Pokémon Alfresco Restaurant

Initialising

```
var WINDOWBORDERSIZE = 10;
var HUGE = 999999; //Sometimes useful when testing for big or small numbers
var animationDelay = 200; //controls simulation and transition speed
var isRunning = false; // used in simStep and toggleSimStep
var surface; // Set in the redrawWindow function. It is the D3 selection of the svg drawing surface
var simTimer; // Set in the initialization function

//The drawing surface will be divided into logical cells
var maxCols = 40;
var maxRows = 40;

var cellWidth; //cellWidth is calculated in the redrawWindow function
var cellHeight; //cellHeight is calculated in the redrawWindow function

//You are free to change images to suit your purpose. These images came from icons-land.com.
// The copyright rules for icons-land.com require a backlink on any page where they appear.
// See the credits element on the html page for an example of how to comply with this rule.

const urlSquirtle = "images/squirtle.png";
const urlPikachu = "images/pikachu.png";
const urlCharmander = "images/charmander.png";
const urlEevee = "images/eeve.png";
const urlBulbasaur = "images/bulbasaur.png";

var characters = ["squirtle","pikachu","charmander","bulbasaur","eevee"];
```

Initialising - States

```
187
188  const UNORDERED = 0;
189  const WAITING = 1;
190  const STAGING = 2;
191  const ORDERING = 3;
192  const ORDERED = 4;
193  const ORDERING2 = 5;
194  const EATING = 6;
195  const EATEN = 7;
196  const EXITED = 8;
197  const REJECTED = 9;
198
```

These are the states the customers go through

```
198
199  // The cashier can be either BUSY treating a customer, or IDLE, waiting for a customer
200  const IDLE = 0;
201  const BUSY = 1;
202
203  // There are two types of staticmembers in our system: cashiers and entrances
204
205  const CASHIER = 0;
206  const ENTRANCE = 1;
207  const DRINKMACHINE = 2;
208
```

States of Cashier

Fixed Parameters

```
358
359  // The probability of a customer arrival needs to be less than the probability of a departure, else an infinite queue will build.
360  // You also need to allow travel time for customers to move from their seat in the waiting room to get close to the cashier.
361  // So don't set probDeparture too close to probArrival.
362  var probArrival = 0.2;
363  var probOrdered = 0.9;
364  var probEaten = 0.01;
365  var probNoDrinks = 0.6;
366  var probDrinks = 0.9;
367  var probEntrycondition = 1;
368
```

We fixed our parameter throughout all simulations that we ran

probArrival is the time between arriving customers
probOrdered is the time taken to order
probEaten is time taken to eat
probNoDrinks is percentage of customers who order drinks
probDrinks is time taken to collect drinks

Initialising - Tables

```
236   const TABLE1 = 0;
237   const TABLE2 = 1;
238   const TABLE3 = 2;
239   const TABLE4 = 3;
240   const TABLE5 = 4;
241   const TABLE6 = 5;
```

Our base condition consists of 6 tables

```
248 ▼ var tablesIN = [
249     {"type":TABLE1,"label":"Table1","location":{"row":tableRow_1,"col":tableCol_1},"state":IDLE},
250     {"type":TABLE2,"label":"Table2","location":{"row":tableRow_2,"col":tableCol_2},"state":IDLE},
251     {"type":TABLE3,"label":"Table3","location":{"row":tableRow_3,"col":tableCol_3},"state":IDLE},
252     {"type":TABLE4,"label":"Table4","location":{"row":tableRow_4,"col":tableCol_4},"state":IDLE},
253     {"type":TABLE5,"label":"Table5","location":{"row":tableRow_5,"col":tableCol_5},"state":IDLE},
254     {"type":TABLE6,"label":"Table6","location":{"row":tableRow_6,"col":tableCol_6},"state":IDLE},
```

Initialising our tables

Initialising - Chairs

```
315 var chairsIN = [
316   {"type":CHAIR1a,"label":"Chair1a","location":{"row":chairRow_1a,"col":chairCol_1a},"state":IDLE},
317   {"type":CHAIR1b,"label":"Chair1b","location":{"row":chairRow_1b,"col":chairCol_1b},"state":IDLE},
318   {"type":CHAIR1c,"label":"Chair1c","location":{"row":chairRow_1c,"col":chairCol_1c},"state":IDLE},
319   {"type":CHAIR1d,"label":"Chair1d","location":{"row":chairRow_1d,"col":chairCol_1d},"state":IDLE},
320   {"type":CHAIR2a,"label":"Chair2a","location":{"row":chairRow_2a,"col":chairCol_2a},"state":IDLE},
321   {"type":CHAIR2b,"label":"Chair2b","location":{"row":chairRow_2b,"col":chairCol_2b},"state":IDLE},
322   {"type":CHAIR2c,"label":"Chair2c","location":{"row":chairRow_2c,"col":chairCol_2c},"state":IDLE},
323   {"type":CHAIR2d,"label":"Chair2d","location":{"row":chairRow_2d,"col":chairCol_2d},"state":IDLE},
324   {"type":CHAIR3a,"label":"Chair3a","location":{"row":chairRow_3a,"col":chairCol_3a},"state":IDLE},
325   {"type":CHAIR3b,"label":"Chair3b","location":{"row":chairRow_3b,"col":chairCol_3b},"state":IDLE},
326   {"type":CHAIR3c,"label":"Chair3c","location":{"row":chairRow_3c,"col":chairCol_3c},"state":IDLE},
327   {"type":CHAIR3d,"label":"Chair3d","location":{"row":chairRow_3d,"col":chairCol_3d},"state":IDLE},
328   {"type":CHAIR4a,"label":"Chair4a","location":{"row":chairRow_4a,"col":chairCol_4a},"state":IDLE},
329   {"type":CHAIR4b,"label":"Chair4b","location":{"row":chairRow_4b,"col":chairCol_4b},"state":IDLE},
330   {"type":CHAIR4c,"label":"Chair4c","location":{"row":chairRow_4c,"col":chairCol_4c},"state":IDLE},
331   {"type":CHAIR4d,"label":"Chair4d","location":{"row":chairRow_4d,"col":chairCol_4d},"state":IDLE},
332   {"type":CHAIR5a,"label":"Chair5a","location":{"row":chairRow_5a,"col":chairCol_5a},"state":IDLE},
333   {"type":CHAIR5b,"label":"Chair5b","location":{"row":chairRow_5b,"col":chairCol_5b},"state":IDLE},
334   {"type":CHAIR5c,"label":"Chair5c","location":{"row":chairRow_5c,"col":chairCol_5c},"state":IDLE},
335   {"type":CHAIR5d,"label":"Chair5d","location":{"row":chairRow_5d,"col":chairCol_5d},"state":IDLE},
336   {"type":CHAIR6a,"label":"Chair6a","location":{"row":chairRow_6a,"col":chairCol_6a},"state":IDLE},
337   {"type":CHAIR6b,"label":"Chair6b","location":{"row":chairRow_6b,"col":chairCol_6b},"state":IDLE},
338   {"type":CHAIR6c,"label":"Chair6c","location":{"row":chairRow_6c,"col":chairCol_6c},"state":IDLE},
339   {"type":CHAIR6d,"label":"Chair6d","location":{"row":chairRow_6d,"col":chairCol_6d},"state":IDLE},
```

Initialising our chairs

Waiting Spaces

```
379  // declarations of waiting room
380  var EMPTY = 0;
381  var OCCUPIED = 1;
382
```

Conditions that check for available seats

```
220  var areas =[
221   {"label":"Waiting Area","startRow":cashierRow,"numRows":1,"startCol":26,"numCols":8,"color":"pink"},
222   {"label":"Staging Area","startRow":cashierRow,"numRows":1,"startCol":cashierCol-2,"numCols":1,"color":"red"},
223   {"label":"Drinks Area","startRow":drinkdispenserRow,"numRows":1,"startCol":drinkdispenserCol-5,"numCols":5,"color":"blue"},
224   {"label":"Ordering Area","startRow":cashierRow,"numRows":1,"startCol":cashierCol-1,"numCols":1,"color":"white"}
225  ];
226  var waitingRoom = areas[0]; // the waiting room is the first element of the areas array
```

Statistics

We used 3 statistics to conduct our simulation

```
228  var currentTime = 0;
229  var statistics = [
230  {"name":"Average time spent in restaurant by Customer: ","location":{"row":15,"col":1},"cumulativeValue":0,"count":0},
231  {"name":"Average time spent in queue by Customer: ","location":{"row":16,"col":1},"cumulativeValue":0,"count":0},
232  {"name":"Average percentage of rejected Customers: ","location":{"row":17,"col":1},"cumulativeValue":0,"count":0}
233  ];
234
```

Update surface - Customers

```
485 function updateSurface(){
486     // This function is used to create or update most of the svg elements on the drawing surface.
487     // See the function removeDynamicAgents() for how we remove svg elements
488
489     //Select all svg elements of class "customer" and map it to the data list called
490     var allcustomers = surface.selectAll(".customer").data(customers);
491
492     // If the list of svg elements is longer than the data list, the excess elements are in the .exit() list
493     // Excess elements need to be removed:
494     allcustomers.exit().remove(); //remove all svg elements associated with entries that are no longer in the data list
495     // (This remove function is needed when we resize the window and re-initialize the customers array)
496
497     // If the list of svg elements is shorter than the data list, the new elements are in the .enter() list.
498     // The first time this is called, all the elements of data will be in the .enter() list.
499     // Create an svg group ("g") for each new entry in the data list; give it class "customer"
500     var newcustomers = allcustomers.enter().append("g").attr("class","customer");
501     //Append an image element to each new customer svg group, position it according to the location data, and size it to fill a cell
502     // Also note that we can choose a different image to represent the customer based on the customer type
503     newcustomers.append("svg:image")
504      .attr("x",function(d){var cell= getLocationCell(d.location); return cell.x+"px";})
505      .attr("y",function(d){var cell= getLocationCell(d.location); return cell.y+"px";})
506      .attr("width", Math.min(cellWidth,cellHeight)+"px")
507      .attr("height", Math.min(cellWidth,cellHeight)+"px")
508      .attr("xlink:href",function(d){if (d.character=="squirtle") return urlSquirtle; else if (d.character == "pikachu") return urlPikachu; else if (
509
```

Update surface – Static Members

```
526  //Select all svg elements of class "staticmember" and map it to the data list called staticmembers
527  var allstaticmembers = surface.selectAll(".staticmember").data(staticmembers);
528  //This is not a dynamic class of agents so we only need to set the svg elements for the entering data elements.
529  // We don't need to worry about updating these agents or removing them
530  // Create an svg group ("g") for each new entry in the data list; give it class "staticmember"
531  var newstaticmembers = allstaticmembers.enter().append("g").attr("class","staticmember");
532  newstaticmembers.append("svg:image")
533   .attr("x",function(d){var cell= getLocationCell(d.location); return cell.x+"px";})
534   .attr("y",function(d){var cell= getLocationCell(d.location); return cell.y+"px";})
535   .attr("width", Math.min(cellWidth,cellHeight)+"px")
536   .attr("height", Math.min(cellWidth,cellHeight)+"px")
537   .attr("xlink:href",function(d){if (d.type==CASHIER) return urlCashier; if (d.type==DRINKMACHINE) return urlDrinksdispenser; if (d.type == ENTRAN
538
539  // It would be nice to label the staticmembers, so we add a text element to each new staticmember group
540  newstaticmembers.append("text")
541    .attr("x", function(d) { var cell= getLocationCell(d.location); return (cell.x+cellWidth)+"px"; })
542    .attr("y", function(d) { var cell= getLocationCell(d.location); return (cell.y+cellHeight/2)+"px"; })
543    .attr("dy", ".35em")
544    .text(function(d) { return d.label; });
545
```

Static Members are the cashier, drinks dispenser and entrance

Update surface - Statistics

```
550   var allstatistics = surface.selectAll(".statistics").data(statistics);
551   var newstatistics = allstatistics.enter().append("g").attr("class","statistics");
552   // For each new statistic group created we append a text label
553   newstatistics.append("text")
554   .attr("x", function(d) { var cell= getLocationCell(d.location); return (cell.x+cellWidth)+"px"; })
555     .attr("y", function(d) { var cell= getLocationCell(d.location); return (cell.y+cellHeight/2)+"px"; })
556     .attr("dy", ".35em")
557     .text("");
558
559   // The data in the statistics array are always being updated.
560   // So, here we update the text in the labels with the updated information.
561   allstatistics.selectAll("text").text(function(d) {
562     var avgLengthOfStay = d.cumulativeValue/(Math.max(1,d.count)); // cumulativeValue and count for each statistic are always changing
563     return d.name+avgLengthOfStay.toFixed(1); }); //The toFixed() function sets the number of decimal places to display
564
```

Update surface – Tables & Chairs

```
576     var alltables = surface.selectAll(".tables").data(tablesIN);
577     //This is not a dynamic class of agents so we only need to set the svg elements for the entering data elements.
578     // We don't need to worry about updating these agents or removing them
579     // Create an svg group ("g") for each new entry in the data list; give it class "staticmember"
580     var newtables = alltables.enter().append("g").attr("class","tables");
581
582     newtables.append("svg:image")
583      .attr("x",function(d){var cell= getLocationCell(d.location); return cell.x+"px";})
584      .attr("y",function(d){var cell= getLocationCell(d.location); return cell.y+"px";})
585      .attr("width", Math.min(cellWidth *2,cellHeight*2)+"px")
586      .attr("height", Math.min(cellWidth *2,cellHeight*2)+"px")
587      .attr("xlink:href",function(d){return urlTable});
588
589      ////////////// Chairs
590     var allchairs = surface.selectAll(".chairs").data(chairsIN);
591     var newchairs = allchairs.enter().append("g").attr("class","chairs");
592
593     newchairs.append("svg:image")
594      .attr("x",function(d){var cell= getLocationCell(d.location); return cell.x+"px";})
595      .attr("y",function(d){var cell= getLocationCell(d.location); return cell.y+"px";})
596      .attr("width", Math.min(cellWidth,cellHeight)+"px")
597      .attr("height", Math.min(cellWidth,cellHeight)+"px")
598      .attr("xlink:href", urlChair);
599
600 }
601
```

Creating the images of the chairs and tables

Adding Customers

```
603 function addDynamicAgents(){
604   // customers are dynamic agents: they enter the clinic, wait, get EATING, and then leave
605   // We have entering customers of two types "A" and "B"
606   // We could specify their probabilities of arrival in any simulation step separately
607   // Or we could specify a probability of arrival of all customers and then specify the probability of a Type A arrival.
608   // We have done the latter. probArrival is probability of arrival a customer and probTypeA is the probability of a type A customer who arrives.
609   // First see if a customer arrives in this sim step.
610   if (Math.random()< probArrival){
611     var newcustomer = {"id":1,"type":"A","location":{"row":25,"col":20}, "seatNum":null, "character":'bulbasaur',
612     "target":{"row":entranceRow,"col": entranceCol},"state":UNORDERED,"timeAdmitted":0};
613     if (Math.random()<probEntrycondition) newcustomer.type = "A";
614     //else newcustomer.type = "B";
615
616     var characterNum = Math.floor(Math.random() * characters.length);
617     newcustomer.character = characters[characterNum];
618
619     customers.push(newcustomer);
620   }
621
622 }
```

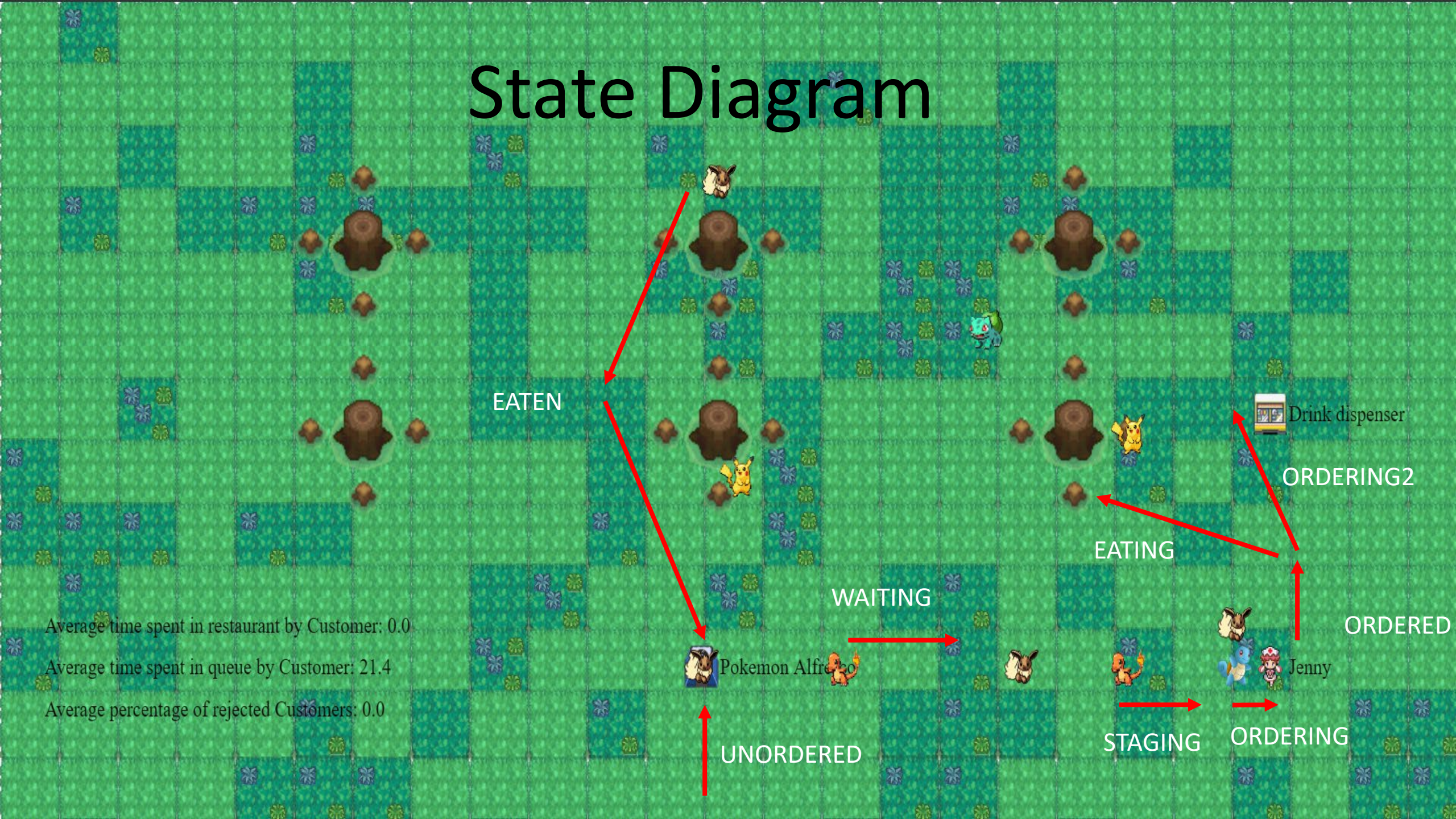
Traits of the customer that enters the restaurant

Character randomiser

Updating Customers

```
624 ▼ function updateCustomer(customerIndex){
625     //customerIndex is an index into the customers data array
626     customerIndex = Number(customerIndex); //it seems customerIndex was coming in as a string
627     var customer = customers[customerIndex];
628     // get the current location of the customer
629     var row = customer.location.row;
630     var col = customer.location.col;
631     var type = customer.type;
632     var state = customer.state;
633
634
635
636
637     // determine if customer has arrived at destination
638     var hasArrived = (Math.abs(customer.target.row-row)+Math.abs(customer.target.col-col))==0;
639
```

State Diagram



States - UNORDERED

```
654     // Behavior of customer depends on his or her state
655     switch(state){
656       case UNORDERED:
657         if (hasArrived){
658           customer.timeAdmitted = currentTime;
659           statistics[2].count++; // number of customers who have arrived at entrance
660           // pick a random spot in the waiting area to queue
661           var emptySeats = waitingSeats.filter(function(d){return d.state==EMPTY;});
662
663           //console.log("Stat1: " + statistics[1].count);
664           //console.log("Stat2: " + statistics[2].cumulativeValue);
665
666           var customersInRestaurant = statistics[2].count - statistics[0].count - (statistics[2].cumulativeValue/100);
667           //console.log(customersInRestaurant);
668           if (customersInRestaurant < 17){
669             //console.log(queueSeats);
670             customer.state = WAITING;
671             var emptySeat = emptySeats[emptySeats.length-1];
672             //var emptySeatNum = Math.max.apply(Math, emptySeats.map(function(o) { return o.waitingseatnum; }))
673             //var emptySeat = emptySeats.filter(function(d){return d.waitingseatnum==emptySeatNum;});
674             emptySeat.state=OCCUPIED;
675             customer.target.row = emptySeat.row;
676             customer.target.col = emptySeat.col;
677             // entrance assigns a sequence number to each admitted customer to govern order of treatment
678             customer.id = ++nextcustomerID_A;
679             queueSeats = queueSeats - 1;
680             //console.log(statistics[1].count);
681
682           } else {
683             // There are no empty seats. We must reject this customer.
684             customer.state = REJECTED;
685             customer.target.row = 20;
686             customer.target.col = 17;
687             statistics[2].cumulativeValue = (statistics[2].cumulativeValue + 1*100); // count of rejected customers in percentage terms
688             var percentagerejected = statistics[2].cumulativeValue/statistics[2].count;
689             //console.log(percentagerejected);
690
691           }
692         }
693
694       break;
```

Rejection criteria: If restaurant seats are filled

Rejection of customers when restaurant is full

States - WAITING

```
675     case WAITING:
676       var emptySeatRow = 0;
677       var emptySeatCol = 0;
678       switch (type){
679
680         case "A":
681           if (customer.id == nextorderingcustomerID_A){
682             emptySeatRow = customer.target.row
683             emptySeatCol = customer.target.col
684             customer.target.row = cashierRow;
685             customer.target.col = cashierCol-2;
686             customer.state = STAGING;
687           }
688           else if (customer.id == nextorderingcustomerID_A+1){
689             emptySeatRow = customer.target.row
690             emptySeatCol = customer.target.col
691             customer.target.row = cashierRow;
692             customer.target.col = cashierCol-3;
693             }
694           else if (customer.id == nextorderingcustomerID_A+2){
695             emptySeatRow = customer.target.row
696             emptySeatCol = customer.target.col
697             customer.target.row = cashierRow;
698             customer.target.col = cashierCol-4;
699             }
700           else if (customer.id == nextorderingcustomerID_A+3){
701             emptySeatRow = customer.target.row
702             emptySeatCol = customer.target.col
703             customer.target.row = cashierRow;
704             customer.target.col = cashierCol-5;
705             }
706           else if (customer.id == nextorderingcustomerID_A+4){
707             emptySeatRow = customer.target.row
708             emptySeatCol = customer.target.col
709             customer.target.row = cashierRow;
710             customer.target.col = cashierCol-6;
711             }
```

When the seat in front of the customer in queue is taken, the customer behind moves forward to occupy it. They are served first-come-first-serve basis.

This is to simulate an ‘infinite’ queue. Customers only queue when there are empty seats in the restaurant. Else they are rejected.

States - WAITING

```
706         else if (customer.id == nextorderingcustomerID_A+4){
707           emptySeatRow = customer.target.row
708           emptySeatCol = customer.target.col
709           customer.target.row = cashierRow;
710           customer.target.col = cashierCol-6;
711           }
712         else if (customer.id == nextorderingcustomerID_A+5){
713           emptySeatRow = customer.target.row
714           emptySeatCol = customer.target.col
715           customer.target.row = cashierRow;
716           customer.target.col = cashierCol-7;
717           }
718         else if (customer.id == nextorderingcustomerID_A+6){
719           emptySeatRow = customer.target.row
720           emptySeatCol = customer.target.col
721           customer.target.row = cashierRow;
722           customer.target.col = cashierCol-8;
723           }
724         else if (customer.id == nextorderingcustomerID_A+7){
725           emptySeatRow = customer.target.row
726           emptySeatCol = customer.target.col
727           customer.target.row = cashierRow;
728           customer.target.col = cashierCol-9;
729           }
730
731
732
733       break;
734
735       }
736       //create
737       var newEmptySeat = waitingSeats.filter(function(d){return d.row == emptySeatRow && d.col == emptySeatCol})
738       if (newEmptySeat.length >0) newEmptySeat[0].state = EMPTY;
739
740   break;
```

We created 8 arbitrary seats, but expanded to 20 when running our simulation. This was sufficient to mirror an Infinite queue in our system given our parameters.

States - STAGING

```
741    case STAGING:
742      // Queueing behavior depends on the customer priority
743      // For this model we will give access to the cashier on a first come, first served basis
744      if (hasArrived){
745        //The customer is staged right next to the cashier
746        if (cashier.state == IDLE){
747          // the cashier is IDLE so this customer is the first to get access
748          cashier.state = BUSY;
749          customer.state = ORDERING;
750          customer.target.row = cashierRow;
751          customer.target.col = cashierCol-1;
752          nextorderingcustomerID_A++;
753
754
```

The last customer moves to get served by the cashier

States – ORDERING/ORDERED

```
case ORDERING:
  // Complete treatment randomly according to the probability of departure

  if (Math.random()< probOrdered){
    var availableseats = chairsIN.filter(function(d){return d.state==IDLE});
    if(availableseats.length != 0){
    var chairNum = Math.floor(Math.random() * availableseats.length);
    cashier.state = IDLE;
    var chairType = availableseats[chairNum].type
    chairsIN[chairType].state = BUSY;
    customer.seatNum = chairType;
    customer.target.row = customer.location.row - 1;
    customer.target.col = customer.location.col;
    customer.state = ORDERED;
    var timeInQueue = currentTime - customer.timeAdmitted;
    //console.log("Time in Queue: " + timeInQueue)
    statistics[1].cumulativeValue = statistics[1].cumulativeValue+timeInQueue;
    statistics[1].count = statistics[1].count + 1;
    queueSeats = queueSeats + 1;
    }
  }

break;

case ORDERED:
if (hasArrived) {
  if (Math.random() <probNoDrinks){
    customer.state = EATING;
    targetChair = customer.seatNum;
    customer.target.row = chairsIN[targetChair].location.row;
    customer.target.col = chairsIN[targetChair].location.col;
  }

  else{
    customer.state = ORDERING2;
    customer.target.row = drinkdispenserRow;
    customer.target.col = drinkdispenserCol -1;
  }
  }
break;
```

After ordering, they are each allocated a seat at random regardless if they are buying a drink or not

Customers move away from the queue, and now decide if they would like to order a drink or not

States – ORDERING2

```
801     case ORDERING2:
802       if (hasArrived){
803         if (Math.random()<probDrinks){
804         customer.state = EATING;
805         targetChair = customer.seatNum;
806         customer.target.row = chairsIN[targetChair].location.row;
807         customer.target.col = chairsIN[targetChair].location.col;
808         }
809       }
810     break;
```

Customers who order their drinks, move off to find the seat previously allocated to them

States - EATING

```
812 ▼   case EATING:
813 ▼     if (hasArrived){
814         if (Math.random()< probEaten){
815         customer.state = EATEN;
816         customer.target.row = entranceRow -2;
817         customer.target.col = entranceCol;
818         // compute statistics for EATEN customer
819
820       }
821       }
822     break;
```

Customers finish eating and leave. This is using a fixed parameter of 0.01.

States - EATEN

```
823     case EATEN:
824       if (hasArrived){
825         customer.state = EXITED;
826         var timeInRestaurant = currentTime - customer.timeAdmitted;
827         //console.log("Time in Restaurant: " + timeInRestaurant)
828         var stats = statistics[0];
829         stats.cumulativeValue = stats.cumulativeValue+timeInRestaurant;
830         stats.count = stats.count + 1;
831         customer.target.row = maxRows;
832         customer.target.col = entranceCol;
833         if (customer.seatNum != null){
834         chairsIN[customer.seatNum].state = IDLE;
835       }
836     }
837     break;
```

Customers leave the restaurant.
The seats they occupied is set back to IDLE.
Time in Restaurant statistic is recorded here

Removing/Updating Customers

```
function removeDynamicAgents(){
  // We need to remove customers who have been EATEN.
  //Select all svg elements of class "customer" and map it to the data list called customers
  var allcustomers = surface.selectAll(".customer").data(customers);
  //Select all the svg groups of class "customer" whose state is EXITED
  var eatencustomers = allcustomers.filter(function(d,i){return d.state==EXITED;});
  // Remove the svg groups of EXITED customers: they will disappear from the screen at this point
  eatencustomers.remove();

  // Remove the EXITED customers from the customers list using a filter command
  customers = customers.filter(function(d){return d.state!=EXITED;});
  // At this point the customers list should match the images on the screen one for one
  // and no customers should have state EXITED
}


function updateDynamicAgents(){
  // loop over all the agents and update their states
  for (var customerIndex in customers){
    updateCustomer(customerIndex);
  }
  updateSurface();
}

function simStep(){
  //This function is called by a timer; if running, it executes one simulation step
  //The timing interval is set in the page initialization function near the top of this file
  if (isRunning){ //the isRunning variable is toggled by toggleSimStep
    // Increment current time (for computing statistics)
    currentTime++;
    // Sometimes new agents will be created in the following function
    addDynamicAgents();
    // In the next function we update each agent
    updateDynamicAgents();
    // Sometimes agents will be removed in the following function
    removeDynamicAgents();
  }
}
```

Removing and updating Customers