# Software Report:

The implementation was done using TypeScript

Using an object orientated structure, one file, house.ts contains the discrete event simulation that run on an engine in the file housing-engine.ts. This engine is a transcribed version of the airplane sim that listed in the resources of this class.

All parameters are set at the top of house.ts in the **params** object, and should be modified there. Notice the structure of the config object:

```
const params = {
  commissionRate: .03,
  travelCost: 100,
  FixedSaleCost: 500,
  numBuyers: 100,
  numSellers: 10,
  runtime: 10000,
}
```

A Buyer is a class that has a randomly assigned buying price for which he is looking to buy a property. Upon being requested to browse listings, he will determine which properties to visit based on the price range. There is an incurred cost of travel and time, so not every property can be visited. This variable is handled by the parameter **TravelCost**. When a Buyer is created using the new Buyer syntax, certain properties are automatically populated into the class, as can be seen here:

```
class Buyer {
  constructor(
    public bias:number=genBias(),
    public budgetPrice:number=genBudgetPrice(),
    public name:string=genName(),
    public time:number=0
  ){
    console.log(`Buyer  ${this.name} has entered the market!`)

  }
  ...
```

these properties are set using "gen-erator" functions, which are global in scope, and mostly generate random values to make unique Buyers and Sellers. See genName here:

```
function genName():string{
  let firsts = ['Mike','Sam','Bill','John','Ben','Emanuel','Chris','Stacey']
  let surs =
['Smith','Wells','Orms','Brown','Banks','Davis','Barns','Davey','Harvey','Oswald','Rice-
aroni','Sanders']
  let name = firsts[Math.floor(Math.random()*firsts.length)] + " " +
  surs[Math.floor(Math.random()*surs.length)]
  return name
}
```

The queues for the sellers is a singleton named SellerQueue and contains code for scheduling events to spawn sellers into the queue, and thus set the listings in the market.

The BuyerQueue is a singleton that similarly does this for buyers, however, it also schedules events for telling buyers to start browsing home listings to potentially visit.

The simulation begins by the Buyer and Seller queues spawning the objects, and it is ended when the engine's simulation time surpasses that of the **param.runtime** parameter.

Probably the most feature of using TypeScript is the ability to throw around functions like they're variables. I utilized this to simplify my engine. No longer does my Engine Event include a data property, simply a timestamp and a callback. All my Engine has to do is pick the next scheduled event and fire it. This also helps maintain references and scope on the variables that I schedule events on. Check it out:

```
console.log(this.name + ' visited property')
    engine.schedule(engine.currentTime()+5,()=>{

      let sold : number
      let diff = seller.postingPrice - this.budgetPrice
      let sellerLeeway = seller.bias*1000
      let buyerLeeway = this.bias*1000
      if (diff < 0) { //if buyer gets a good deal
        if (Math.abs(diff) > sellerLeeway) {
          engine.schedule(engine.currentTime()+5, ()=>{ //event scheduling-CEPTION
            seller.failedSale()
            this.failedBuy()
          })
        }
```

Notice how this enables the nesting of scheduled events, without having to define more functions and pass variables around! Maintain the scope, and nest events at the same time!

Initial Experiments: There were a few experiments I did with the simulation, which involved tweaking the parameters and runtime of the discrete event simulation (in the form of a max time stamp). These consisted of the following:

- Changing the base biases of the Buyers and Sellers: this made the buyers more willing to pay more for a property and thus less likely to fail at a sale. When changing the initial biases, which originally were a simple random distribution from 0-1 that would multiply a $1000 base allowance for compromises.
Decreasing this value resulted in fewer failed purchaes overall, as both parties were able to compromise to a greater extent
- changing the max timescale of the sim: I noticed that changing the max timestamp of the sim had diminishing returns on the amount of sales that could become processed with a fixed number of buyers and sellers. However, it was more likely that all listings were purchased with a longer timescale
- With increased paperwork costs: Houses were more likey to fail, yeilding a much larger "failedPurchases" number.

# Project 1 Proposal - Urban Housing Markets 🏠

*Housing markets are complex, and a discrete event simulation lends itself to modeling how different forces and individuals' decisions affect the market as a whole.*

## Project Description

**SUI (System Under Investigation)** : Housing Market

### The Problem

How can one model the development of the housing market and the time to buy/sell for the average person in the market?

**Possible Insights**:

- How does changing the commission rates of real estate agents affect the amount of home improvements?
- How do people's willingness to negotiate affect the time a house could be on market?
- Would prices trend a certain way if buyers are more likely to compromise over sellers?
- How many properties could get sold in a given time frame?

- Does the paperwork cost affect a home sale?

**Goals**:

- model the process of the housing market by simulating potential sales and listings of properties
- using a discrete event simulation, NOT time-stepped, in order to focus on how events themselves affect variables in the simulation
- include phenomena common to housing markets such as:
    - housing purchases / sales
    - renovations
    - buying/holding
    - housing market listings
- Give insight into the time frame of a property

# Conceptual Model

The model is based on the model provided by An Empirical Stationary Equilibrium Search - Model of the Housing Market. I have adapted it to fit the ABCmod implementation

## Overview

The housing market is modeled as a **resource** that can take a **Person** who can be a **buyer** or a **seller** and connect them for a potential sale. The market pulls from **queues** of the buyers and sellers and calculates the sales of homes.
**Sellers** and **buyers** have distinct predispositions to favor selling or buying quickly, modeled with attribute **bias**. These are randomly generated per person.
A home's value is determined by its attribute **features** which is a set of **resources** having a predefined utility or value, as well as the **land** value that it sits on.

**What's Modeled**:

- Housing purchases / sales
- Liklihood of sale of a home
- Value of homes as a product of land price and features (furnishings)

**What's Abstracted or Ignored**:

- **Crime rates, local schools, local attractions**: we assume this is baked into house prices already. Accounting for this would introduce more explicit complexity when it can be abstracted into housing values.
- **Bus routes, proximity to publix transit**: we assume proximity to public transportation is baked into housing values. Accounting for this could make the simulation too area specific, where some urban areas have more value in public transportation, or traffic.
- **Stability of housing market**: we allow for a parameter to control the overall trend of the market by affecting demand, and the values of the houses are determined by the demand, so simulating a recession or crash is solely determined by that parameter. No foreclosures are modeled to prevent needing to account for home loans.
- **Distribution of houses**: we assume a very simple distribution of houses, where they are all equidistant from each other. This is to allow the model to focus on the system of supply/demand and parameters rather than area specifics. All houses are modeled to have the same capacity of 1 for simplicity.
- **Value of a house**: The value of a house is assumed to be rapidly increasing if being renovated, slightly increasing if inhabited by owner, slightly decreasing if inhabited by renter, and rapidly decreasing if inhabited by a squatter. It is neutral if empty but has a chance to become sqautted on (rented for $0).

## ABCmod Implementation

*see Appendix for ABCmod entity and activity tables*

**Overview of Entities**:

- **Seller** (Consumer) - a seller in the market, has attributes **postingPrice**: the price listing for sale, **bias**: the bias towards a faster sale (getting less)
- **Buyer** (Consumer) - a buyer in the market. Has attributes **budgetPrice**: the preferred max budget, **bias** the bias

towards a faster sale (paying more). Has actions **visitHome**-visit a home with fixed cost **TravelCost**

- **Market** (Service) - computes the likelihood that a sale will go though, triggers events from buyers and sellers
- **Buyers** Queue - list of buyers
- **Sellers** Queue - list of sellers

**Parameters**

- TravelCost: cost to buyer for checking out a home
- CommissionRate: cost in percent for real estate agent commission
- FixedSaleCost: fixed cost to buyer to sell any home

**Events and Actions**:

- Buyer:
  - LooksIntoHouse: buyer decides which housed to visit
  - VisitsHouse: buyer decides to spend a cost to visit house
  - failedBuy: buyer visted house but failed to buy
  - Bought: buyer bought the house and is removed from the buyers list
- Seller:
  - failedToSell: sale failed and will increase bias towards selling for less
  - sold: sold the house, removed from the market **Activities**:
- Buyer:
  - **Buying**
    
    LooksToBuy ---------- Buying ---------- HouseSold
  - **Owning/Renting**
    
    Moves ---------- Owning/Renting ---------- Moves

**Parameters**:

- **Trend**: constant or function of time that determines the multiplier on Persons' willingToPay.
- **Renovation Time**: time to renovate a house

# Appendix

**Entities**:

| **Consumer Class: Buyer** | |
|---|---|
| one of many buyers | |
| **Attribute** | **Description** |
| value : num | determined by buy price and events |
| budgetPrice : enum(owner \| renter \| empty) | type of inhabiter |
| bais : list(houses) | willingness to compromise |

...

| **Consumer Class:**<br>**Person** |
|---|
| one of many people who<br>buy/sell/live in houses |

| Attribute | Description |
|---|---|
| lookingToRent : num | how likely one is to rent over own |
| willingToBuyFor : num | how much one is willing to pay. Decreases significantly after house purchase |
| willingToSellFor : num | how much one is willing to sell for, increases significantly after house purchase, decreases over time closer to housing value (willing to move increases). |

...

| Queue Urnary: Market | |
|---|---|
| a queue of Houses looking to be sold | |
| **Attribute** | **Description** |
| houseList : list(houses) | Houses currently on the market |
| timeLastSold : time | time elapsed since last home sale |

...

| Queue Urnary: Customers | |
|---|---|
| a queue of Persons looking to buy a house | |
| **Attribute** | **Description** |
| personList : list(persons) | Persons currently in the market |

...

**Constants**:
n/a
...

**Parameters**:

| Name | Description | Value |
|---|---|---|
| Trend | determines the multiplier on Persons' willingToPay | constant or function of time : (%) |
| Renovation_Time | time to renovate a house | constant or function of time : (days) |

**Activities : House**

| Activity: Renovating | |
|---|---|
| house is currently being renovated | |
| scheduled | |
| initiating event | Renovates |
| duration | Renovation_Time |
| Terminating event | HouseValueChange |

| Activity: OnMarket | |
|---|---|
| house is currently listed on the market, in the market queue | |
| scheduled | |
| initiating event | HouseListed |

| | |
|---|---|
| duration | based on supply and demand |
| Terminating event | HouseSold |

**Activity: Sitting**

| | |
|---|---|
| house is currently sitting with certain inhabitants | |
| conditional | |
| initiating event | HouseInhabitantsChange |
| duration | conditional (HouseInhabitantsChange \| HouseSold) |
| Terminating event | HouseValueChange |

**Actions : House**

**Activity: Renovating**

| | |
|---|---|
| house is currently being renovated | |
| scheduled | |
| initiating event | Renovates |
| duration | Renovation_Time |
| Terminating event | HouseValueChange |

**Activities : Person**

**Activity: Buying**

| | |
|---|---|
| Person enters the market for a house | |
| initiating event | LooksToBuy |
| duration | conditional |
| Terminating event | HouseSold |

**Activity: Owning/Renting**

| | |
|---|---|
| Person is owning or renting | |
| initiating event | Moves |
| duration | conditional |
| Terminating event | Moves |

References: An Empirical Stationary Equilibrium Search - Model of the Housing Market Modeling of the housing market