# Food Magnate Simulation

## Introduction

The *Food Magnate Simulation* models how profitable different types of restaurant chain would be within a simulated settlement. The structure of the simulation, conceptually, is as follows:

**Simulation**

(Large) Settlement

Household

Household

Household

Household

Company

Outlet

Outlet

Outlet

Company

Outlet

Outlet

Outlet

Company

Outlet

Outlet

Outlet

The program contains the following objects:

* A single object of type Simulation, which is responsible for constructing, either directly or indirectly, the remaining objects
* A single object of type Settlement, which is stored within the Simulation object. Alternatively, this can be an object of type LargeSettlement, which behaves identically but can cover a larger area and contain more households.
* An unlimited number of Household objects, each of which is contained within the settlement. Initially, a settlement begins with 250 of these, but a LargeSettlement object might have more.
* An unlimited number of Company objects, although the default starting number of companies is three. The Company objects are contained directly within the Simulation object.
* An unlimited number of Outlet objects, with each outlet being stored in a particular Company object.   
  A company cannot exist without at least one outlet.

Let's look at the attributes for each class in detail…

## Class: Simulation

The Main subroutine creates the program's single Simulation object and calls its Run subroutine. From that point onwards, it is the Simulation object that is responsible for creating and managing all other objects. Its attributes are as follows:

* A single Settlement object, SimulationSettlement; since the Settlement object contains the Household objects, the simulation cannot interact directly with a household without calling an accessor subroutine in the Settlement class
* An integer variable to store the number of companies, NoOfCompanies
* A float called FuelCostPerUnit, which is used to calculate delivery costs between outlets belonging to the same company; it is passed to the constructor of the Company class
* A second float, called BaseCostForDelivery, which is also passed to the Company constructor and subsequently used to calculate delivery costs
* A list called Companies, to store objects of type Company; a list is a better choice than an array, since any number of companies can be stored

## Class: Company

A company can be either a fast-food company, a family company or a named chef company. This means that all outlets belonging to a company will also be of that type; a company cannot have, for example, some fast-food outlets and some family outlets. The Company class's attributes are as follows:

* A string, Name, to store the name of the company
* A second string, called Category; this stores the type of company, and can be either 'fast food', 'family' or 'named chef'
* A series of float attributes:
  + Balance: the amount of money a company owns, which can be positive, negative or zero
  + ReputationScore: a measure of how well regarded the company is; a company with a high reputation score is more likely to be visited than one with a low reputation score
  + AvgCostPerMeal: how much the company pays for a meal
  + AvgPricePerMeal: how much a customer pays for a meal when visiting one of the company's outlets
  + DailyCosts: single cost per day, per company, initially set to 100, but with a small chance that it will change, up or down, between days
  + FamilyOutletCost: the cost of opening an outlet for a 'family' company
  + FastFoodOutletCost: the cost of opening an outlet for a 'fast food' company
  + NamedChefOutletCost: the cost of opening an outlet for a 'named chef' company
  + FuelCostPerUnit: for companies with multiple outlets, this is part of the delivery cost from their 'main' outlet (the first one to be created) and each subsequent outlet
  + BaseCostOfDelivery: for companies with any number of outlets, this amount is paid once per day
* A list called Outlets, to store objects of type Outlet; again, this offers more flexibility than using an array
* A series of integer attributes:
  + FamilyFoodOutletCapacity: capacity for a 'family' outlet, initialised to 150
  + FastFoodOutletCapacity: capacity for a 'fast food' outlet, initialised to 200
  + NamedChefOutletCapacity: capacity for a 'named chef' outlet, initialised to 50

## Class: Outlet

An Outlet object is stored within a data structure in a Company object, modelling the fact that an outlet is owned by a single company. Outlets are also associated with the settlement, since each outlet has a location within that settlement, and households, if they choose to visit a company's outlet, will always visit the closest one.

The Outlet class's attributes are as follows:

* A series of integer attributes:
  + VisitsToday: the number of times a household has visited this outlet on the current day; it is worth noting that there is nothing preventing this exceeding the outlet's capacity
  + XCoord and YCoord: the outlet's location within the settlement
  + Capacity: how many seats are in the outlet, used to calculate daily costs
  + MaxCapacity: the maximum to which the capacity can be extended, also used to calculate daily costs
* A float, DailyCosts, which is how much the outlet costs to run each day, irrespective of visitors; this is not fixed, and can be changed as capacity changes. It could also be changed as a result of a call to the Outlet class's AlterDailyCost subroutine; however, this is never called, nor is GetCapacity.

## Class: Settlement

The simulation contains a single settlement, and each household and outlet has a location within that settlement. The settlement is easy to visualise in terms of a grid, but it is not stored as a two-dimensional array. The reason for this is twofold. Firstly, most elements within a settlement array would be empty. The default settings create a settlement with one million possible locations, but only 250 households and 12 company outlets. Secondly, a settlement contains objects of different types – namely households and outlets. Instead, each outlet and each household stores its own X and Y coordinates, which must be within the bounds of the settlement.

The Settlement class's attributes are as follows:

* An integer variable, StartNoOfHouseholds, which is how many households exist at the start of the simulation. Based on existing code, this can go up but not down as the simulation runs. The default value is 250.
* Two further integers, XSize and YSize, which store between them the size of the settlement. The default value for each of these is 1,000, meaning there are one million possible locations.
* A list, Households, to store each Household object

The LargeSettlement class inherits from Settlement, and allows the user to add to (but not subtract from) the values of StartNoOfHouseholds, XSize and YSize.

## Class: Household

To all intents and purposes, this class models a consumer, since the whole household either eats out or does not. There is scope here for modelling households of different sizes, or households in which some members (but not all) go out to eat, or even households where different individuals eat out at different outlets at the same time. The Household class's attributes are as follows:

* A float, ChanceEatOutPerDay, storing a value between 0 and 1, representing the probability of the household going out to eat
* Integer variables XCoord and YCoord, representing a household’s location in the settlement
* A static integer, NextID, which numbers each new household incrementally, starting from 1
* An additional integer, ID, which stores the value contained in NextID at point of instantiation

## Overview

### When the program begins:

Simulation()

Company()

Company.  
OpenOutlet()

LargeSettlement()

Simulation.  
AddCompany()

Settlement()

Main

Simulation.  
Run()

1. A new Simulation object is constructed
2. That Simulation object constructs a new object of type Settlement or LargeSettlement (which inherits from Settlement), depending on user input. The Settlement constructor calls the constructor of Household repeatedly.
3. That Simulation object, as part of being constructed, also constructs new Company objects, which are either the three default companies hard-coded into the Skeleton Program or user-defined, depending on user input
4. The AddCompany subroutine is called if user-defined companies are selected, allowing the user to enter specific details of each company
5. The Company constructor will make at least one call to the Outlet constructor, since each company must have at least one outlet
6. The Simulation object's Run subroutine is called (see next hierarchy diagram)

|  |  |
| --- | --- |
|  | *Main menu* |
|  | *'Modify Company' menu, shown when the user selects option 3 from the main menu and enters a valid company name* |

### When the Run subroutine is called:

Simulation.  
Run()

Simulation.  
DisplayMenu()

Simulation.  
DisplayCompanies()

Simulation.  
ModifyCompany()

Simulation.  
ProcessDayEnd()

Settlement.  
DisplayHouseholds()

Simulation.Get  
IndexOfCompany()

Simulation.  
AddCompany()

1. The user is presented with a menu as a result of a call to DisplayMenu
2. If the user selects option 1, a call is made to DisplayHouseholds, which lists details of all households in the settlement via a call to DisplayHouseholds in the Settlement class, which in turn calls GetDetails in each instance of the Household class
3. If the user selects option 2, details of all companies are displayed via a call to DisplayCompanies, which calls the GetDetails subroutine in each Company object. The GetDetails subroutine in the Company class calls a GetDetails subroutine for each outlet, meaning that selecting option 2 from the menu results in details of all companies, and their outlets, being displayed.
4. If the user selects option 3, to modify a company, they are prompted for a company name, which is then passed to GetIndexOfCompany in order to get the index of that company. The subsequent call to ModifyCompany presents the user with a second menu containing three options. They can open a new outlet (OpenOutlet()), close an outlet (CloseOutlet()) or expand an existing outlet (ExpandOutlet()).
5. If the user selects option 4, a new company can be created via a call to AddCompany. This prompts the user for details of the new company's name, type and starting balance. The new company will have a single outlet in a random location within the settlement.
6. There is no option 5, which makes it quite likely that adding an option 5 will be part of the exam
7. If the user selects option 6, a call is made to ProcessDayEnd in the Simulation class. Note that an identically named subroutine exists in the Company class, so be sure not to confuse the two. This is the most involved subroutine in the program and is addressed in the next hierarchy diagram.

### When each day ends:

Simulation.  
ProcessDayEnd()

Company.  
NewDay()

Company.  
GetReputation  
Score()

Settlement.  
GetNumberOf  
Households()

Settlement.  
FindOutIfHousehold  
EatsOut()

Company.  
AddVisitTo  
NearestOutlet()

Simulation.  
DisplayCompanies  
AtDayEnd()

Simulation.  
DisplayEvents  
AtDayEnd()

1. The call to NewDay (in the Company class) calls NewDay (a subroutine in each Outlet class) to reset the number of visits to zero
2. The reputation score of each company is accessed and added to a list, with each float value therein storing a running total of all reputation scores so far (i.e. the first value stored is for the first company, the second value stored is the sum of the reputation scores for the first two companies, the third value stored is the sum of the reputation scores for the first three companies, and so on)
3. The call to GetNumberOfHouseholds is to facilitate a loop through all households in a settlement
4. The call to GetHouseholdLocation allows the Simulation class to access an individual household
5. The call to GindOutIfHouseholdEatsOut, for each Household object, returns a Boolean, which is more likely to be true for households with a higher probability of eating out. If it is true, a company is selected at random, using the list from step 15, with companies holding a higher reputation score more likely to be chosen.
6. For the company that is chosen, the nearest outlet to the household eating out is visited. The distances to all outlets belonging to the chosen company are examined, and distances are calculated in the following way:

* Distances are calculated using Pythagoras; taking the x distance, squaring it, adding the square of the y distance and then taking the square root of the result.
* In the event that two outlets are of an equal distance from a house, the outlet examined first (i.e. the outlet appearing earlier in the Outlets list) will be the one visited.

1. The call to DisplayCompaniesAtDayEnd calls the ProcessDayEnd subroutine in the Company class. This subroutine calculates changes to the company's balance, which is affected by visits to each outlet per day, the price at which meals are bought and sold, and the distance between outlets for the same company, for which the delivery of ingredients incurs a cost based on the price of fuel (distances are calculated as above). The old balance and the new balance are then displayed, along with other details of the company and its outlets.
2. The call to DisplayEventsAtDayEnd generates either a random probability of additional households in the settlement, a change of fuel cost for a company chosen at random, a change of reputation for a company chosen at random or a change of daily costs for a company chosen at random

## Program Subroutines

The program’s functions Ⓕ and procedures Ⓟ are described below.

### 'Household' Class

| Subroutine | Data | Description |
| --- | --- | --- |
| GetDetails  Ⓕ | Parameters: -  Returns: Details (str)  Called From: Settlement.DisplayHouseholds  Calls: - | 1. Declares an empty string called Details  2. Populates it with ID, XCoord and YCoord  3. Returns the string |
| GetChanceEatOut  Ⓕ | Parameters: -  Returns: ChanceEatOutPerDay (float)  Called From: Settlement.FindOutIfHouseholdEatsOut  Calls: - | Returns the value of the protected attribute ChanceEatOutPerDay |
| GetX  Ⓕ | Parameters: -  Returns: XCoord (int)  Called From: Settlement.GetHouseholdLocation  Calls: - | Returns the value of the protected attribute XCoord |
| GetY  Ⓕ | Parameters: -  Returns: YCoord (int)  Called From: Settlement.GetHouseholdLocation  Calls: - | Returns the value of the protected attribute YCoord |

### 'Settlement' Class

| Subroutine | Data | Description |
| --- | --- | --- |
| GetNumber  ofHouseholds  Ⓕ | Parameters: -  Returns: len(Households) (int)  Called From: Simulation.ProcessDayEnd  Calls: - | Returns the number of items in the protected attribute list Households |
| GetXSize  Ⓕ | Parameters: -  Returns: XSize (int)  Called From: Simulation.ModifyCompany  Calls: - | Returns the value of the protected attribute XSize |
| GetYSize  Ⓕ | Parameters: -  Returns: YSize (int)  Called From: Simulation.ModifyCompany  Calls: - | Returns the value of the protected attribute YSize |
| GetRandomLocation  Ⓕ | Parameters: -  Returns: X, Y (int, int)  Called From: Simulation.AddCompany  Settlement.AddHousehold  Calls: - | 1. Creates a two integers, X and Y  2. Sets X to a random integer between 0 and XSize-1  3. Sets Y to a random integer between 0 and YSize-1  4. Returns the variables as a tuple |
| CreateHouseholds  Ⓟ *(protected)* | Parameters: -  Returns: -  Called From: Settlement constructor  Calls: Household.AddHousehold | 1. Loops StartNoOfHouseholds (protected attribute) times  2. In each iteration, call AddHousehold |
| AddHousehold  Ⓟ | Parameters: -  Returns: -  Called From: Settlement.CreateHouseholds  Calls: Household.GetRandomLocation | 1. Set X and Y to the return values from GetRandomLocation  3. Create a new Household object using this location  4. Add the Household object to the protected attribute Households list |
| DisplayHouseholds  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.Run  Calls: Household.GetDetails | 1. Prints out a heading over 3 lines  2. Loops through the Households (protected attribute)  3. For each household prints out the return value of Household.GetDetails() |
| FindOutif HouseholdEatsOut  Ⓕ | Parameters: HouseholdNo (int)  Returns: bool, X (int), Y (int)  Called From: Simulation.ProcessDayEnd  Calls: Household.GetX, Household.GetY, Household.GetChanceEatOut | 1. Creates a variable called EatOutRNo and sets it to a random float between 0 and 1  2. If this number is less than the probability of the household's (that's the HouseholdNo passed as a parameter) chance of eating out, return True and the X & Y coordinates of the household  3. Otherwise, return False and the X & Y coordinates of the household |

### 'Outlet' Class

| Subroutine | Data | Description |
| --- | --- | --- |
| GetCapacity  Ⓕ | Parameters: -  Returns: Capacity (int)  Called From: -  Calls: - | Returns the value of the protected attribute Capacity |
| GetX  Ⓕ | Parameters: -  Returns: XCoord (int)  Called From: Company.AddVisitToNearestOutlet  Company.GetDistanceBetweenTwoOutlets  Calls: - | Returns the value of the protected attribute XCoord |
| GetY  Ⓕ | Parameters: -  Returns: YCoord (int)  Called From: Company.AddVisitToNearestOutlet  Company.GetDistanceBetweenTwoOutlets  Calls: - | Returns the value of the protected attribute YCoord |
| AlterDailyCost  Ⓟ | Parameters: Amount (float)  Returns: -  Called From: -  Calls: - | Accepts a value as a parameter and adds this value to the DailyCosts attribute  **NB This method is not called from anywhere.** |
| AlterCapacity  Ⓕ | Parameters: Change (int)  Returns: Change (int)  Called From: Company.ExpandOutlet  Calls: - | 1. Stores the current capacity in a local variable, OldCapacity  2. Adds the parameter to the Capacity attribute; the parameter can be a negative number, in which case Capacity is decreased  3. If the Capacity has, as a result of step (2), risen above MaxCapacity, set Capacity to MaxCapacity and return the difference between OldCapacity and MaxCapacity (i.e. the amount of increase after Capacity was limited)  4. If the Capacity has fallen below zero, set Capacity to zero  5. Re-calculate DailyCosts based on the new Capacity and return the value of Change  **NB If there is a negative change attempted that is larger than possible, e.g. an outlet with a capacity of 3 has this method called with a parameter of -5, it’s -5 that is then returned, even though the capacity only goes down by three.** |

| Subroutine | Data | Description |
| --- | --- | --- |
| IncrementVisits  Ⓟ | Parameters: -  Returns: -  Called From: Company.AddVisitToNearestOutlet  Calls: - | Adds 1 to the VisitsToday attribute |
| NewDay  Ⓟ | Parameters: -  Returns: -  Called From: Company.NewDay  Outlet constructor  Calls: - | Sets the VisitsToday attribute to zero |
| CalculateDaily ProfitLoss  Ⓕ | Parameters: AvgCostPerMeal (float),  AvgPricePerMeal (float)  Returns: float  Called From: Company.ProcessDayEnd  Calls: - | The calculation of daily profit or loss entails calculating the profit/loss for a single meal, multiplying by the number of meals, then subtracting the outlet’s DailyCosts. |
| GetDetails  Ⓕ | Parameters: -  Returns: Details (str)  Called From: Company.GetDetails  Calls: - | Returns a string, containing an intelligible representation of each of the attributes (possibly this should be an \_\_str\_\_ method) |

## 'Company' Class

| Subroutine | Data | Description |
| --- | --- | --- |
| GetName  Ⓕ | Parameters: -  Returns: Name (str)  Called From: Simulation.DisplayCompaniesAtDayEnd  Simulation.ProcessCostOfFuelChangeEvent  Simulation.ProcessReputationChangeEvent  Simulation.ProcessCostChangeEvent  Simulation.GetIndexOfCompany  Calls: - | Returns the value of the protected attribute Name |
| GetNumberOfOutlets  Ⓕ | Parameters: -  Returns: length of the Outlets list (int)  Called From: Simulation.ModifyCompany  Calls: - | Returns the number of outlets in the Outlets list |
| GetReputationScore  Ⓕ | Parameters: -  Returns: ReputationScore (float)  Called From: Simulation.ProcessDayEnd  Calls: - | Returns the value of the protected attribute ReputationScore |
| AlterDailyCosts  Ⓟ | Parameters: Change (float)  Returns: -  Called From: Simulation.ProcessCostChangeEvent  Calls: - | Accepts a value as a parameter and adds this value to the DailyCosts attribute |
| AlterAvgCost PerMeal  Ⓟ | Parameters: Change (float)  Returns: -  Called From: Simulation.ProcessCostChangeEvent  Calls: - | Accepts a value as a parameter and adds this value to the AvgCostPerMeal attribute |
| AlterFuelCost PerUnit  Ⓟ | Parameters: Change (float)  Returns: -  Called From: Simulation.ProcessCostOfFuelChangeEvent  Calls: - | Accepts a value as a parameter and adds this value to the FuelCostPerUnit attribute |
| AlterReputation  Ⓟ | Parameters: Change (float)  Returns: -  Called From: Simulation.ProcessReputationChangeEvent  Calls: - | Accepts a value as a parameter and adds this value to the ReputationScore attribute |
| NewDay  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.ProcessDayEnd  Calls: Outlet.NewDay | 1. Loops through each outlet in the Outlets list  2. Calls the NewDay method for each outlet, resetting the number of visits for each to zero |
| AddVisitTo  NearestOutlet  Ⓟ | Parameters: X (int), Y (int)  Returns: -  Called From: Simulation.ProcessDayEnd  Calls: Outlet.GetX  Outlet.GetY  Outlet.IncrementVisits | 1. The parameters, X and Y, are the coordinates of a house whose occupant(s) will eat out on the current day  2. Initialise local variable NearestOutlet to zero  3. Initialise NearestOutletDistance to first outlet in the list  4. Loops through each outlet in the Outlets list  5. Calculates the distance from the Household (the X and Y coordinates of which were passed in) to the current Outlet  6. If the current outlet being examined is closer than the closest found so far, store the index of the current Outlet in NearestOutlet and the distance to it in NearestOutletDistance  7. After the loop, access the outlet indexed by NearestOutlet and call its IncrementVisits function. |
| GetDetails  Ⓕ | Parameters: -  Returns: Details (str)  Called From: Simulation.DisplayCompanies  Calls: Company.CalculateDeliveryCost  Outlet.GetDetails | 1. Declares an empty string  2. Appends all attributes to this string, including the contents of Outlets, which are addressed in turn via a for loop which calls the GetDetails method on each of them  3. Returns the string - this is essentially an \_\_str\_\_ method. |
| ProcessDayEnd  Ⓕ | Parameters: -  Returns: Details (str)  Called From: Simulation.DisplayCompaniesAtDayEnd  Calls: Company.CalculateDeliveryCost  Outlet.CalculateDailyProfitLoss | 1. Declares an empty string, Details.  2. Initialises new local variables ProfitLossFromOutlets and ProfitLossFromThisOutlet to 0.  3. If there is only one outlet, DeliveryCosts is set to the BaseCostOfDelivery attribute  4. If there is more than one outlet, DeliveryCosts is set to the BaseCostOfDelivery attribute plus a call to CalculateDeliveryCost.  5. DeliveryCosts is then appended to details  6. Loops through each outlet in the outlets list, calling CalculateDailyProfitLoss for each outlet  7. The return from this call is stored in ProfitLossFromThisOutlet , appended to the Details string and added to ProfitLossFromOutlets  8. Once the loop is over, the previous balance is appended to the string, and the new Balance is calculated  9. The Details string is returned |

| Subroutine | Data | Description |
| --- | --- | --- |
| CloseOutlet  Ⓕ | Parameters: ID (int)  Returns: CloseCompany (bool)  Called From: Simulation.ModifyCompany  Calls: - | 1. The outlet, identified by an id integer passed as a parameter, is removed from the Outlets list  2. If the list is now empty, return true, otherwise return false |
| ExpandOutlet  Ⓟ | Parameters: ID (int)  Returns: -  Called From: Simulation.ModifyCompany  Calls: Outlet.AlterCapacity | 1. The parameter is the index of the Outlet to be expanded  2. User is prompted for how much they want to expand capacity  3. The subsequent input is passed as a parameter to the outlet's AlterCapacity method  4. If the expansion took place in full (i.e. did not attempt to take the capacity of the Outlet beyond its maximum capacity), the message 'Capacity adjusted.' is output  5. Otherwise, the message ‘Only some of that capacity added, outlet now at maximum capacity.’ is output |
| OpenOutlet  Ⓟ | Parameters: X (int), Y (int)  Returns: -  Called From: Simulation constructor  Simulation.ModifyCompany  Company constructor  Calls: Outlet constructor | 1. Coordinates are passed as parameters  2. Depending on the type of Company (fast food, family, named chef), the company's Balance is updated by subtracting the start-up costs  3. A new Outlet object is constructed using the coordinates which were passed in as parameters as well as the Capacity (which is an integer atribute, set according to company Category)  4. The new outlet is added to the Outlets list |
| GetListOfOutlets  Ⓕ | Parameters: -  Returns: Temp (Outlet[])  Called From: Company.CalculateDeliveryCost  Calls: - | 1. A new blank list called Temp is created  2. A loop iterates through each Outlet in the Outlets list  3. Adding each Outlet in turn to the Temp list  4. Temp is returned |
| GetDistanceBetweenTwoOutlets  Ⓕ | Parameters: Outlet1 (int), Outlet2 (int)  Returns: float  Called From: Company.CalculateDeliveryCost  Calls: Outlet.GetX  Outlet.GetY | Returns the distance between two outlets. Each outlet has a grid position, and the distance between them is the hypotenuse (the square root of the sum of the horizontal difference (between the column in which each one exists) and the vertical distance (between the row in which each one exists)) i.e. a straight line which is diagonal unless they share a row or column. |

| Subroutine | Data | Description |
| --- | --- | --- |
| Calculate  DeliveryCost  Ⓕ | Parameters: -  Returns: TotalCost (float)  Called From: Company.GetDetails  Company.ProcessDayEnd  Calls: Company.GetListOfOutlets  Company.GetDistanceBetweenTwoOutlets | 1. Calls GetListOfOutlets to receive a list of Outlets and stores it in ListOfOutlets  2. Declares a float, TotalDistance, initialised to 0.0  3. Loops once per outlet minus 1 (this method is actually not called if a company only has one outlet)  4. Calculates the distance between the current outlet (0 on the first iteration, 1 on the second, 2 on the third, etc.) and the outlet indexed one value higher (i.e. distance between outlets 0 and 1, distance between outlets 1 and 2, distance between outlets 2 and 3, etc.)  5. For each run through the loop, add this distance to TotalDistance  6. Return the product of TotalDistance and the FuelCostPerUnit attribute |

### 'Simulation' Class

| Subroutine | Data | Description |
| --- | --- | --- |
| DisplayMenu  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.Run  Calls: - | Displays menu items, one on each line and prompts the user for a choice; this method does not validate input or even store the user's response; user input is handled in run  **NB The menu has no option 5** |
| DisplayCompanies AtDayEnd  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.ProcessDayEnd  Calls: Company.GetName  Company.ProcessDayEnd | 1. Outputs 'Companies:' to the console  2. Loops through each Company in the Companies list  3. Outputs the return value of a call to Company.GetName()  4. Outputs the return value from Company.ProcessDayEnd() for the current Company |
| ProcessAdd  HouseholdsEvent  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.DisplayEventsAtDayEnd  Calls: Settlement.AddHousehold | 1. Assigns a local variable NoOfNewHouseholds to a random integer between 1 and 4  2. Loops that many times, calling Settlement.AddHousehold, which adds that many new households to the settlement  3. Prints the number of new households added to the settlement |

| Subroutine | Data | Description |
| --- | --- | --- |
| ProcessCostOf FuelChangeEvent  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.DisplayEventsAtDayEnd  Calls: Company.GetName  Company.AlterFuelCostPerUnit | 1. Assigns a local variable FuelCostChange to a random number, always to a single decimal place, between 0.1 and 0.9  2. Assigns a local variable UpOrDown to a random integer, either 0 or 1  3. Assigns a local variable CompanyNo to a random company index  4. If UpOrDown is zero, a message is output saying that the CompanyNo’s fuel cost has gone up by FuelCostChange, otherwise a message saying it goes down by that amount is output instead and the sign of FuelCostChange is negated  5. CompanyNo’s AlterFuelCostPerUnit method is invoked, passing FuelCostChange as the parameter |
| ProcessReputation  ChangeEvent  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.DisplayEventsAtDayEnd  Calls: Company.GetName  Company.AlterReputation | 1. Assigns a local variable ReputationChange to a random number, always to a single decimal place, between 0.1 and 0.9  2. Assigns a local variable UpOrDown to a random integer, either 0 or 1  3. Assigns a local variable CompanyNo to a random company index  4. If UpOrDown is zero, a message is output saying that the CompanyNo’s fuel cost has gone up by ReputationChange, otherwise a message saying it goes down by that amount is output instead and the sign of ReputationChange is negated  5. CompanyNo’s AlterReputation method is invoked, passing ReputationChange as the parameter |

| Subroutine | Data | Description |
| --- | --- | --- |
| ProcessCost  ChangeEvent  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.DisplayEventsAtDayEnd  Calls: Company.GetName  Company.AlterDailyCosts  Company.AlterAvgCostPerMeal | 1. Initialises a local variable AmountOfChange to 0.0  2. Assigns a local variable UpOrDown to a random integer, either 0 or 1  3. Assigns a local variable CostToChange to a random integer, either 0 or 1  3. Assigns a local variable CompanyNo to a random company index  4. If CostToChange is 0:  a. Generate a random one-decimal-place number between 0.1 and 1.9  b. If UpOrDown is zero, a message is output saying that the daily costs for CompanyNo have gone up by AmountOfChange  c. Otherwise a message saying that the daily costs for CompanyNo have gone down by AmountOfChange and then negate the sign of AmountOfChange  d. Invoke the AlterDailyCosts method of company CompanyNo passing AmountOfChange as the parameter  5. Otherwise:  a. Generate a random one-decimal-place number between 0.1 and 0.9  b. If UpOrDown is zero, a message is output saying that the average cost for a meal for CompanyNo have gone up by AmountOfChange  c. Otherwise a message saying that the average cost for a meal for CompanyNo have gone down by AmountOfChange and then negate the sign of AmountOfChange  d. Invoke the AlterAvgCostPerMeal method of company CompanyNo passing AmountOfChange as the parameter |

| Subroutine | Data | Description |
| --- | --- | --- |
| DisplayEvents AtDayEnd  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.ProcessDayEnd  Calls: Simulation.ProcessAddHouseholdsEvent  Simulation.ProcessCostOfFuelChangeEvent  Simulation.ProcessReputationChangeEvent  Simulation.ProcessCostChangeEvent | 1. Prints out 'Events:'  2. Generates a random float, with a 25% chance of entering a selection structure  3. If the selection structure is entered, another random float is generated, giving a 25% chance of a call to ProcessAddHouseholdsEvent  4. Another random float is generated, still within the selection structure described in step (2); this one causes a 50% chance of a call to ProcessCostOfFuelChangeEvent  5. Still within the selection structure from step (2), another random float is generated, causing a 50% chance of a call to ProcessReputationChangeEvent  6. Still within the selection structure from step (2), a final random float is generated, causing a 50% chance of a call to ProcessCostChangeEvent  7. If the original random float (step 2) did not cause the initial selection structure to be entered, 'No events' is output to the console  **NB The events in steps 3 to 6 are independent of one another, meaning they might all happen, or none of them, or any combination. If none happen, but the initial selection structure from step (2) is entered, 'No events' will not be displayed** |

| Subroutine | Data | Description |
| --- | --- | --- |
| ProcessDayEnd  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.Run  Calls: Company.NewDay  Company.GetReputationScore  Simulation.GetNumberOfHouseholds  Settlement.GetHouseholdLocation  Settlement.FindOutIfHouseholdEatsOut  Company.AddVisitToNearestOutlet  Simulation.DisplayCompaniesAtDayEnd  Simulation.DisplayEventsAtDayEnd | 1. Initialise a new local variable TotalReputation to 0.0  2. Initialise a new local variable Reputations as an empty list  3. Loop through each company in the companies list  4. Call the NewDay method for each Company  5. Invoke GetReputationScore() on each company and add the result to TotalReputation  6. For each company, append TotalReputation to Reputations  7. Loop through each household in the settlement  8. For each household, if an occupant will now visit a restaurant (determined using a random number in FindOutIfHouseholdEatsOut), set a local variable CompanyRNo to a random integer between 1 and int(TotalReputation)  9. Loop through Reputations, incrementing a zero-initialised variable Current with each iteration  10. If CompanyRNo (from step 8) is less than the value in Reputations[Current], a call is made for to AddVisitToNearestOutlet passing the parameters X & Y which are the coordinates of the current household that were returned by FindOutIfHouseholdEatsOut. The while loop is then exited (using break) as each household can only visit one outlet.  11. Invoke DisplayCompaniesAtDayEnd for this Simulation  12. Invoke DisplayEventsAtDayEnd for this Simulation |
| AddCompany  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.Run  Calls: Company.GetRandomLocation  Company constructor | 1. Local variable CompanyName initialised to user input  2. Local variable Balance initialised to user input (converted to int)  3. User is repeatedly prompted to enter 1, 2 or 3 (fast food, family restaurant, named chef respectively) until either 1, 2 or 3 is entered  4. Depending on user input, the local variable TypeOfCompany is set to either 'fast food', 'family' or 'named chef'  5. A call to GetRandomLocation gets the new X and Y coordinates  6. New Company object is constructed and added to the Companies list |
| GetIndexOf  Company  Ⓕ | Parameters: CompanyName (str)  Returns: Index (int)  Called From: Simulation.Run  Calls: Company.GetName | 1. Local variable Index declared and set to -1  2. Loop through all companies in turn  3. If a company name matches the passed parameter (both converted to  lower case), return the index of that company  4. If the loop terminates without a match, return -1 |

| Subroutine | Data | Description |
| --- | --- | --- |
| ModifyCompany  Ⓟ | Parameters: Index (int)  Returns: -  Called From: Simulation.Run  Calls: Company.GetNumberOfOutlets  Company.CloseOutlet  Company.ExpandOutlet  Settlement.GetXSize  Settlement.GetYSize  Company.OpenOutlet | 1. Menu displayed, with 3 options and user input is accepted  2. If input is 2 (close outlet) or 3 (expand outlet), the user is prompted for the ID  of the outlet; if input is 1 (open outlet), jump to step 7  3. Entered ID is cast as an integer and stored in OutletIndex  4. If 2 (close outlet) was entered, a call to CloseOutlet is made, and if that was the last outlet for that company, the company is removed  5. If 3 (expand outlet) was entered, a call to ExpandOutlet is made  6. If an out-of-range ID is entered in step 3, display error message  7. If input is 1 (open new outlet), prompt user for X and Y coordinates and store them as integers in the local variables X and Y  8. If the coordinates are within the grid bounds, call OpenOutlet, passing  X and Y as parameters  9. Otherwise, display an error message  **NB No error is displayed if an entry other than 1, 2 or 3 is made** |
| DisplayCompanies  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.Run  Calls: Company.GetDetails | 1. Prints out Details of all companies:'  2. Loop initiated to run once per company in the protected attribute Companies  3. The return value of a call to GetDetails for each company is printed out |
| Run  Ⓟ | Parameters: -  Returns: -  Called From: Main  Calls: Simulation.DisplayMenu  Settlement.DisplayHouseholds  Simulation.DisplayCompanies  Simulation.GetIndexOfCompany  Simulation.ModifyCompany  Simulation.AddCompany  Simulation.ProcessDayEnd | 1. Initiates a loop that will repeat until the user enters 'Q' at the prompt  2. If the user enters '1', call DisplayHouseholds (on the settlement for this simulation)  3. If the user enters '2', call DisplayCompanies  4. If the user enters '3', prompt the user for a company name, which is passed to GetIndexOfCompany. The call to this method will return the integer index of the company, if it exists, and -1 if it doesn't. If -1 is returned, the user is prompted for a company name until -1 is not returned. Once a valid index has been returned, this is passed as a parameter in a call to ModifyCompany  5. If the user enters '4', call AddCompany  6. If the user enters '6', call ProcessDayEnd  7. If the user enters 'Q', an exit message is displayed and the program ends.  **NB There is no option 5 on the menu** |

## Program Classes

The classes defined in the program, and their attributes, are described below.

| Class | Description |
| --- | --- |
| Household | An individual household within a settlement; each household has its own unique location, and each household has a probability of a person eating out each day. |
| Settlement | A 1,000 by 1,000 grid, with some spaces within the grid occupied by a household (initially 250). Households are stored within a list, meaning the empty 'cells' of the grid are not themselves represented. |
| LargeSettlement | Inherits from Settlement, but the grid can be larger than 1,000 by 1,000, and the starting number of households can be larger than 250. |
| Outlet | An individual restaurant, belonging to a company. Although each outlet is a distinct object, all outlets of a company are of a particular category (i.e. all fast food, all family or all named chef - never a combination). |
| Company | Each company consists of one or more outlets, each of which is stored in a list. A company can close an outlet, expand an outlet or open a new outlet. |
| Simulation | 'Manages' each day, processing events which are generated via random numbers. These events include the construction of new households within a settlement as well as changes to a company in cost or reputation. |

### ‘Household’ Attributes

| Attribute | Type | Default Value | Accessibility | Description |
| --- | --- | --- | --- | --- |
| ChanceEatOutPerDay | Float | Random (0..1) | Protected | Represents the probability that a member of the household will eat out during any one day |
| ID | Integer | Initialised to NextID +1 | Protected | Unique identifier for each household, with the first household numbered '1', the second '2' and so forth |
| NextID | Integer | Class attribute (static), initialised to 1 | Protected | Class attribute (only one for the whole class), used to keep track of the next ID to use |
| XCoord | Integer | Passed as a parameter to constructor | Protected | The x coordinate of the household's location within the settlement |
| YCoord | Integer | Passed as a parameter to constructor | Protected | The y coordinate of the household's location within the settlement |

### ‘Settlement’ Attributes

The LargeSettlement class inherits from the Settlement class, but does not include any additional methods or attributes. Instead, it includes three additional parameters to the constructor, allowing any calling class to vary the initialisation values of XSize, YSize and StartNoOfHouseholds

| Attribute | Type | Accessibility | Default Value | Description |
| --- | --- | --- | --- | --- |
| Households | List of Household objects | Protected | Empty | Collection of all Household objects within the settlement |
| StartNoOfHouseholds | Integer | Protected | 250 | The number of households in the settlement at the beginning of the simulation |
| XSize | Integer | Protected | 1,000 | One dimension of the size of the settlement 'grid'. As it's the 'x' dimension, it might be helpful to visualise this as the settlements width, but the program does not store the grid in any form, such as a 2D array |
| YSize | Integer | Protected | 1,000 | The other dimension for the size of the settlement |

### ‘Outlet’ Attributes

| Attribute | Type | Accessibility | Default Value | Description |
| --- | --- | --- | --- | --- |
| Capacity | Integer | Protected | See 🡪 | The maximum number of visits that an outlet can receive, initialised to 60% of MaxCapacityBase, which is a parameter passed to the Outlet class's constructor. |
| DailyCosts | Float | Protected | See 🡪 | The cost of running the outlet irrespective of customer footfall. This is initialised to a sum of the following:   * £0.50 per seat within Capacity * £0.20 per potential seat up to MaxCapacityBase (in addition to £0.50 for seats within Capacity) * £100 one-off |
| MaxCapacity | Integer | Protected | See 🡪 | The highest value that Capacity can be increased to by way of calls to the class's AlterCapacity method. This is initialised to the MaxCapacityBase parameter, with a random integer between 0 and 49 added to it, then another random number in the same range subtracted from it. |
| VisitsToday | Integer | Protected | 0 (via the NewDay() method call) | The number of times the outlet has been visited on the current day, which is incremented with each visit. |
| XCoord | Integer | Protected | Passed as a parameter to constructor | The x coordinate of the outlet's location within the settlement. |
| YCoord | Integer | Protected | Passed as a parameter to constructor | The y coordinate of the outlet's location within the settlement. |

### ‘Company’ Attributes

| Attribute | Type | Accessibility | Default Value | Description |
| --- | --- | --- | --- | --- |
| AvgCostPerMeal | Float | Protected | See 🡪 | The average cost for a meal (as bought by the company), set to 5, 12 and 20 for fast food, family and named chef companies respectively |
| AvgPricePerMeal | Float | Protected | See 🡪 | The average price for a meal (as sold to a customer), set to 10, 14 and 40 for fast food, family and named chef companies respectively |
| Balance | Float | Protected | Passed as a parameter to constructor | The amount of money the company has |
| BaseCostOfDelivery | Float | Protected | Passed as a parameter to constructor | Used in the calculation of delivery to a company regardless of the number of outlets |
| Category | String | Protected | Passed as a parameter to constructor | The type of company - one of 'fast food', 'family' or 'named chef'. |
| DailyCosts | Float | Protected | 100 | Part of a company's expenses, regardless of the number of outlets |
| FamilyFoodOutletCapacity | Integer | Protected | 150 | The base maximum capacity for a family outlet |
| FamilyOutletCost | Float | Protected | 1000 | The cost of a family company opening a new outlet |
| FastFoodOutletCapacity | Integer | Protected | 200 | The base maximum capacity for a fast food outlet |
| FastFoodOutletCost | Float | Protected | 2000 | The cost of a fast food company opening a new outlet |
| FuelCostPerUnit | Float | Protected | Passed as a parameter to constructor | Used in the calculation of delivery to each outlet |
| Name | String | Protected | Passed as a parameter to constructor | The name of the company. |
| NamedChefOutletCapacity | Integer | Protected | 50 | The base maximum capacity for a named chef outlet |
| NamedChefOutletCost | Float | Protected | 15000 | The cost of a named chef company opening a new outlet |
| Outlets | List of Outlet objects | Protected | Empty | Collection of all of the company's outlets |
| ReputationScore | Float | Protected | See Description | A measure of a company's reputation. It is initialised to 100, then altered depending on the category of the company, with named chef companies likely to have a higher score than family companies, which in turn are likely to have a higher score than fast food companies. This is not guaranteed however, as random numbers are used in these calculations. |

**Simulation Attributes**

| Attribute | Type | Accessibility | Default Value | Description |
| --- | --- | --- | --- | --- |
| BaseCostForDelivery | Float | Protected | 100 | Passed to Company to subsequently be used in calculations of delivery costs for a company |
| Companies | List of Company objects | Protected | Empty | Collection of all of the simulation's companies |
| FuelCostPerUnit | Float | Protected | 0.0098 | Passed to Company to subsequently be used in calculations of delivery for each of a company's outlets |
| NoOfCompanies | Integer | Protected | See 🡪 | The number of companies in the simulation, which is initialised to either 3 or a user-input value, depending on a menu selection |
| SimulationSettlement | Settlement | Protected | See 🡪 | The single settlement for the game, which will be initialised to either a new instance of Settlement or a new instance of LargeSettlement, depending on user input |