tax_paid_this_step`
5. Update `self.capital` by adding `self.profit`
6. Calculate `self.revenue_per_employee`

Outputs/State Variables Updated:

- `self.revenue`
- `self.tax_paid_this_step`
- `self.profit`
- `self.capital`
- `self.revenue_per_employee`

10. **Update Historical Metrics and Reset Counters (Part of `step` method)**

Purpose: To maintain historical data for adaptive decision-making and reset step-specific counters.

Inputs:

- `self.total_requested_this_step`
- `self.profit`
- `self.revenue_per_employee`

Process:

1. Set `self.last_step_revenue_per_emp` to current `self.revenue_per_employee` (if first step, initialize it)
2. Append `self.total_requested_this_step` to `self.demand_history`, maintaining `demand_history_length`
3. Calculate `self.average_demand` using `demand_history` and `demand_averaging_weights`
4. Append `self.profit` to `self.profit_history`, maintaining `profit_history_length`
5. Reset `self.units_sold_this_step` and `self.total_requested_this_step` to 0 for the next step
6. Store `self.total_requested_this_step` in `self.demand_for_tracking` for data collection

Outputs/State Variables Updated:

- `self.last_step_revenue_per_emp`
- `self.demand_history`
- `self.average_demand`
- `self.profit_history`
- `self.units_sold_this_step` (reset)
- `self.total_requested_this_step` (reset)
- `self.demand_for_tracking`

<u>Household Agent Submodels</u>

1. **Update Employment Counts (`_update_employment_counts`)**

Purpose: To update the household's internal counts of working, job-seeking, and non-seeking members.

Inputs:

- `self.members`: List of `PersonAgent`s in the household.

Process:

1. Reset `self.num_working_people`, `self.num_not_seeking_job`, `self.num_seeking_job` to 0

2. Iterate through `self.members`:
    a. If a member has an `employer`, increment `self.num_working_people`
    b. Else if a member is not `job_seeking`, increment `self.num_not_seeking_job`
    c. Else (unemployed and job-seeking), increment `self.num_seeking_job`

Outputs/State Variables Updated:

- `self.num_working_people`
- `self.num_not_seeking_job`
- `self.num_seeking_job`

2. **Find Cheapest Firm (`_get_cheapest_firm`)**

Purpose: To find a firm in a given category that offers a low price, simulating a price-sensitive consumer.

Inputs:

- `firm_category`: The business area of firms to consider (e.g., "physical", "service").
- `candidate_firms` (optional): A pre-filtered list of firms. If `None`, searches all firms in the model.

Process:

1. Filter firms to consider: must match `firm_category`, have `product_price > 0`, and `inventory > 0`
2. If no suitable firms, returns `None`
3. Sort considered firms by `product_price` (cheapest first)
4. Select the top 25% cheapest firms (at least 1)
5. Randomly choose one firm from this cheapest tier

Returns: A `FirmAgent` object or `None`.

3. **Purchase Goods (`_calculate_cost_and_buy`)**

Purpose: To simulate the household purchasing goods from firms in a specific category, trying to meet a spending target.

Inputs:

- `firm_category`: The category of goods to purchase.
- `target_spend`: The desired amount to spend.
- `self.wealth_bracket`: The household's current wealth status ("low", "middle", "high").

Process:

1. Initialize `total_spent_for_category = 0.0` and `remaining_spend_target = target_spend`
2. Get a list of `potential_purchase_candidates` (firms of `firm_category` with positive price and inventory), and shuffle it
3. Loop while `remaining_spend_target` is meaningful and `potential_purchase_candidates` exist:
    a. Filter `potential_purchase_candidates` for firms that still have inventory (`currently_available_firms`)
    b. If `self.wealth_bracket` is "middle" or "high", randomly choose a firm from `currently_available_firms`
    c. Else (low wealth), call `_get_cheapest_firm` on `currently_available_firms`
    d. If a `chosen_firm` is found:
        i. Calculate `desired_units` based on `remaining_spend_target` and `chosen_firm.product_price`
        ii. If `desired_units` > 0, call `chosen_firm.fulfill_demand_request(desired_units)`
        iii. If units are bought, update `total_spent_for_category` and `remaining_spend_target`
    e. Remove the `chosen_firm` from `potential_purchase_candidates` for this purchasing round

Returns: The actual `total_spent_for_category`.

4. **Spend on Luxuries (`_spend_on_luxuries`)**

Purpose: To simulate household spending on luxury goods if budget allows after necessities.

Inputs:

- `remaining_budget`: Funds available after necessity spending.

- `percentage_range`: A tuple (min_percent, max_percent) of the `remaining_budget` to allocate to luxuries.

Process:

1. If `remaining_budget` <= 0, returns 0
2. Determine `total_luxury_budget_to_spend` by choosing a random percentage from `percentage_range` and applying it to `remaining_budget`
3. Randomly sample 2 luxury firm areas (e.g., "technical", "creative").
4. Divide `total_luxury_budget_to_spend` equally among the chosen luxury types
5. For each chosen luxury type:
    a. Identify potential firms of that type with positive price and inventory, and shuffle them.
    b. Iteratively attempt to purchase from these firms (random choice from those still having inventory) until the budget for this luxury type is exhausted or no more firms are available. Purchase logic is similar to `_calculate_cost_and_buy` but selection is always random from available firms

Returns: The total amount actually spent on luxury goods.

5. **Household Step (`step`)**

Purpose: To execute the household's complete economic cycle for one simulation step. This is the main submodel for households.

Process:

1. **Income Calculation:** Sum `wage` from all employed `self.members` to set `self.household_step_income`
2. **Determine Income Bracket:** Set `self.income_bracket` ("low", "middle", "high") based on `self.household_step_income` relative to `total_necessity_target` (calculated as `self.necessity_spend_per_person * self.num_people`)
3. **Calculate Post-Tax Income:** `self.household_step_income_posttax = self.household_step_income * (1 - self.income_tax_rate)` (Note: `income_tax_rate` is set by the government based on the bracket)
4. **Determine Wealth Bracket:** Set `self.wealth_bracket` ("low", "middle", "high") based on `self.total_household_savings +`

`self.household_step_income_posttax` relative to `total_necessity_target`

5. **Necessity Spending:**
    - Calculate `available_funds` (post-tax income + savings)
    - Calculate `attemptable_necessity_budget` as `min(total_necessity_target, available_funds)`
    - Attempt to spend half of this on "physical" goods using `_calculate_cost_and_buy`
    - Attempt to spend the remainder on "service" goods using `_calculate_cost_and_buy`
    - Calculate `total_necessity_spent`
6. **Luxury Spending:**
    - Calculate `remaining_funds` after necessity spending.
    - If `self.wealth_bracket` is "middle" or "high" and `remaining_funds` > 0, call `_spend_on_luxuries` with appropriate percentage ranges (0.6-1.0 for middle, 0.8-1.0 for high).
7. **Update Financial Metrics:**
    - `self.household_step_expense` = `total_necessity_spent` + `luxury_spent`.
    - `self.total_household_savings` = `available_funds` - `self.household_step_expense`.
    - Update `self.debt_level`.
8. **Update Health and Welfare:**
    - Calculate `necessity_fulfillment` percentage. If < 1.0, increment `self.model.unmet_necessity_households_count`.
    - Set `base_health` based on `self.wealth_bracket`.
    - `self.health_level` = `base_health` * `necessity_fulfillment`.
    - `self.welfare` is calculated as a weighted sum of `household_step_income_posttax`, `household_step_expense`, `total_household_savings`, and `health_level`.

Outputs/State Variables Updated: All household financial, bracket, health, and welfare variables. `model.unmet_necessity_households_count` is also potentially updated.

Person Agent Submodels

1. **Skill and Job Seeking Update (`step`)**

   Purpose: To manage a person's job-seeking status and skill development.

   Inputs:

   - `self.job_seeking`: Current job-seeking status.
   - `self.studying_for_min_skills`: Flag indicating if studying.
   - `self.skill_level`: Current skill level.
   - `self.skill_type`: The person's skill area.
   - `self.skill_improvement_rate`: Base rate of skill improvement.
   - `self.employer`: Current employer (if any).
   - (Implicitly) `FirmAgent.min_skill_levels_config`: Sensed from any firm in the model to determine minimum entry skill for their type.

   Process:

   1. Determine `min_entry_level` for the person's `skill_type` by checking the `min_skill_levels_config` of a *Firm*
   2. If the *Person* is currently `studying_for_min_skills` OR if they are `job_seeking` but their `skill_level` is below `min_entry_level`:
      a. Set `self.studying_for_min_skills = True`.
      b. Set `self.job_seeking = False`
      c. Increase `self.skill_level` by an enhanced rate (`self.skill_improvement_rate * 1.5`), capped at 100
      d. If `self.skill_level` now meets or exceeds `min_entry_level`, set `self.studying_for_min_skills = False` and `self.job_seeking = True`
   3. If the person has an `employer`, ensure `self.job_seeking` is `False` and `self.studying_for_min_skills` is `False`

   Outputs/State Variables Updated:

   - `self.job_seeking`
   - `self.studying_for_min_skills`
   - `self.skill_level`

Government Agent Submodels

1. **Inflation Calculation (`_calculate_inflation_rate`)**

   Purpose: To calculate the economy-wide inflation rate based on firm price changes.

   Inputs:

   - *self.model.current_step*: Current simulation step.
   - Price data (*product_price*, *price_two_steps_ago*) from all FirmAgents.
   - *firm_type* ("necessity" or "luxury") of FirmAgents.

   Process:

   1. Use a fixed inflation rate for an initial *BURN_IN_PERIOD* (5 steps)
   2. After burn-in, collect price changes ((*current_price - price_two_steps_ago*) / *price_two_steps_ago*) for all firms with valid historical price data
   3. Separate price changes for "necessity" and "luxury" firms
   4. Calculate average price change for necessity goods and luxury goods
   5. Calculate the overall *self.inflation_rate* as a weighted average (80% necessity, 20% luxury). Handle cases where one or both categories have no price changes

   Outputs/State Variables Updated:

   - *self.inflation_rate*

2. **Tax Collection (`_collect_taxes`, `_collect_corporate_taxes`)**
   - **`_collect_taxes` (Household Income Tax):**

   Purpose: To collect income tax from all HouseholdAgents.

   Inputs:

   - *household_step_income* and *income_bracket* from each HouseholdAgent;
   - *self.tax_rates* (dictionary of rates per bracket).

   Process:

   1. Iterate through *Households*

2. Determine the applicable `tax_rate` based on the *Household's* `income_bracket`
3. Calculate `tax_amount`
4. Add `tax_amount` to `self.step_tax_revenue`
5. Update the household's `income_tax_rate` attribute

Outputs/State Variables Updated:

- `self.step_tax_revenue`. (Indirectly updates `HouseholdAgent.income_tax_rate`).

- **`_collect_corporate_taxes`:**

  Purpose: To aggregate corporate taxes paid by `FirmAgent`s.

  Inputs:

  - `tax_paid_this_step` attribute from each `FirmAgent`.

  Process:

  1. Iterate through all `FirmAgent`s
  2. Sum *Firm* `tax_paid_this_step` (which firms calculated in their own step based on profits and `self.corporate_tax_rate`)

  Outputs/State Variables Updated:

  - `self.step_corporate_tax_revenue`.

3. **Welfare Distribution (`_calculate_and_distribute_unemployment_payments`, `_calculate_and_distribute_low_income_transfers`)**

- **`_calculate_and_distribute_unemployment_payments`:**

  Purpose: To provide unemployment benefits to eligible `PersonAgent`s.

  Inputs:

  - Employment status (`employer`)
  - `job_seeking` status of all `PersonAgent`s.

  Process:

1. Identify unemployed *Persons* (`employer is None` and `job_seeking is True`)
2. Pay each unemployed *Person* a fixed `payment_per_person` (10,000), which is set as their `wage` for the step
3. Accumulate `total_unemployment_payments`

Outputs/State Variables Updated:

- `PersonAgent.wage` for unemployed individuals.

Returns: `total_unemployment_payments`.

- **`_calculate_and_distribute_low_income_transfers`:**

  Purpose: To provide financial assistance to low-income households.

  Inputs:

  - `household_step_income_posttax`
  - `total_household_savings`
  - `necessity_spend_per_person`
  - `num_people`

  from each `HouseholdAgent`.

  Process:

  1. For each household, calculate `total_necessity_target`
  2. If `available_funds` (income + savings) is less than this target, calculate the `deficit` and provide a `transfer_amount` (`deficit` + 5,000 buffer) directly to `household.total_household_savings`
  3. Accumulate `total_low_income_transfers`

  Outputs/State Variables Updated:

  - `HouseholdAgent.total_household_savings` for eligible households.

  Returns: `total_low_income_transfers`.

4. **Government Spending (`_execute_government_necessity_spending`)**

   Purpose: To simulate government purchasing necessity goods from firms.

   Inputs:

   - `budget`: The amount allocated for this spending.
   - List of "necessity" `FirmAgent`s, their `product_price` and `inventory`.

   Process:

   1. Split the `budget` equally between "physical" and "service" necessity categories.
   2. For each category:
      a. Identify potential firms (necessity type, matching area, positive price and inventory). Shuffle these firms
      b. Iteratively attempt to purchase from these firms:
         i. Calculate `desired_units` based on remaining category budget and firm price
         ii. Call `chosen_firm.fulfill_demand_request(desired_units)`
         iii. If units are bought, update `total_spent_on_necessities`, reduce remaining category budget, and record the purchase amount in `self.government_purchases_from_firms_step`

   Outputs/State Variables Updated:

   - `self.government_purchases_from_firms_step`
   - (Indirectly) `FirmAgent.inventory` and sales metrics.

   Returns: `total_spent_on_necessities`

5. **Economic Indicator Calculation (`_calculate_unemployment_rate`, `_calculate_gdp`, `calculate_gini_coefficient`)**

   - **`_calculate_unemployment_rate`:**

     Purpose: To calculate the overall unemployment rate.

     Inputs:

     - Employment status of all `PersonAgent`s

- `job_seeking` status of all `PersonAgent`s

Process:

1. Define labor force (employed + job-seeking unemployed)
2. Calculate `unemployment_rate` as (number of actively unemployed / labor force size) * 100

Outputs/State Variables Updated:

- `self.unemployment_rate`

- **`_calculate_gdp`**:

  Purpose: To calculate the Gross Domestic Product for the step.

  Inputs:

  - `produced_units` from all `FirmAgent`s
  - `product_price` from all `FirmAgent`s

  Process:

  1. Sum the value of production (`produced_units * product_price`) across all firms

  Outputs/State Variables Updated:

  - `self.GDP`

- **`calculate_gini_coefficient`**:

  Purpose: To measure income inequality.

  Inputs:

  - `wage` of all `PersonAgent`s.

  Process:

  1. Collect all *Person* wages (using `max(0, wage))`
  2. Sort *Person* wages
  3. Apply the standard Gini coefficient formula

  Outputs/State Variables Updated:

- `Self.gini_coefficient`

6. **Government Step (`step`)**

    Purpose: To execute the government's complete fiscal and monitoring cycle for one simulation step. This is the main submodel for the government.

    Process (Order of Operations):

    1. Reset `self.government_purchases_from_firms_step`
    2. Call `_calculate_inflation_rate()`
    3. Calculate and distribute welfare:
        a. `unemployment_payments_total = _calculate_and_distribute_unemployment_payments()`
        b. `low_income_transfers_total = _calculate_and_distribute_low_income_transfers()`
    4. Set `self.step_public_spending` to the sum of these welfare payments
    5. Decrease `self.reserves` by this initial `step_public_spending`
    6. If not the first step, allocate 10% of *remaining* `self.reserves` as a budget for `_execute_government_necessity_spending()`
    7. Decrease `self.reserves` by the amount spent on necessity goods and add this amount to `self.step_public_spending`
    8. Collect taxes:
        a. `self.step_tax_revenue = _collect_taxes()` (household income tax)
        b. `self.step_corporate_tax_revenue = _collect_corporate_taxes()`
    9. Increase `self.reserves` by total tax revenue
    10. Calculate other economic indicators: `_calculate_unemployment_rate()`, `self.GDP = _calculate_gdp()`, `self.gini_coefficient = calculate_gini_coefficient()`
    11. Update `self.previous_reserves`

    Outputs/State Variables Updated:

    - All government financial and economic indicator state variables.
    - (Indirectly) States of `HouseholdAgent`s, `PersonAgent`s, and `FirmAgent`s through transfers, taxes, and purchases.

Intermediary Firm Agent Submodels

1. **Initial Workforce Population (`_populate_initial_workforce`)**

   Purpose: To hire the initial set of employees for the intermediary firm.

   Inputs:

   - *`target_total_employees`*: Desired total number of initial employees.
   - *`self.model.available_persons`*: Global list of `PersonAgent`s.
   - *`self.skill_types_to_hire`*: List of all skill types the firm aims to hire.

   Process:

   1. Calculate the number of employees to hire per skill type to achieve a balanced workforce
   2. For each skill type in *`self.skill_types_to_hire`*:
      a. Filter *`self.model.available_persons`* for candidates who are *`job_seeking`*, have no *`employer`*, and match the current *`skill_type`*
      b. Randomly shuffle these candidates
      c. Hire candidates until the target for that skill type is met or *`target_total_employees`* is reached

   Outputs/State Variables Updated:

   - *`self.employees`*: List of `PersonAgent`s employed
   - *`self.num_employees`*: Updated count
   - For hired `PersonAgent`s: *`employer = self`*, *`job_seeking = False`*, *`job_level = "entry"`*, *`wage = 0`* (wage is set dynamically in the **`step`** method)

2. **Receive Firm Demand (`receive_firm_demand`)**

   Purpose: To accumulate the costs (representing demand for its services/inputs) submitted by other `FirmAgent`s.

   Inputs:

   - *`cost`*: The amount of production cost received from a `FirmAgent`.

   Process:

   1. Increments *`self.demand_received_from_firms`* by *`cost`*

Outputs/State Variables Updated:

- `self.demand_received_from_firms`

3. **Intermediary Firm Step (`step`)**

Purpose: To process accumulated demand, set revenue, and distribute wages to its employees. This is the main submodel for the intermediary firm.

Process:

1. Set `self.revenue = self.demand_received_from_firms`
2. If `self.num_employees` > 0, calculate `wage_per_employee` as `self.revenue / self.num_employees`. Otherwise, `wage_per_employee` is 0
3. Iterate through `self.employees` and set each `PersonAgent.wage` to `wage_per_employee`
4. Reset `self.demand_received_from_firms` to 0 for the next step

Outputs/State Variables Updated:

- `self.revenue`
- `PersonAgent.wage` for its employees.
- `self.demand_received_from_firms` (reset).

3. RESULTS

As mentioned in the Initialization section, the biggest problem with this model is its initial values. Due to the time limit on this project, the values are not fully calibrated to a level where everything “works”. While some results are generally the same for every initial value set and are representative of how the system works (demand, production levels), some results are purely dependent on the initial value set (markup levels, profit levels), this section will only demonstrate the former. For all results, the first 10 steps’ results should be ignored as stated in the Initialization section. It should be noted that each graph shown below is derived from the exact same simulation run.

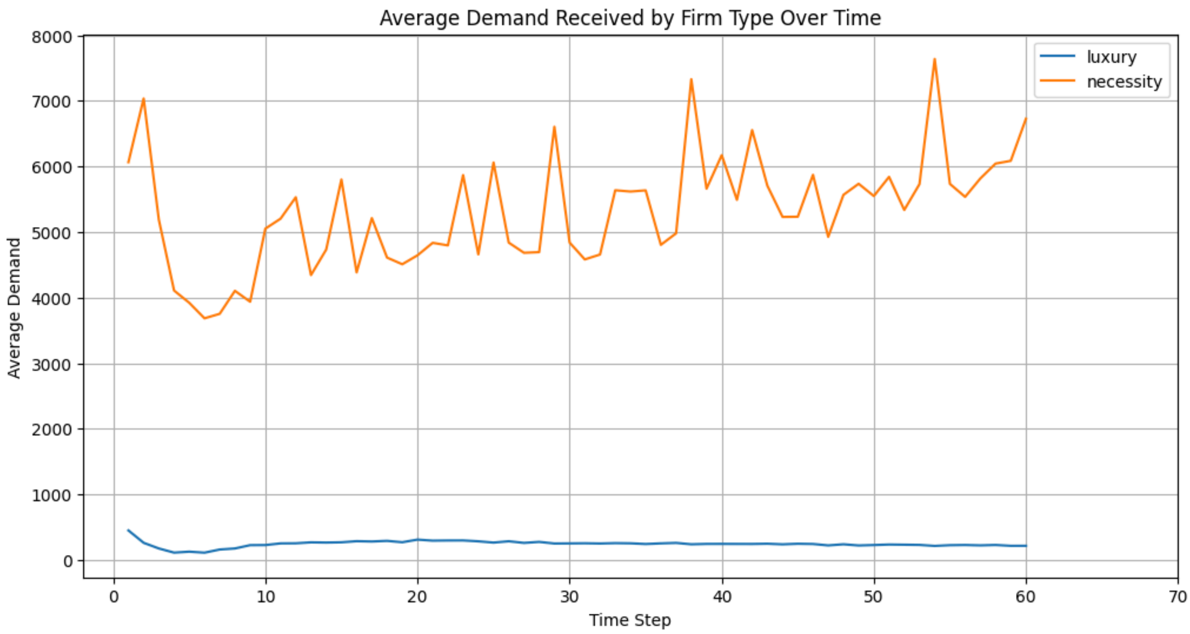

Figure 3.1: Demand received by Firms over time

Figure 3.1 shows the average demand *Firms* receive over time. The graph shows that “necessity” items are demanded more than “luxury” items, which is a realistic pattern.

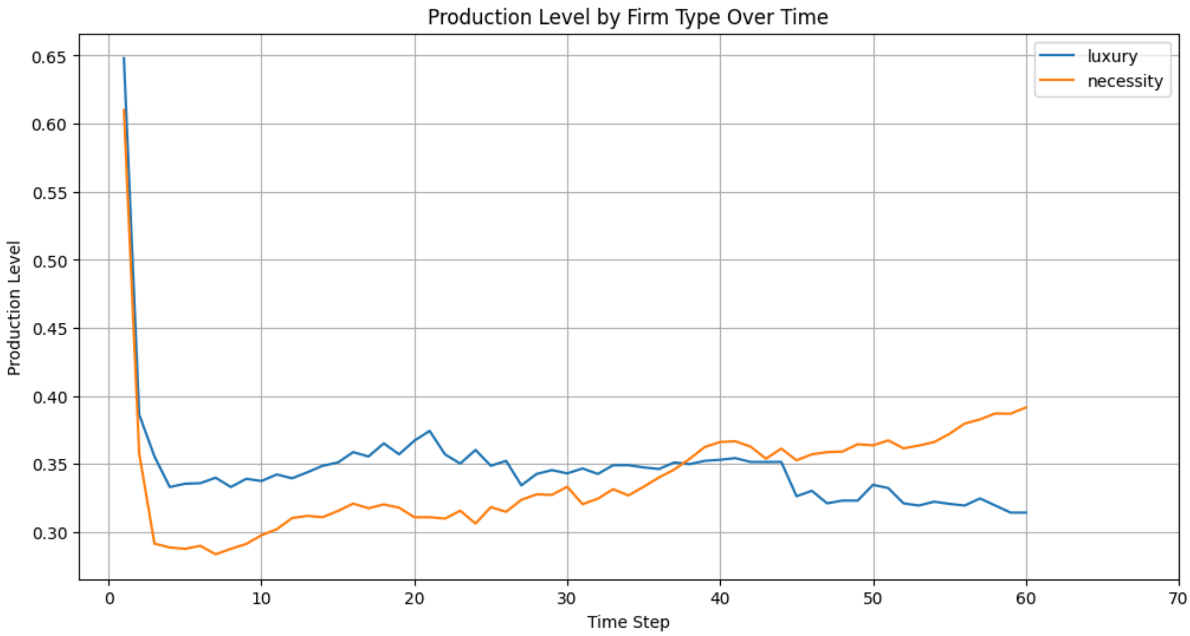

Figure 3.2: Production level by Firm type over time

In correlation with *Figure 3.1* graph, as the average demand rises, production levels also rise for "necessity" items. This is an intended feature of the model, and represents how *Firms* react to demand from their consumers.

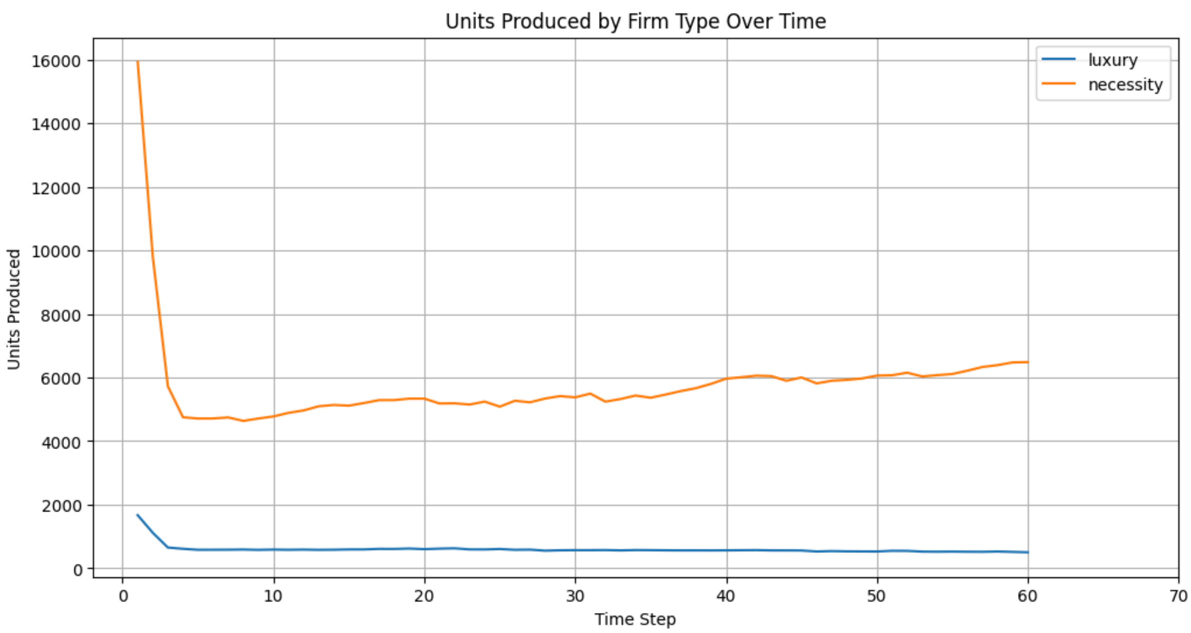

Figure 3.3: Units produced by Firm type over time

As expected, *Figure 3.3* shows that produced units go up for "necessity" items and down for "luxury" items. This directly correlates with *Figure 3.2*. It should be noted that the actual amount of units produced is explicitly determined as an initial value, meaning the difference in unit count is not determined by the simulation but the simulation runner.

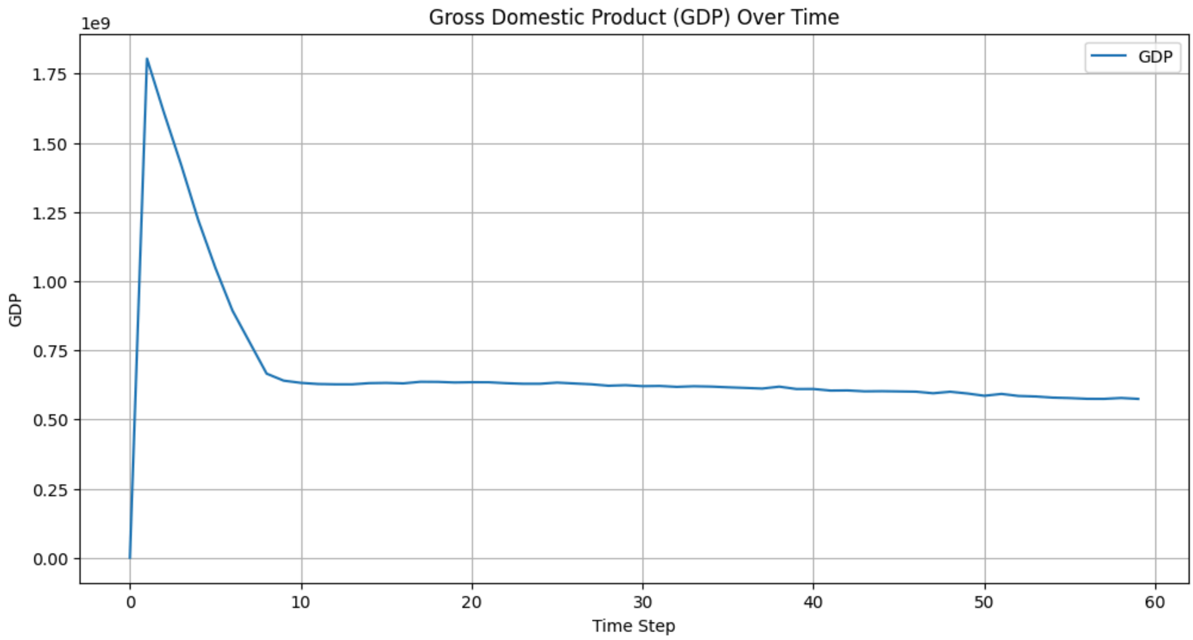

Figure 3.4: GDP over time

The model calculates GDP as total production from all *Firms* in a single step. While the GDP value might not suit real-world examples, it directly correlates with the production graphs.

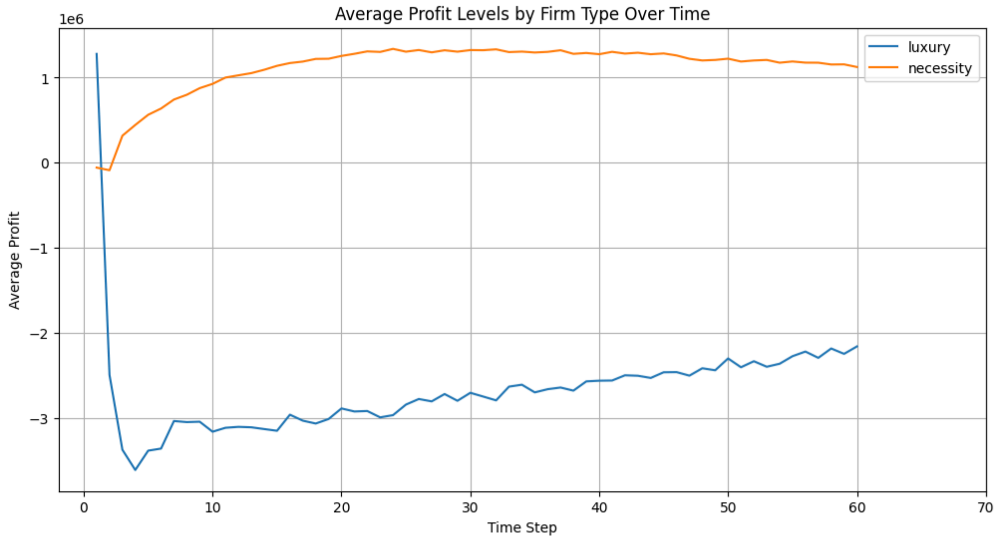

Figure 3.5: Average profit of Firms by Firm type over time

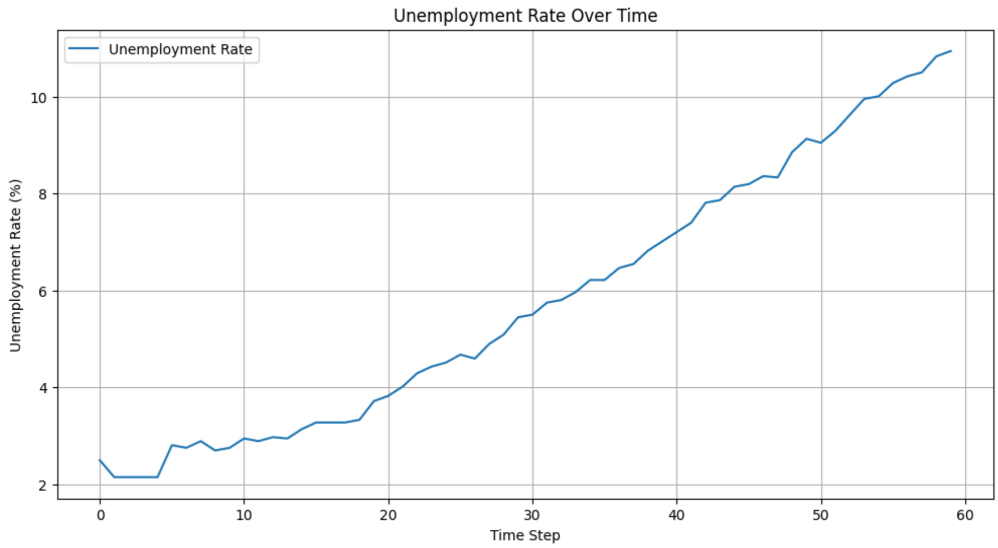

Figure 3.6: Unemployment rate over time

Due to how the initial values were given, “luxury” *Firms* are at a loss while “necessity” Firms are profiting. The model is designed so that *Firms* at a loss fire their employees to get out of a slump. As seen in *Figure 3.6*, the unemployment rate constantly goes up due to “luxury” *Firms’* firing cycles.

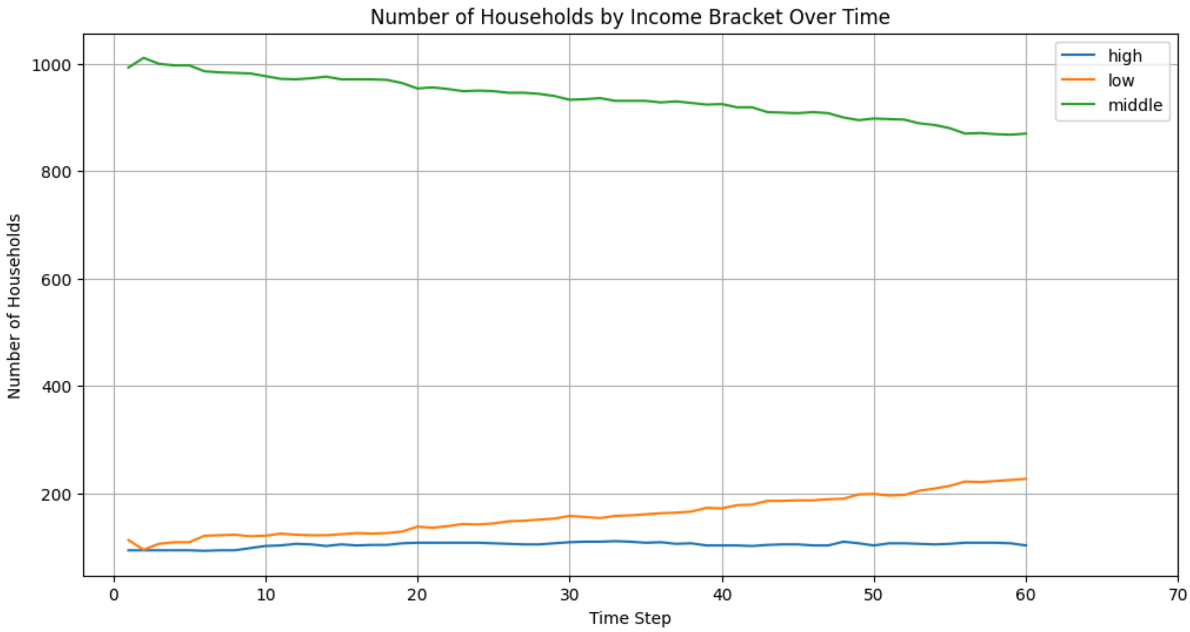

Figure 3.7: Number of Households by income bracket over time

Because of the firings, the distribution of income brackets naturally change. Less employed people means less income, which means more people in the lower income brackets. As seen in *Figure 3.7*, “low” income bracket *Households* rise as the simulation comes to an end.

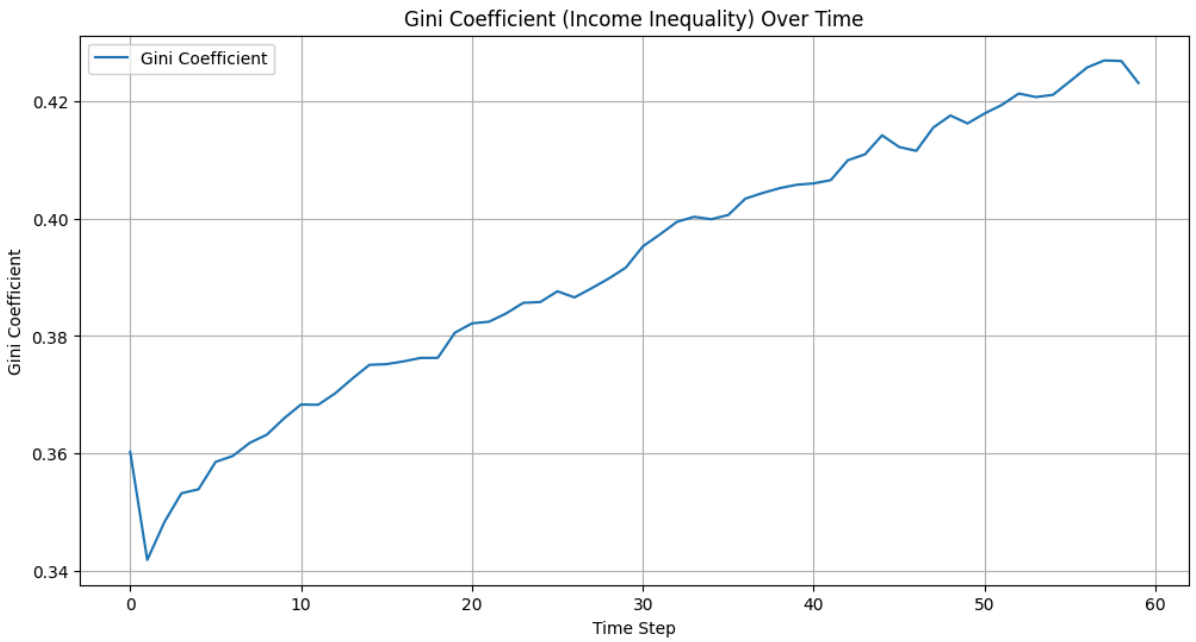

Figure 3.8: Gini Coefficient over time

The change in income brackets also affects the Gini score of the simulation, where inequality in the system increases.

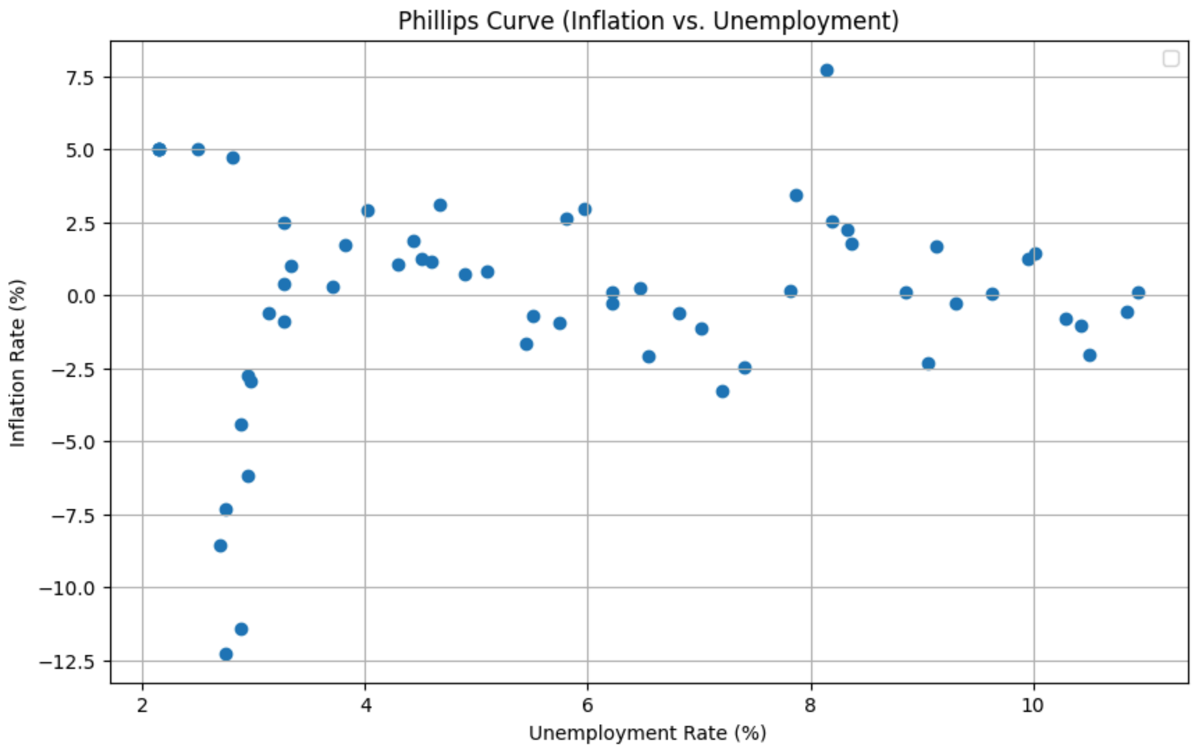

Figure 3.9: Phillips Curve

The Phillips Curve definition states that as inflation increases, unemployment drops. In this simulation run, even with distorted inflation rates, the Phillips Curve doesn't seem to exist.

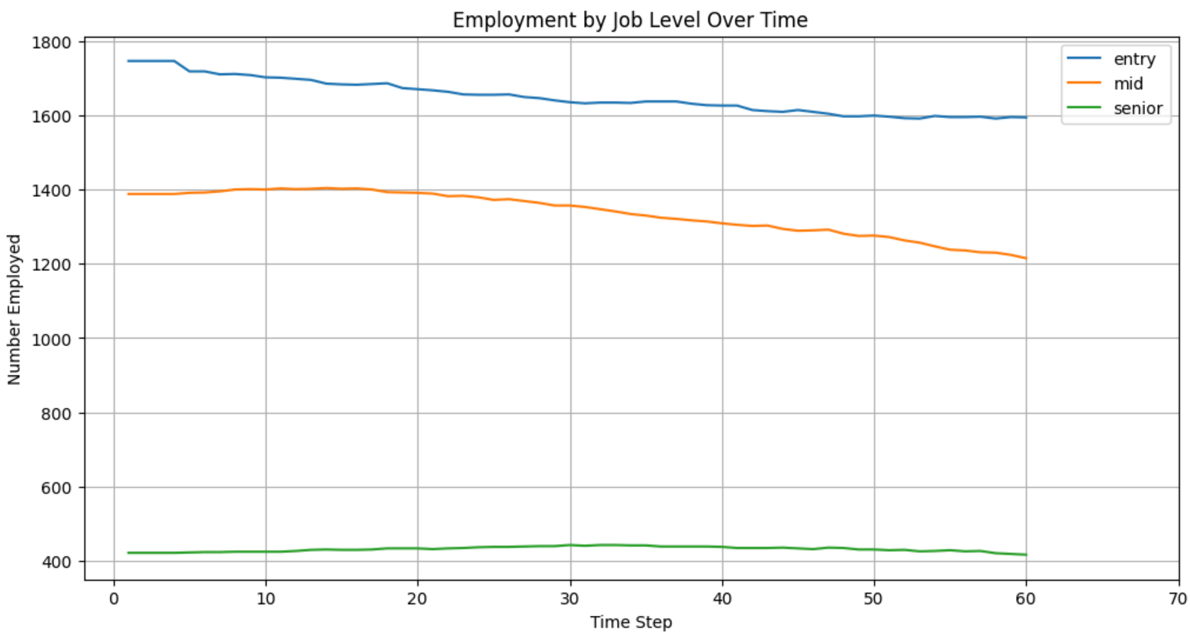

Figure 3.10: Employment number by job level over time

Figure 3.10 demonstrates the way the skill distribution across job levels works in the model. Higher-skilled *Persons* are naturally less in the entire system, and lower-skilled *Persons* are more, thus the amount of entry level job positions are the most in the labor market.

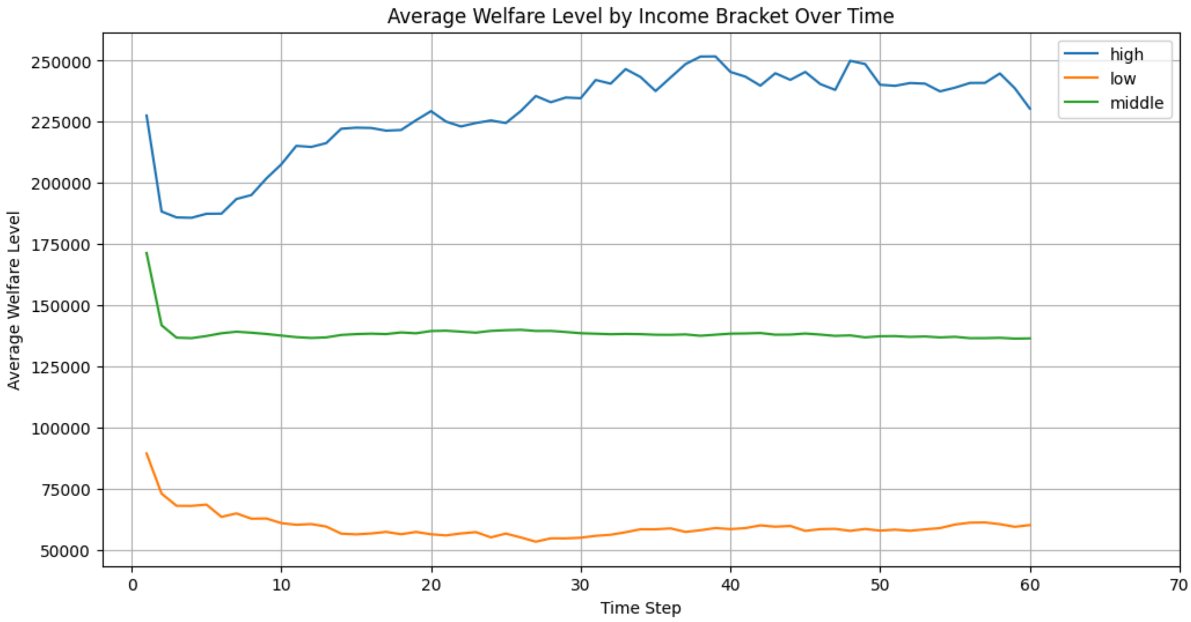

Figure 3.11: Average welfare by income bracket over time

As mentioned in the Submodels section, welfare is calculated as `self.household_step_income_posttax`*0.3 + `self.household_step_expense`*0.2 + `self.total_household_savings`*0.2 + `self.health_level`*0.3

Naturally, higher income *Households* have the highest welfare score, where low income *Households* have the lowest welfare score.

4. CONCLUSION

This report has presented a comprehensive description of Economixim, an agent-based model designed to simulate a national economy, adhering to the ODD (Overview, Design Concepts, Details) protocol. The primary goal of this project was to design a virtual laboratory to understand economic systems, explore the complex interactions that occur between micro-level players in an economy, and monitor the emergent behaviors that are generated through these interactions.

As the results indicate, the system-level implementation and the logic behind the system works as intended and creates meaningful and surprising results. However, it is not without its problems. As mentioned in the Initialization chapter, the most difficult part of models such as this one is what values the simulation starts off with and how it is calibrated. The model is structured such that simulations are anticipated to demonstrate coherent dynamic patterns (such as convergence or cyclical behavior), though the nature of these numerical outcomes will be highly sensitive to initial parameterization. Real-world economies are not computational values, they are so-called "free" markets that generate their values intrinsically, over a long period of time, which means finding the "perfect" values also require some sort of natural process. Learning mechanisms such as RL are a good solution to this problem. Values can be tried and tested thousands of times to find a good balance that researchers will be content with.

Another limitation of this project is in its granularity, which is somewhat paradoxical: it offers more granularity than existing models thanks to the attributes each agent has and the existence of a complex Person agent, yet still falls short of what's needed to accurately simulate the real world. For example, as discussed in the Purpose section, the model currently does not simulate demographic occurrences such as death, birth, and marriage. The model also lacks a supply chain system, a mechanic where Firms can produce new product lines, and a banking system; things necessary for a "realistic" economy simulation. These limitations naturally point towards several promising routes for improvement. Enhancing this model would require the implementation of such features and better calibration than its current state. For example, developing a banking sector would allow for the exploration of credit cycles and financial stability, integrating international trade would open the model to global economic influences, and so on.

Economixim serves as a decent starting point for economic ABMs aiming for advanced granularity. With further development, Economixim has the potential to grow into a more robust tool for both theoretical and qualitative exploration of economic policy impacts and the understanding of human decision-making.

REFERENCES

Cited

- Dawid, H., & Delli Gatti, D. (2018). Agent-based macroeconomics. In C. Hommes & B. LeBaron (Eds.), *Handbook of Computational Economics* (Vol. 4, pp. 63–156). Elsevier.
- Heckbert, S., Baynes, T., & Reeson, A. (2010). Agent-based modeling in ecological economics. *Annals of the New York Academy of Sciences, 1185*(1), 39–53.
- Macal, C. M., & North, M. J. (2009). Agent-based modeling and simulation. *Proceedings of the 2009 Winter Simulation Conference (WSC)* (pp. 86–98).
- Tesfatsion, L. (2006). Agent-based computational economics: A constructive approach to economic theory. In L. Tesfatsion & K. L. Judd (Eds.), *Handbook of computational economics* (Vol. 2, pp. 831–880). Elsevier.
- Farmer, J. D., & Foley, D. (2009). The economy needs agent-based modelling. *Nature, 460*(7256), 685–686.
- Schelling, T. C. (1971). Dynamic models of segregation. *Journal of Mathematical Sociology, 1*(2), 143–186.
- Epstein, J. M., & Axtell, R. (1996). *Growing artificial societies: Social science from the bottom up*. Brookings Institution Press.
- Glielmo, A., Devetak, M., Meligrana, A., & Poledna, S. (2025). *BeforeIT.jl: High-Performance Agent-Based Macroeconomics Made Easy*. arXiv preprint arXiv:2502.13267.
- Delli Gatti, D., Fagiolo, G., Gallegati, M., Richiardi, M., & Russo, A. (2018). *Agent-based models in economics: A toolkit*. Cambridge University Press.
- Assenza, T., Delli Gatti, D., & Grazzini, J. (2015). Emergent dynamics of a macroeconomic agent based model with capital and credit. *Journal of Economic Dynamics and Control, 50*, 5–28.
- Osoba, O. A., Vardavas, R., Grana, J., Zutshi, R., & Jaycocks, A. (2020). *Policy-focused agent-based modeling using RL behavioral models*. arXiv preprint arXiv:2006.05048.
- Brusatin, S., Padoan, T., Coletta, A., Delli Gatti, D., & Glielmo, A. (2024). Simulating the Economic Impact of Rationality through Reinforcement Learning and Agent-Based Modelling. *Proceedings of the 5th ACM International Conference on AI in Finance (ICAIF '24)* (pp. 159–167). Association for Computing Machinery.
- Zheng, S., Trott, A., Srinivasa, S., Naik, N., Gruesbeck, M., Parkes, D. C., & Socher, R. (2020). *The AI Economist: Improving Equality and Productivity with AI-Driven Tax Policies*. arXiv preprint arXiv:2004.13332.
- Dwarakanath, K., Vyetrenko, S., Tavallali, P., & Balch, T. (2024). *ABIDES-Economist: Agent-Based Simulation of Economic Systems with Learning Agents*. arXiv preprint arXiv:2402.09563.
- Salle, I., Yıldızoğlu, M., & Sénégas, M.-A. (2013). Inflation targeting in a learning economy: An ABM perspective. *Economic Modelling, 34*, 114–128.
- Yang, Y., Zhang, Y., Wu, M., Zhang, K., Zhang, Y., Yu, H., Hu, Y., & Wang, B. (2025). *TwinMarket: A Scalable Behavioral and Social Simulation for Financial Markets*. arXiv preprint arXiv:2502.01506.

- Grimm, V., Railsback, S. F., Vincenot, C. E., Berger, U., Gallagher, C., DeAngelis, D. L., Edmonds, B., Ge, J., Giske, J., Groeneveld, J., Johnston, A. S. A., Milles, A., Nabe-Nielsen, J., Polhill, J. G., Radchuk, V., Rohwäder, M.-S., Stillman, R. A., Thiele, J. C., & Ayllón, D. (2020). The ODD protocol for describing agent-based and other simulation models: A second update to improve clarity, replication, and structural realism. *Journal of Artificial Societies and Social Simulation, 23*(2), 7.

Uncited

- North, M. J., Collier, N. T., & Vos, J. R. (2006). Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. ACM Transactions on Modeling and Computer Simulation, 16(1), 1–25.
- ter Hoeven, E., Kwakkel, J., Hess, V., Pike, T., Wang, B., rht, & Kazil, J. (2025). Mesa 3: Agent-based modeling with Python in 2025. Journal of Open Source Software, 10(107), 7668. https://doi.org/10.21105/joss.07668
- Wilensky, U. (1999). *NetLogo*. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.
- Arthur, W. B. (2021). Foundations of complexity economics. Nature Reviews Physics, 3(2), 136–145.
- Holland, J. H., & Miller, J. H. (1991). Artificial Adaptive Agents in Economic Theory. *The American Economic Review, 81*(2), 365–370.
- Jang, I., Lee, D., Kim, D., & Son, Y. (2018). An Agent-Based Simulation Modeling with Deep Reinforcement Learning for Smart Traffic Signal Control. *Proceedings of the 2018 International Conference on Information and Communication Technology Convergence (ICTC)* (pp. 1028-1030). IEEE.
- Lalmohammed, S. (2025). *Welfare modeling with AI as economic agents: A game-theoretic and behavioral approach*. arXiv preprint arXiv:2501.15317.
- Pangallo, M., & del Rio-Chanona, R. M. (2024). *Data-Driven Economic Agent-Based Models*. arXiv preprint arXiv:2412.16591.
- Poledna, S., Miess, M. G., Hommes, C., & Rabitsch, K. (2023). Economic forecasting with an agent-based model. *European Economic Review, 151*, 104306.
- Stiglitz, J. E. (2018). Where modern macroeconomics went wrong. *Oxford Review of Economic Policy, 34*(1-2), 70–106.
- Vargas-Pérez, V. A., Mesejo, P., Chica, M., & Cordón, O. (2022). Deep reinforcement learning in agent-based simulations for optimal media planning. *Information Fusion, 87*, 1-18.