# 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
* An ArrayList 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
* A random number generator called rnd, used to generate random events between days

## 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
* An ArrayList 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
* A random number generator called rnd, used to generate random reputation scores

## 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.
* A random number generator called rnd, used to generate maxCapacity values in this class's constructor

## 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.
* An ArrayList, households, to store each Household object
* A random number generator called rnd, used to generate random locations for new households

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()

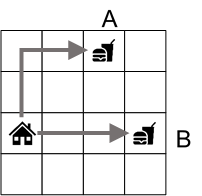
Settlement.  
GetNumberOf  
Households()

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 an ArrayList, 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 findOutIfHouseholdEatsOut, 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 ArrayList from step 15, with companies holding a higher reputation score more likely to be chosen.
5. 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 by assuming movement is only possible north, south, east or west. This means that the distance from the house to outlet A is 4, while the distance from the house to outlet B is 3.
* 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 |
| --- | --- | --- |
| GetChanceEatOut  Ⓕ | Parameters: -  Returns: ChanceEatOutPerDay: Single  Called From: Settlement.FindOutIfHouseholdEatsOut  Calls: - | Returns the value of ChanceEatOutPerDay |
| GetDetails  Ⓕ | Parameters: -  Returns: details: String  Called From: Settlement.displayHouseholds  Calls: - | 1. Declares an empty string  2. Populates it with all attributes other than Nextid and rnd  3. Returns the string |
| GetX  Ⓕ | Parameters: -  Returns: XCoord: Integer  Called From: Settlement.FindOutIfHouseholdEatsOut  Calls: - | Returns the value of XCoord |
| GetY  Ⓕ | Parameters: -  Returns: YCoord: Integer  Called From: Settlement.FindOutIfHouseholdEatsOut  Calls: - | Returns the value of YCoord |

### 'Settlement' Class

| Subroutine | Data | Description |
| --- | --- | --- |
| AddHousehold  Ⓟ | Parameters: -  Returns: -  Called From: Settlement.CreateHouseholds  Calls: - | 1. Declare integers X and Y  2. Set X and Y to random locations within the settlement  3. Create a new household object using this location  4. Add the household object to the Households ArrayList |
| CreateHouseholds  Ⓟ | Parameters: -  Returns: -  Called From: Settlement constructor  Calls: Settlement.AddHousehold | 1. Loops StartNoOfHouseholds times  2. In each iteration, call AddHousehold |
| DisplayHouseholds  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.Run  Calls: Household.GetDetails | 1. Outputs 'details of all households'  2. Loops through each household in the Households list  3. Outputs the result of a call to the GetDetails subroutine of each household |
| FindOutIf HouseholdEatsOut  Ⓕ | Parameters: HouseholdNo: Integer,  X: Integer, Y: Integer  Returns: Boolean  Called From: Simulation.ProcessDayEnd  Calls: Household.GetX  Household.GetY | 1. Generate random float between 0 and 1  2. If this number is less than the probability of the household's (that's the household passed as a parameter) chance of eating out, return 'true'  3. Otherwise, return 'false' |
| GetNumberOf Households  Ⓕ | Parameters: -  Returns: Households.Count: Integer  Called From: Simulation.ProcessDayEnd  Calls: - | Returns the number of items in the Households list |
| GetRandom  Location  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.AddCompany  Settlement.AddHousehold  Calls: - | 1. Accepts references to X and Y as parameters; as these are passed by reference, they change in the subroutine from which GetRandomLocation is called, even though there is no return value  2. Sets X and Y to random values within the bounds of the Settlement object |
| GetXSize  Ⓕ | Parameters: -  Returns: XSize: Integer  Called From: Simulation.ModifyCompany  Calls: - | Returns the value of XSize |
| GetYSize  Ⓕ | Parameters: -  Returns: YSize: Integer  Called From: Simulation.ModifyCompany  Calls: - | Returns the value of YSize |

### 'Outlet' Class

| Subroutine | Data | Description |
| --- | --- | --- |
| AlterCapacity  Ⓕ | Parameters: Change: Integer  Returns: Change: Integer  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. Recalculate the daily costs 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 subroutine called with a parameter of -5, it is -5 that is then returned, even though the capacity only goes down by three.** |
| AlterDailyCost  Ⓟ | Parameters: Amount: Single  Returns: -  Called From: -  Calls: - | Accepts a value as a parameter and adds this value to the DailyCosts attribute  **NB This subroutine is not called from anywhere.** |
| CalculateDaily ProfitLoss  Ⓕ | Parameters: AvgCostPerMeal: Single,  AvgPricePerMeal: Single  Returns: Single  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 daily costs |
| GetCapacity  Ⓕ | Parameters: -  Returns: Capacity: Integer  Called From: -  Calls: - | Returns the value of Capacity |
| GetDetails  Ⓕ | Parameters: -  Returns: Details: String  Called From: Company.GetDetails  Calls: - | Returns a string containing an intelligible representation of each of the attributes – a toString subroutine in all but name |
| GetX  Ⓕ | Parameters: -  Returns: XCoord: Integer  Called From: Company.AddVisitToNearestOutlet  Company.GetDistanceBetweenTwoOutlets  Calls: - | Returns the value of XCoord |
| GetY  Ⓕ | Parameters: -  Returns: YCoord: Integer  Called From: Company.AddVisitToNearestOutlet  Company.GetDistanceBetweenTwoOutlets  Calls: - | Returns the value of YCoord |
| 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 |

### ‘Company' Class

| Subroutine | Data | Description |
| --- | --- | --- |
| AddVisitTo  NearestOutlet  Ⓟ | Parameters: X: Integer, Y: Integer  Returns: -  Called From: Simulation.ProcessDayEnd  Calls: Outlet.GetX  Outlet.GetY  Outlet.IncrementVisits | 1. The parameters, X and Y, are the coordinates of a household whose occupant(s) will eat out on the current day  2. Initialise local variable NearestOutlet to zero and declare two floats  3. Initialise NearestOutlet to first outlet in the list (see step 5 below)  4. Loops through each outlet in the Outlets list  5. Calculates the distance from the household to the outlet. This is only along a straight line if the outlet and the household are in the same row or column of the settlement; otherwise, there will be a combination of either up or down with either left or right, along with a 90-degree turn.  6. If the current outlet being examined is closer than the closest found so far, store the index of the current outlet in NearestOutlet  7. After the loop, access the outlet indexed by NearestOutlet and call its IncrementVisits subroutine |
| AlterAvgCost PerMeal  Ⓟ | Parameters: Change: Single  Returns: -  Called From: Simulation.ProcessCostChangeEvent  Calls: - | Accepts a value as a parameter and adds this value to the AvgCostPerMeal attribute |
| AlterDailyCosts  Ⓟ | Parameters: Change: Single  Returns: -  Called From: Simulation.ProcessCostChangeEvent  Calls: - | Accepts a value as a parameter and adds this value to the DailyCosts attribute |
| AlterReputation  Ⓟ | Parameters: Change: Single  Returns: -  Called From: Simulation.ProcessReputationChangeEvent  Calls: - | Accepts a value as a parameter and adds this value to the ReputationScore attribute |

| Subroutine | Data | Description |
| --- | --- | --- |
| AlterFuelCost PerUnit  Ⓟ | Parameters: Change: Single  Returns: -  Called From: Simulation.ProcessCostOfFuelChangeEvent  Calls: - | Accepts a value as a parameter and adds this value to the FuelCostPerUnit attribute |
| Calculate  DeliveryCost  Ⓕ | Parameters: -  Returns: TotalCost: Single  Called From: Company.GetDetails  Company.ProcessDayEnd  Calls: Company.GetListOfOutlets  Company.GetDistanceBetweenTwoOutlets | 1. Calls GetListOfOutlets to receive an integer ArrayList, with one integer for each outlet, beginning with zero  2. Declares an integer, TotalDistance, initialised to zero  3. Loops once per outlet minus 1 (this subroutine is actually not called if a company has only 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 (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 |
| CloseOutlet  Ⓕ | Parameters: ID: Integer  Returns: CloseCompany: Boolean  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: Integer  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 subroutine  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, a message stating that the outlet is now at maximum capacity is output |

| Subroutine | Data | Description |
| --- | --- | --- |
| GetReputationScore  Ⓕ | Parameters: -  Returns: ReputationScore: Single  Called From: Simulation.ProcessDayEnd  Calls: - | Returns the value of ReputationScore | |
| GetDetails  Ⓕ | Parameters: -  Returns: Details: String  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 the Outlets list, which are addressed in turn via a for loop  3. Returns the string – this is essentially a toString subroutine | |
| GetDistanceBetween TwoOutlets  Ⓕ | Parameters: Outlet1: Integer, Outlet2: Integer  Returns: Single  Called From: Company.CalculateDeliveryCost  Calls: - | Returns the distance between two outlets. Each outlet has a grid position, and the distance between outlets is 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. not a straight line unless they share a row or a column. | |
| GetListOfOutlets  Ⓕ | Parameters: -  Returns: Temp: ArrayList  Called From: Company.CalculateDeliveryCost  Calls: - | 1. A new ArrayList object is created  2. A loop iterates through each Outlet in the Outlets list  3. The zero-based index of each outlet is added to the ArrayList  4. The ArrayList is returned | |
| GetName  Ⓕ | Parameters: -  Returns: Name: String  Called From: Simulation.DisplayCompaniesAtDayEnd  Simulation.ProcessCostOfFuelChangeEvent  Simulation.ProcessReputationChangeEvent  Simulation.ProcessCostChangeEvent  Simulation.GetIndexOfCompany  Calls: - | Returns the value of Name | |
| GetNumberOfOutlets  Ⓕ | Parameters: -  Returns: Integer  Called From: Simulation.ModifyCompany  Calls: - | Returns the number of outlets in the Outlets list | |

| Subroutine | Data | Description |
| --- | --- | --- |
| NewDay  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.ProcessDayEnd  Calls: Outlet.NewDay | 1. Loops through each outlet in the Outlets list  2. Calls the NewDay subroutine for each outlet, resetting the number of visits for each to zero | |
| OpenOutlet  Ⓟ | Parameters: X: Integer, Y: Integer  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 variable, set according to company category)  4. The new outlet is added to the Outlets list | |
| ProcessDayEnd  Ⓕ | Parameters: -  Returns: Details: String  Called From: Simulation.DisplayCompaniesAtDayEnd  Calls: Company.CalculateDeliveryCost  Outlet.CalculateDailyProfitLoss | 1. Declares an empty string, Details  2. Declares Single variable to track profit/loss overall and a further variable to track profit/loss for each outlet in turn, as well as one to track delivery costs  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 appended to the Details string and added to the total profit/loss variable  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 | |

### 'Simulation' Class

| Subroutine | Data | Description |
| --- | --- | --- |
| AddCompany  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.Run  Calls: Company.GetReputationScore | 1. Local variables declared to store details of the new company  2. User is prompted for name and starting balance for company  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 sets the new X and Y coordinates  6. New Company object is constructed and added to the Companies list |
| 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 company's name  4. Outputs the return from a call to the ProcessDayEnd subroutine |
| DisplayEvents AtDayEnd  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.ProcessDayEnd  Calls: Simulation.ProcessAddHouseholdEvent  Simulation.ProcessCostOfFuelChangeEvent  Simulation.ProcessReputationChangeEvent  Simulation.ProcessCostChangeEvent | 1. Writes 'Events:' to the console  2. Generates a random float, with 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 ProcessAddHouseholdEvent  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 either they might all happen, none of them will happen, or any combination of events will happen. If none of the events happens, but the initial selection structure from step 2 is entered, 'No events' will not be displayed.** |

| Subroutine | Data | Description |
| --- | --- | --- |
| DisplayCompanies  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.Run  Calls: Company.GetDetails | 1. Text 'details of all companies:' is output to the console  2. Loop initiated to run once per object in the companies ArrayList  3. Call to GetDetails for each company is made |
| DisplayMenu  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.Run  Calls: - | Displays menu items, one on each line, and prompts the user for a choice; this subroutine does not validate input or even store the user's response; user input is handled in run  **NB The menu has no option 5** |
| GetIndexOfCompany  Ⓕ | Parameters: CompanyName: String  Returns: Index: Integer  Called From: Simulation.Run  Calls: Company.GetName | 1. Integer variable declared and set to -1  2. Loops through all companies in turn  3. If a company name matches the passed parameter, return the zero-based index of that company  4. If the loop terminates without a match, return -1 |
| ModifyCompany  Ⓟ | Parameters: Index: Integer  Returns: -  Called From: Simulation.Run  Calls: Company.GetNumberOfOutlets  Company.CloseOutlet  Company.ExpandOutlet  Settlement.GetXSize  Settlement.GetYSize  Company.OpenOutlet | 1. Submenu is displayed, with three 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 6  3. 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  4. If 3 (expand outlet) was entered, a call to ExpandOutlet is made  5. If an out-of-range ID is entered in step 3, display error message  6. If input is 1 (open new outlet), prompt user for X and Y coordinates  7. If the coordinates are within the grid bounds, call OpenOutlet  8. If not, display an error message  **NB No error is displayed if an entry other than 1, 2 or 3 is made.** |

| Subroutine | Data | Description |
| --- | --- | --- |
| ProcessReputation  ChangeEvent  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.DisplayEventsAtDayEnd  Calls: Company.GetName  Company.AlterReputation | 1. Generates a random number, always to a single decimal place, between 0.1 and 0.9  2. Generates a random integer of either 1 or 0 called UpOrDown  3. Generates a random company index  4. If UpOrDown is zero, the randomly chosen company's reputation score goes up by the decimal amount; otherwise, it goes down by that amount  5. Step 4 is implemented by outputting the result and by calling the company's AlterReputation subroutine | |
| ProcessAdd  HouseholdEvent  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.DisplayEventsAtDayEnd  Calls: Settlement.AddHousehold | 1. Generates a random integer between 1 and 4  2. Loops that many times, calling the Settlement class's AddHousehold subroutine, effectively creating that many new Household objects  3. Output to console the number of new households created | |
| ProcessCostOfFuel ChangeEvent  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.DisplayEventsAtDayEnd  Calls: Company.GetName  Company.AlterFuelCostPerUnit | 1. Generates a random number, always to a single decimal place, between 0.1 and 0.9  2. Generates a random integer of either 1 or 0 called UpOrDown  3. Generates a random company index  4. If UpOrDown is zero, the randomly chosen company's fuel cost per unit goes up by the decimal amount; otherwise, it goes down by that amount  5. Step 4 is implemented by outputting the result and by calling the company's AlterFuelCostPerUnit subroutine | |
| ProcessCost  ChangeEvent  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.DisplayEventsAtDayEnd  Calls: Company.GetName  Company.AlterDailyCosts  Company.AlterAvgCostPerMeal | 1. Integer variables CostToChange and UpOrDown are randomly initialised to either zero or one  2. A further integer, CompanyNo, is initialised to a randomly selected index of a company  3. If CostToChange was initialised to zero:  a. Generate a random one-decimal-place number between 0.1 and 1.9  b. If UpOrDown is zero, daily costs go up by the decimal amount  c. Otherwise, they go down by the decimal amount  4. Otherwise, if CostToChange was initialised to one:  a. Generate a random one-decimal-place number between 0.1 and 0.9  b. If UpOrDown is zero, the average cost per meal goes up by the decimal amount  c. Otherwise, it does down by the decimal amount | |

| Subroutine | Data | Description |
| --- | --- | --- |
| ProcessDayEnd  Ⓟ | Parameters: -  Returns: -  Called From: Simulation.Run  Calls: Company.NewDay  Company.GetReputationScore  Settlement.GetNumberOfHouseholds  Settlement.FindOutIfHouseholdEatsOut  Company.AddVisitToNearestOutlet  Simulation.DisplayCompaniesAtDayEnd  Simulation.DisplayEventsAtDayEnd | 1. Loop through each company in the Companies list  2. Call the NewDay subroutine for each company, resetting the number of visits for each company to zero  3. Get each company's reputation score in turn, adding it to a TotalReputation local variable  4. For each company, append the cumulative reputation total to an ArrayList  5. Loop through each household in the simulation  6. For each household, if an occupant will now visit a restaurant (determined using a random number in FindOutIfHouseholdEatsOut), set a local variable to a random integer between zero and the cumulative reputation score  7. Loop through the cumulative reputation ArrayList, incrementing a zero-initialised variable Current with each iteration  8. If the randomly generated number (step 6) is less than the value in the ArrayList at index Current, a visit is added to the company indexed by Current in the Companies list, with the household’s coordinates passed to AddVisitToNearestOutlet | |
| Run  Ⓟ | Parameters: -  Returns: -  Called From: Module1.Main  Calls: Simulation.DisplayMenu  Settlement.DisplayHouseholds  Simulation.DisplayCompanies  Simulation.DetIndexOfCompany  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 (which is in the Settlement class)  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 DetIndexOfCompany. The call to this subroutine will return either the integer index of the company, if it exists, or -1, if it doesn't. If -1 is returned, the user is prompted for a company name until -1 is not returned.  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 (initially 250) occupied by households. 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. |
| Module1 | Contains the Main subroutine to run the program, which entails creating a new Simulation object |

### ‘Household’ Attributes

| Attribute | Type | Default Value | Description |
| --- | --- | --- | --- |
| ChanceEatOutPerDay | Single | Random, between 0 and 1 | Represents the probability that a member of the household will eat out on a given day |
| XCoord | Integer | Passed as a parameter to constructor | The X coordinate of the household's location within the settlement |
| YCoord | Integer | Passed as a parameter to constructor | The Y coordinate of the household's location within the settlement |
| NextID | Integer | Static/shared; initialised to 1 | Unique identifier for each household, with the first household numbered '1', the second '2', and so forth |
| ID | Integer | Current value of NextID | Value in NextID is stored in ID when a Household object is constructed |

### ‘Settlement’ Attributes

*The LargeSettlement class inherits from the Settlement class, but does not include any additional subroutines 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 | Default Value | Description |
| --- | --- | --- | --- |
| StartNoOf Households | Integer | 250 | The number of households in the settlement at the beginning of the simulation |
| XSize | Integer | 1,000 | One dimension of the size of the settlement 'grid'. As it is the 'x' dimension, it might be helpful to visualise this as the settlement’s width, but the program does not store the grid in any form, such as a 2D array. |
| YSize | Integer | 1,000 | The other dimension for the size of the settlement |
| StartNoOf Households | Integer | 250 | The number of households in the settlement at the beginning of the simulation |

### ‘Outlet’ Attributes

| Attribute | Type | Default Value | Description |
| --- | --- | --- | --- |
| VisitsToday | Integer | 0 | The number of times the outlet has been visited on the current day, which is incremented with each visit |
| XCoord | Integer | Passed as a parameter to constructor | The X coordinate of the outlet's location within the settlement |
| YCoord | Integer | Passed as a parameter to constructor | The Y coordinate of the outlet's location within the settlement |
| Capacity | Integer | 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 |
| MaxCapacity | Integer | See 🡪 | The highest value that capacity can be increased to by way of calls to the class's AlterCapacity subroutine. This is initialised to the MaxCapacityBase parameter, with a random integer between 0 and 50 added to it, then another random number in the same range subtracted from it. |
| DailyCosts | Single | 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 seat or potential seat up to MaxCapacityBase (in addition to £0.50 that has already been paid) * £100 one-off |

### ‘Company’ Attributes

| Attribute | Type | Default Value | Description |
| --- | --- | --- | --- |
| Name | String | Passed as a parameter to constructor | The name of the company |
| Category | String | Passed as a parameter to constructor | The type of company – one of 'fast food', 'family' or 'named chef' |
| Balance | Single | Passed as a parameter to constructor | The amount of money the company has |
| ReputationScore | Single | See 🡪 | 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. |
| AvgCostPerMeal | Single | See 🡪 | The average cost of a meal (as bought by the company), set to 5, 12 and 20 for fast-food, family and named chef companies respectively |
| AvgPricePerMeal | Single | See 🡪 | The average price of a meal (as sold to a customer), set to 10, 14 and 40 for fast-food, family and named chef companies respectively |
| DailyCosts | Single | 100 | Part of a company's expenses, regardless of the number of outlets |
| FamilyOutletCost | Single | 1,000 | The cost of a family company opening a new outlet |
| FastFoodOutletCost | Single | 2,000 | The cost of a fast-food company opening a new outlet |
| NamedChefOutletCost | Single | 15,000 | The cost of a named chef company opening a new outlet |
| FuelCostPerUnit | Single | Passed as a parameter to constructor | Used in the calculation of delivery to each outlet |
| BaseCostOfDelivery | Single | Passed as a parameter to constructor | Used in the calculation of delivery to a company, regardless of the number of outlets |
| Outlets | ArrayList of Outlet objects | Empty | Collection of all the company's outlets |
| FamilyFoodOutletCapacity | Integer | 150 | The base maximum capacity for a family outlet |
| FastFoodOutletCapacity | Integer | 200 | The base maximum capacity for a fast-food outlet |
| FamedChefOutletCapacity | Integer | 50 | The base maximum capacity for a named chef outlet |
| Name | String | Passed as a parameter to constructor | The name of the company |

### Simulation Attributes

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

### The class containing the Main subroutine has no attributes