Stephen Russell
EEUE0F

Data Structures & Algorithms
18th November 2012

ICP-2027 ASSIGNMENT ONE

The Task

Simulate and model the behaviour of customers queuing and moving through a multiple checkout line system as seen in a supermarket.

Defining a queue

A queue can be defined as a structure holding a sequence of elements, with both a front and a rear. Addition to the queue can only occur at the rear while removal is strictly from the front. The elements contained in a queue are not required to be unique and are forbidden from being sorted. These strict rules present a first in first out (FIFO) data structure where elements are processed exclusively in the order they presented. A correctly implemented queue can return a time complexity of O*(1)* for the majority of operations.

Choosing an abstract data type

Although a queue seems the obvious data type to tackle the problem of modelling checkout lines, there are other data types to be considered. A set is a collection of distinct values, whose order is insignificant. A checkout line could easily be described as a set if each customer is considered to be unique. This implementation requires that every new addition to the set to be tested against every element currently stored returning a time complexity of O*(n)* where *n* is the number of elements stored. This is an unnecessary operation which can be improved upon. A stack is an example of a last in first out (LIFO) data structure which provides the ability to add and remove elements from one end, the top. As such this is not suitable for representing a checkout line where customers are served in a FIFO order. A list is simply a sequence of elements with a fixed order, it offers the ability to add and remove elements from any location in the list and represents a checkout line quite correctly, if we choose only to add elements at the rear and remove elements from the front. This more specific version of a list is in fact the definition of a queue.

Implementing a queue

A queue can be implemented in two ways, either bounded or unbounded. A bounded queue, as the name suggests, is limited by a predetermined and fixed maximum length. To achieve the best efficiency from a this implementation, the notion of a cyclic array is required where every element including the first and last have a neighbour on both sides (*see fig 1*). The indices of the front and rear elements are then tracked along with the queue length to allow for addition and removal from the correct array locations. Elements are added initially from the array's left most component, each subsequent element added to it's right, becoming the rear of the queue until the maximum length is reached. When removing an element from the queue, the cyclic array along with markers for the first element of the queue has a profound effect on efficiency. If the front of the queue was to remain at array location 0, every element would need to be shuffled accordingly with every removal, producing a time complexity of *O(n)*. However by simply updating the marker of the first queue position when elements are removed, the efficiency increases to the much more appealing *O(1)*. Removing elements in this way allows for both the front and rear of the queue to move relatively through the array components ensuring the maximum length of the queue is constant. Most operations on a queue implemented in this way have an efficiency of 0*(1)* with the exception of clearing the array which maintains an efficiency of 0*(n)*, because every element must be visited in order to clear it. An unbounded queue is unlimited in length and can be represented by a singly linked list with links to both it's front and rear components. Providing a link to the rear most component of the queue negates the need to traverse the entire structure when adding an element and returns a time complexity of O*(1)* for addition. Removing elements from the front of the queue requires the non complex operation of adjusting the link to the front of the queue so that when the element is removed it then links to the second node. This operation, like addition, has a time complexity of 0*(1)* and unlike the bounded implementation most operation performed on the queue has a time complexity of O*(1)* including the clear operation which needs only to remove the links to first and final node in order for the queue to be cleared.

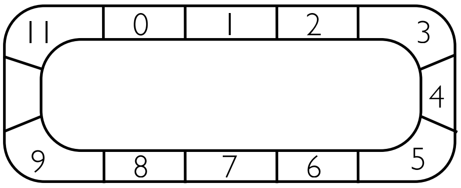


Figure 1. Visualisation of a cyclic array.

Choosing an implementation

Before deciding on a bounded or unbounded implementation it is important to check the Java api for pre existing queue implementations. Java offers an interface called Queue, through the package java.util, which can be implemented using any data structure with the 6 abstract methods: add(), element(), offer(E e), peek(), poll() and remove(). There is however an implementation of the queue interface provided by Java called LinkedList, it is an unbounded implementation using a doubly linked list data structure. I prefer the linked list data structure for this problem, as apposed to an array implementation, because it is not clear in advance what the maximum number of elements needed to be stored will be. Although Java implements arrays, such as java.util.ArrayList, which allow for the array length to be increased at run time. When the array's maximum length is reached there will be a decrease in efficiency while the compiler decides how much to increase the array length by.

Modelling the queue

Before considering the queue model, the contents of the queue must be defined. The queue will hold instances of the object Customer, each Customer is constructed with a QueuingType and a ShoppingBasket which is constructed with a random number of Items between 5 & 99 inclusive. Customers have a random packing speed, if their packing speed is too low they will only join a queue which has a Bagger. A Customers QueuingType defines how to join a queue and is represented by either a 1 or a 0. There is one non accessor method available in Customer called packBags(...) which is discussed later.

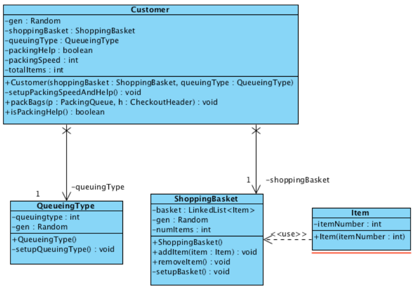

Customer
-gen : Random
-shoppingBasket : ShoppingBasket
-queuingType : QueueingType
-packingHelp : boolean
-packingSpeed : int
-totalItems : int
+Customer(shoppingBasket : ShoppingBasket, queuingType : QueueingType)
-setupPackingSpeedAndHelp() : void
+packBags(p : PackingQueue, h : CheckoutHeader) : void
+isPackingHelp() : boolean
1
-queuingType
1
-shoppingBasket
QueueingType
-queuingtype : int
-gen : Random
+QueueingType()
-setupQueuingType() : void
ShoppingBasket
-basket : LinkedList<Item>
-gen : Random
-numItems : int
+ShoppingBasket()
+addItem(item : Item) : void
+removeItem() : void
-setupBasket() : void
<<use>>
Item
-itemNumber : int
+Item(itemNumber : int)

Figure 2 UML class diagram for Customer with dependancies

A checkout line is modelled in the application by the class Checkout. There are four objects which make up a Checkout: a queue of Customers, a Checker, a Bagger and a PackingQueue. Figure 3 below is a UML class diagram outlining the relationships of these classes.

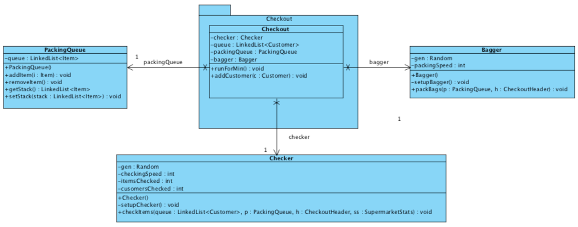

Checkout
Checkout
-checker : Checker
-queue : LinkedList<Customer>
-packingQueue : PackingQueue
-bagger : Bagger
+runForMin() : void
+addCustomer(c : Customer) : void
PackingQueue
-queue : LinkedList<Item>
+PackingQueue()
+addItem(i : Item) : void
+removeItem() : void
+getStack() : LinkedList<Item>
+setStack(stack : LinkedList<Item>) : void
1
packingQueue
bagger
Bagger
-gen : Random
-packingSpeed : int
+Bagger()
-setupBagger() : void
+packBags(p : PackingQueue, h : CheckoutHeader) : void
1
checker
1
Checker
-gen : Random
-checkingSpeed : int
-itemsChecked : int
-cusomersChecked : int
+Checker()
-setupChecker() : void
+checkItems(queue : LinkedList<Customer>, p : PackingQueue, h : CheckoutHeader, ss : SupermarketStats) : void

Figure 2 UML class diagram for Checkout with dependancies

The Checkout constructs the chosen queue implementation, LinkedList, and provides two methods which are not accessors for manipulating the checkout queue. The first method utilises the LinkedList queue method add() and simply provides the ability to add a customer to the end of the checkout queue. The second method provides the ability to run the checkout for one minute and is presented below:

```
public void runForMin() {

    if (queue.size() != 0) {
        checker.checkItems(queue, packingQueue);

        if (bagger != null) {
                bagger.packBags(packingQueue, header);
            }
            if (!packingQueue.getStack().isEmpty()) {
                queue.getFirst().packBags(packingQueue, header);
            }
        }
    }
}
```

This method relies heavily on the classes Checker, Bagger & PackingQueue and in fact passes the responsibility of maintaining the integrity of removing Customers from the queue to the Checker class.

```
public void checkItems(LinkedList<Customer> queue, PackingQueue p) {

    Customer c = queue.getFirst();
    ShoppingBasket b = c.getShoppingBasket();

    for (int i = 0; i < checkingSpeed && !queue.isEmpty(); i++){

        if (!b.getBasket().isEmpty()) {
            p.addItem(b.getBasket().getFirst());
            b.removeItem();
        }
        if (b.getBasket().isEmpty() && p.getPackables().isEmpty()) {
            queue.remove(c);
        }
    }
}
```

A Checker is constructed with a random checking speed between 10 & 30 Items per min and has only one public method which is not an accessor. That method is checkItems(...) and is called by the Checkout once every minute the checkout is run.

This method has a loop controlled by the packing speed of the Checker. For every iteration; if the Customers' ShoppingBasket if it is not empty then move an Item from the ShoppingBasket to the PackingQueue. If both ShoppingBasket and PackingQueue are empty then remove Customer from the queue. Whenever the PackingQueue is not empty both the Customer and Bagger, if there is a Bagger, will remove Items from the PackingQueue until they have reached their limit for that minute. Both objects use the same method when packing called packBags(...).

```
public void packBags(PackingQueue p, CheckoutHeader h) {

    for (int i = 0; i < packingSpeed; i++) {

        if (p.getPackables().size() != 0) {
            p.removeItem();
            h.removeItem();
        }
    }
}
```

The method above removes Item from the PackingQueue which at some point will enable the Customer to be removed from the queue.

Modelling multiple queues

Multiple queues are modelled by the class MultiCheckout. This class is responsible for constructing and maintaining an ArrayList containing instances of the class Checkout described above. MultiCheckout provides methods to add a Bagger to a specified or random Checkout and remove a Bagger from a specified Checkout.

```
public void removeBagger(int checkoutIndex) {
    boolean baggerNeeded = false;
    Checkout ch = getCheckout(checkoutIndex);
    if (ch.getBagger() != null) {
        for (int i = 0; i < ch.getQueue().size(); i++) {
            if (ch.getQueue().get(i).isPackingHelp()) {
                baggerNeeded = true;
            }
        }
        if (!baggerNeeded) {
            ch.setBagger(null);
        }
    }
}
```

The method to remove a Bagger shown above takes into consideration the fact that a Customer requiring a Bagger may have joined the Checkout since the Bagger was added. If such a customer exists in the Checkout queue, the Bagger will not be removed. There are two further methods provided by MultiCheckout, runCheckoutsForMin() which simply calls the Checkout method runForMin() for every Checkout held in the ArrayList and a method called chooseCheckout(...). The chooseCheckout(...) method is responsible for deciding which Checkout a Customer will join, a Customer with a PackingType of 0 will join the queue with least number of total Items however a Customer with a PackingType of 1 will join the queue with the least number of Customers. chooseCheckout(...) is assisted by the private method getQueueScore(...) which simply returns an integer value for a Checkout based on Items or Customers. chooseCheckout is detailed below.

```
public void chooseCheckout(Customer c) {
    int chosenIndex = 0;
    int queueScore = getQueueScore(getCheckout(0), c);
    for (int i = 0; i < numCheckouts; i++) {
        if (c.isPackingHelp()) {
            if (getCheckout(i).getBagger() != null) {
                int a;
                if (queueScore > (a = getQueueScore(getCheckout(i), c))) {
                    queueScore = a;
                    chosenIndex = i;
                }
            }
        } else {
            int a;
            if (queueScore > (a = getQueueScore(getCheckout(i), c))) {
                queueScore = getQueueScore(getCheckout(i), c);
                chosenIndex = i;
            }
        }
    }
    getCheckout(chosenIndex).addCustomer(c);
}
```

Visualising a checkout line

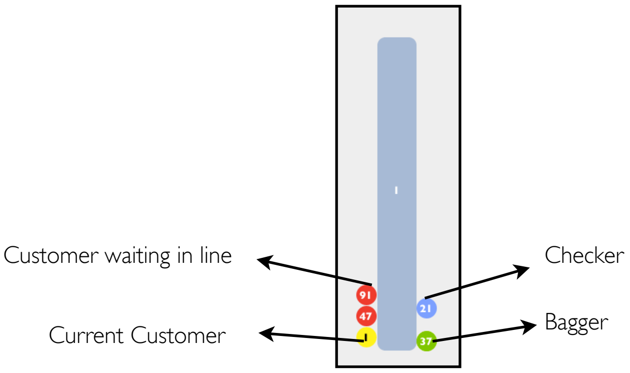


Figure 3. Checkout Visualisation

A Checkout line is visualised by the class VisualCheckout, it constructs an instance of the Checkout class for use at runtime. Based on the current status of all the classes instantiated by Checkout, VisualCheckout draws components representing the state.

Visualising multiple checkout lines

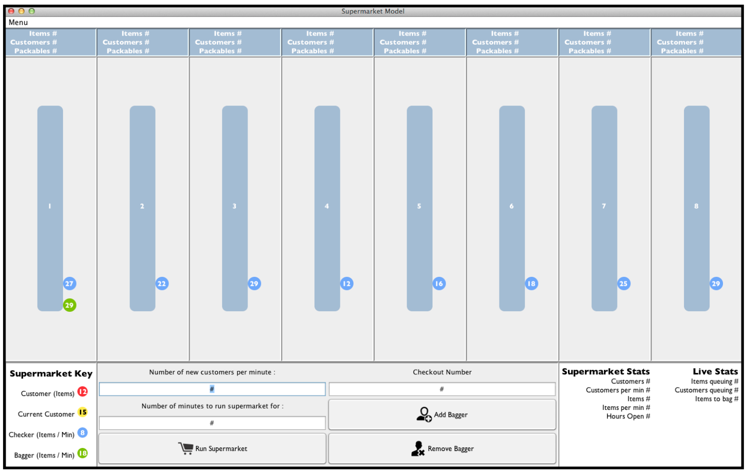


Figure 4. Program output on opening

The class Supermarket utilises an instance of MultiCheckout and provides an interface for the model to be visualised. The methods of Supermarket provide the functionality of the buttons seen in section five below. The method runSuperMarketForMins(int i) will run the Supermarket for the number of minutes indicated by the int supplied while addCustomers(int numCustomers) will add the specified number of Customers to the Supermarket, each choosing it's own queue to join. The MultiCheckout methods for adding and removing Baggers are implemented here also.

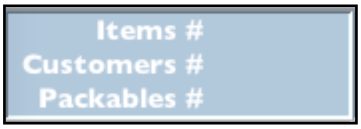
The Interface presented in *Figure 4* can be split into six distinct sections:

1. Menu bar

The menu bar simply offer the user an exit button.

2. Checkout statistics panel

This panel displays the number of Customers and Item currently in the queue as well as the size of the PackingQueue

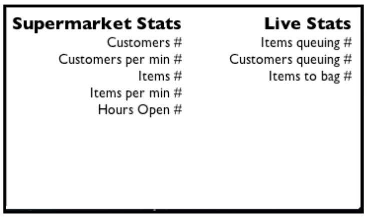


3. Checkout visualisation

As described in *Figure 3*.

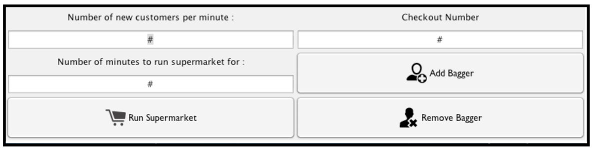
4. Supermarket statistics panel

This panel is split into two sections. Supermarket Stats shows the overall total number of Customers and Items precessed along with hour open and average number of Customers and Items processed per minute. Live Stats shows the current total Customers, Items and PackingQueue size.



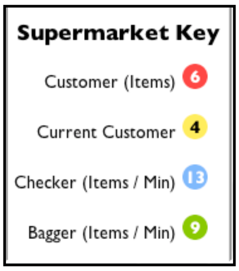
5. Supermarket control panel

Run Supermarket will animate the model with the number of customers and minutes entered. Add Bagger will add to a random queue unless a checkout number is specified.


Number of new customers per minute :
#
Checkout Number
#
Number of minutes to run supermarket for :
#
Add Bagger
Run Supermarket
Remove Bagger

6.Supermarket key panel

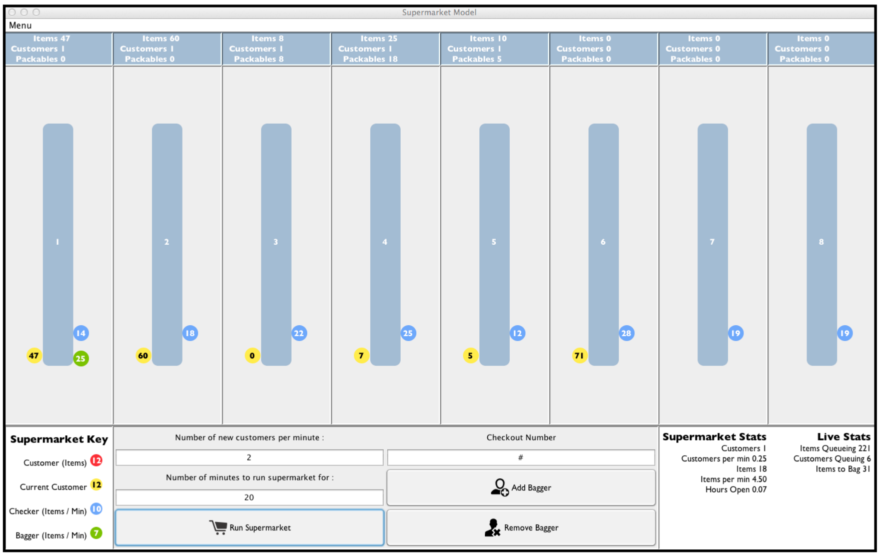
This panel offers a quick reference to users of the various elements displayed in the program.



Program screen shots

The following is a sequence of screen shots designed to show the program running and validate correctness. Capturing the animation is a difficult procedure and because of this there is also a program demonstration available at : https://vimeo.com/54027717

Values of 2 and 20 are supplied for the number of customers per minute and number of minute to run respectively then Run Supermarket is selected. Add bagger is then selected first without, and then with a checkout number selected.



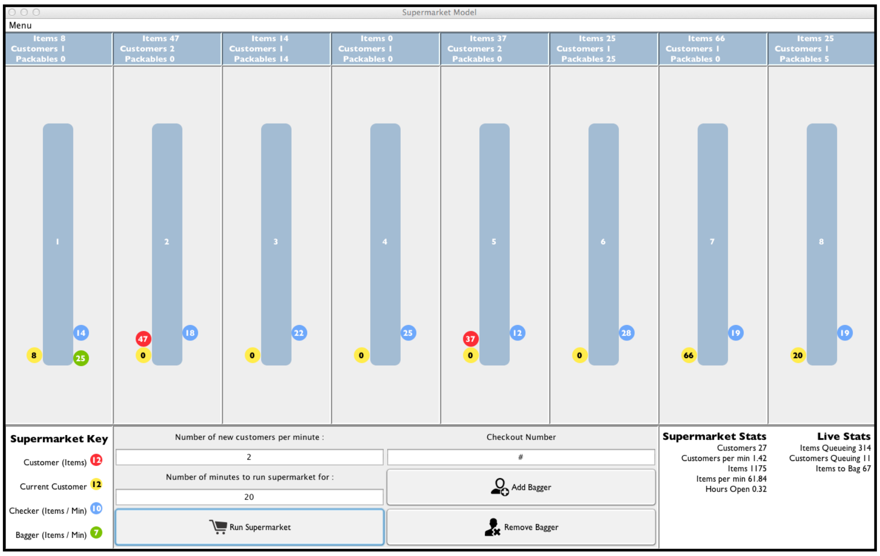
Items 47 Customers 1 Packables 0	Items 60 Customers 1 Packables 0	Items 8 Customers 1 Packables 8	Items 25 Customers 1 Packables 18	Items 10 Customers 1 Packables 5	Items 0 Customers 0 Packables 0	Items 0 Customers 0 Packables 0	Items 0 Customers 0 Packables 0

Supermarket Key
Customer (Items) 12
Current Customer 12
Checker (Items / Min) 10
Bagger (Items / Min) 7

Number of new customers per minute : 2
Number of minutes to run supermarket for : 20
Checkout Number #

Supermarket Stats
Customers 1
Customers per min 0.25
Items 18
Items per min 4.50
Hours Open 0.07

Live Stats
Items Queueing 221
Customers Queuing 6
Items to Bag 31



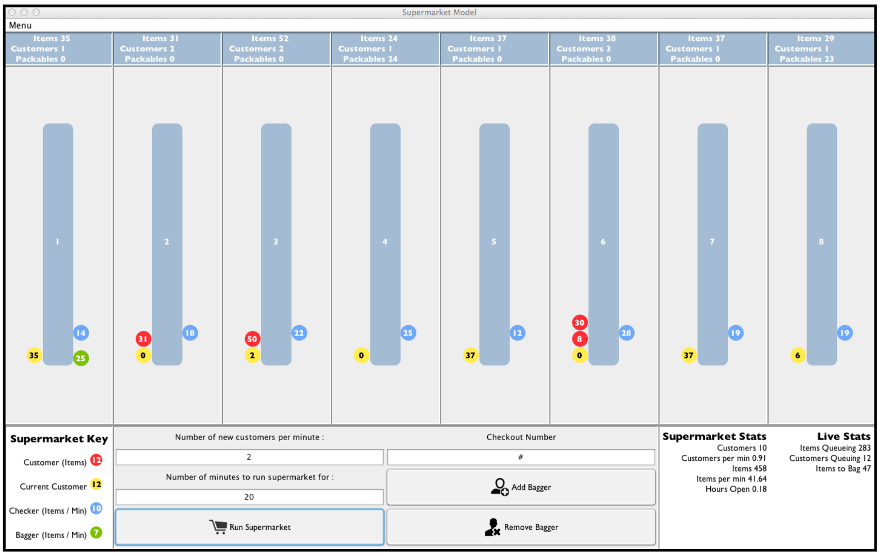
Items 8 Customers 1 Packables 0	Items 47 Customers 2 Packables 0	Items 14 Customers 1 Packables 14	Items 0 Customers 1 Packables 0	Items 37 Customers 2 Packables 0	Items 25 Customers 1 Packables 25	Items 66 Customers 1 Packables 0	Items 25 Customers 1 Packables 5

Supermarket Key
Customer (Items) 12
Current Customer 12
Checker (Items / Min) 10
Bagger (Items / Min) 7

Number of new customers per minute : 2
Number of minutes to run supermarket for : 20
Checkout Number #

Supermarket Stats
Customers 27
Customers per min 1.42
Items 1175
Items per min 61.84
Hours Open 0.32

Live Stats
Items Queueing 314
Customers Queuing 11
Items to Bag 67



Items 35 Customers 1 Packables 0	Items 31 Customers 2 Packables 0	Items 52 Customers 2 Packables 0	Items 24 Customers 1 Packables 24	Items 37 Customers 1 Packables 0	Items 38 Customers 3 Packables 0	Items 37 Customers 1 Packables 0	Items 29 Customers 1 Packables 23

Supermarket Key

Customer (Items)

Current Customer

Checker (Items / Min)

Bagger (Items / Min)

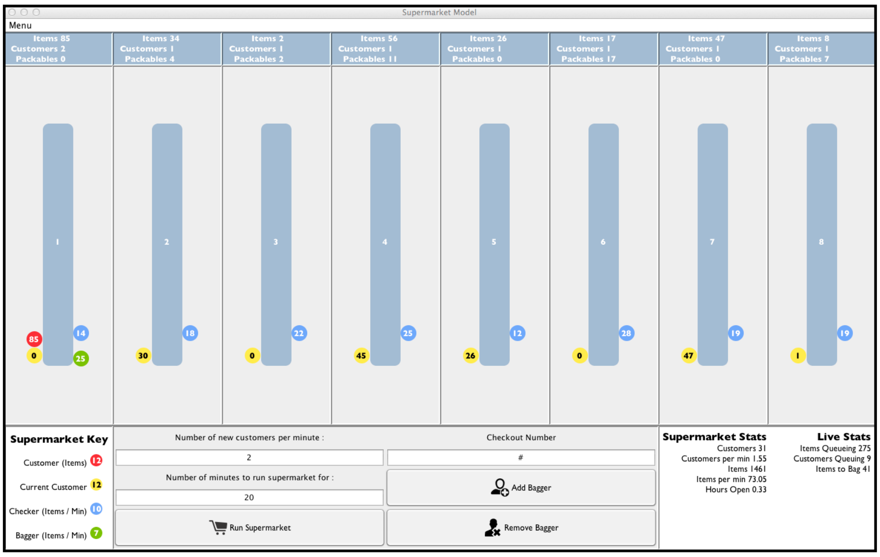
Number of new customers per minute : 2

Number of minutes to run supermarket for : 20

Checkout Number #

Supermarket Stats
Customers 10
Customers per min 0.91
Items 458
Items per min 41.64
Hours Open 0.18

Live Stats
Items Queueing 283
Customers Queuing 12
Items to Bag 47



Items 85 Customers 2 Packables 0	Items 34 Customers 1 Packables 4	Items 2 Customers 1 Packables 2	Items 56 Customers 1 Packables 11	Items 26 Customers 1 Packables 0	Items 17 Customers 1 Packables 17	Items 47 Customers 1 Packables 0	Items 8 Customers 1 Packables 7

Supermarket Key

Customer (Items)

Current Customer

Checker (Items / Min)

Bagger (Items / Min)

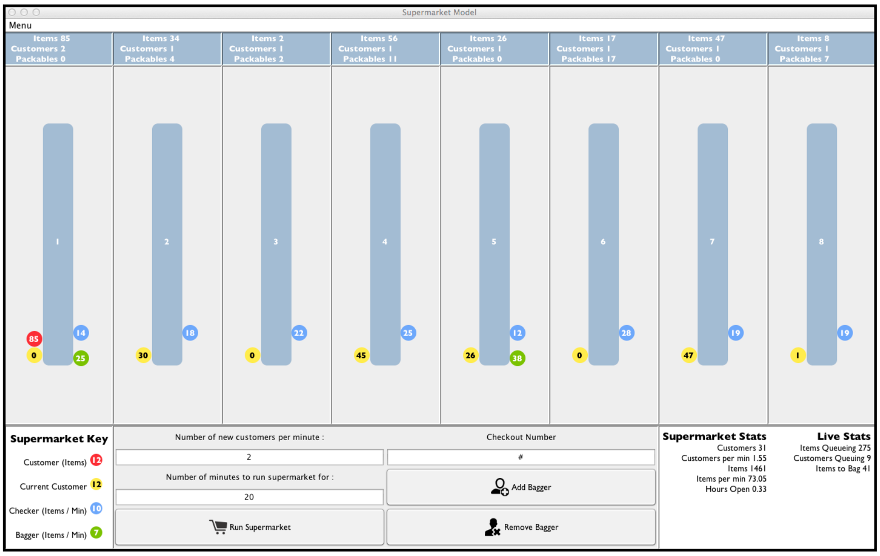
Number of new customers per minute : 2

Number of minutes to run supermarket for : 20

Checkout Number #

Supermarket Stats
Customers 31
Customers per min 1.55
Items 1461
Items per min 73.05
Hours Open 0.33

Live Stats
Items Queueing 275
Customers Queuing 9
Items to Bag 41



Items 85 Customers 2 Packables 0	Items 34 Customers 1 Packables 4	Items 2 Customers 1 Packables 2	Items 56 Customers 1 Packables 11	Items 26 Customers 1 Packables 0	Items 17 Customers 1 Packables 17	Items 47 Customers 1 Packables 0	Items 8 Customers 1 Packables 7

Supermarket Key

Customer (Items)

Current Customer

Checker (Items / Min)

Bagger (Items / Min)

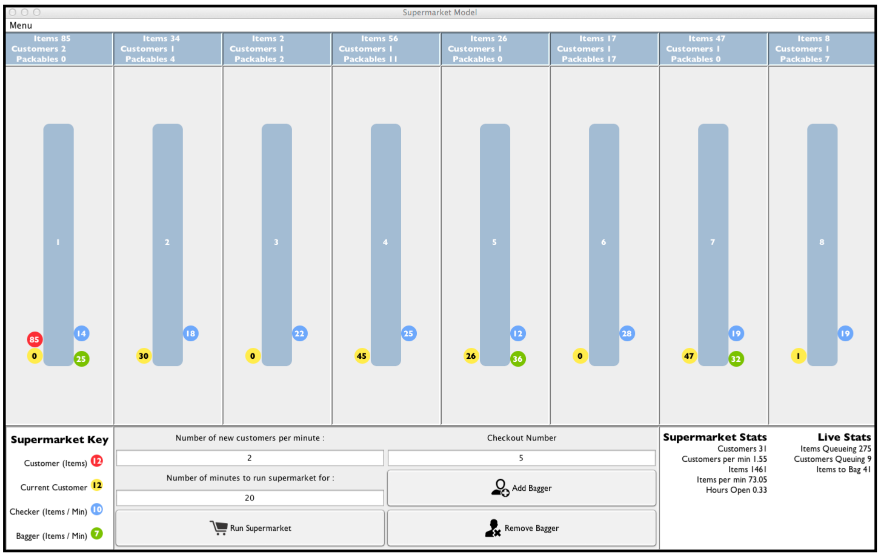
Number of new customers per minute : 2

Number of minutes to run supermarket for : 20

Checkout Number #

Supermarket Stats
Customers 31
Customers per min 1.55
Items 1461
Items per min 73.05
Hours Open 0.33

Live Stats
Items Queueing 275
Customers Queuing 9
Items to Bag 41



Items 85	Items 34	Items 2	Items 56	Items 26	Items 17	Items 47	Items 8
Customers 2	Customers 1	Customers 1	Customers 1	Customers 1	Customers 1	Customers 1	Customers 1
Packables 0	Packables 4	Packables 2	Packables 11	Packables 0	Packables 17	Packables 0	Packables 7

Supermarket Key

Customer (Items) 12
Current Customer 12
Checker (Items / Min) 10
Bagger (Items / Min) 7

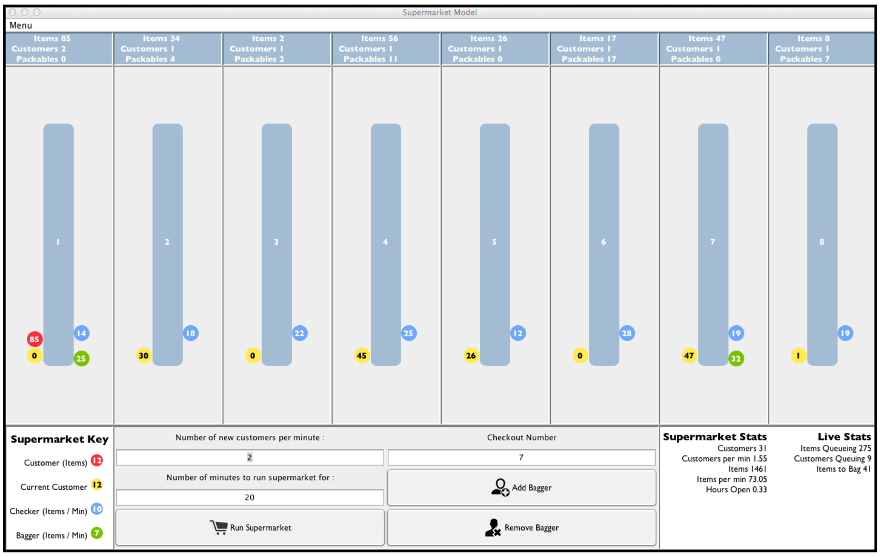
Number of new customers per minute : 2

Checkout Number 5

Number of minutes to run supermarket for : 20

Supermarket Stats
Customers 31
Customers per min 1.55
Items 1461
Items per min 73.05
Hours Open 0.33

Live Stats
Items Queueing 275
Customers Queuing 9
Items to Bag 41



Items 85	Items 34	Items 2	Items 56	Items 26	Items 17	Items 47	Items 8
Customers 2	Customers 1	Customers 1	Customers 1	Customers 1	Customers 1	Customers 1	Customers 1
Packables 0	Packables 4	Packables 2	Packables 11	Packables 0	Packables 17	Packables 0	Packables 7

Supermarket Key

Customer (Items) 12
Current Customer 12
Checker (Items / Min) 10
Bagger (Items / Min) 7

Number of new customers per minute : 2

Checkout Number 7

Number of minutes to run supermarket for : 20

Supermarket Stats
Customers 31
Customers per min 1.55
Items 1461
Items per min 73.05
Hours Open 0.33

Live Stats
Items Queueing 275
Customers Queuing 9
Items to Bag 41