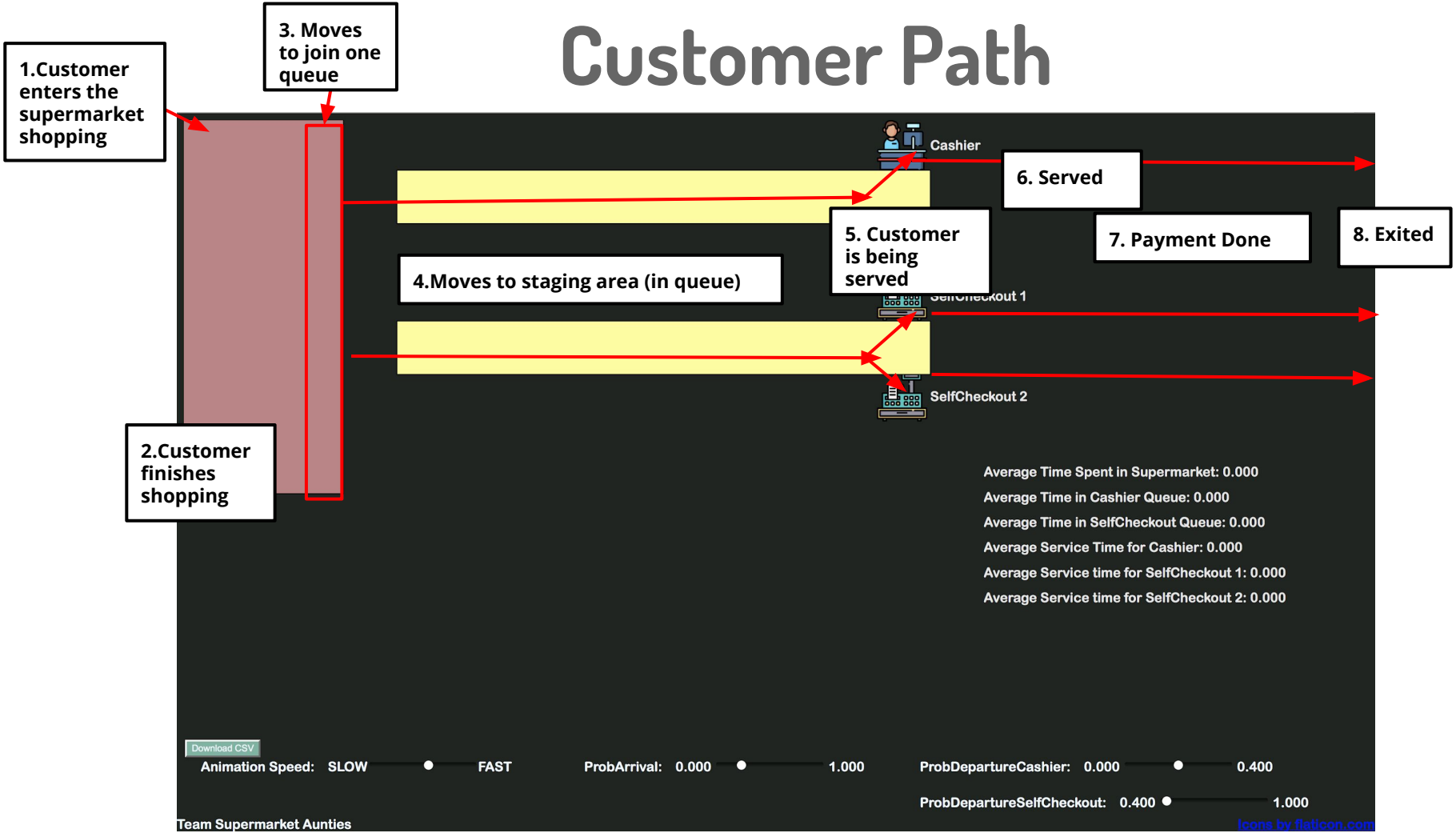
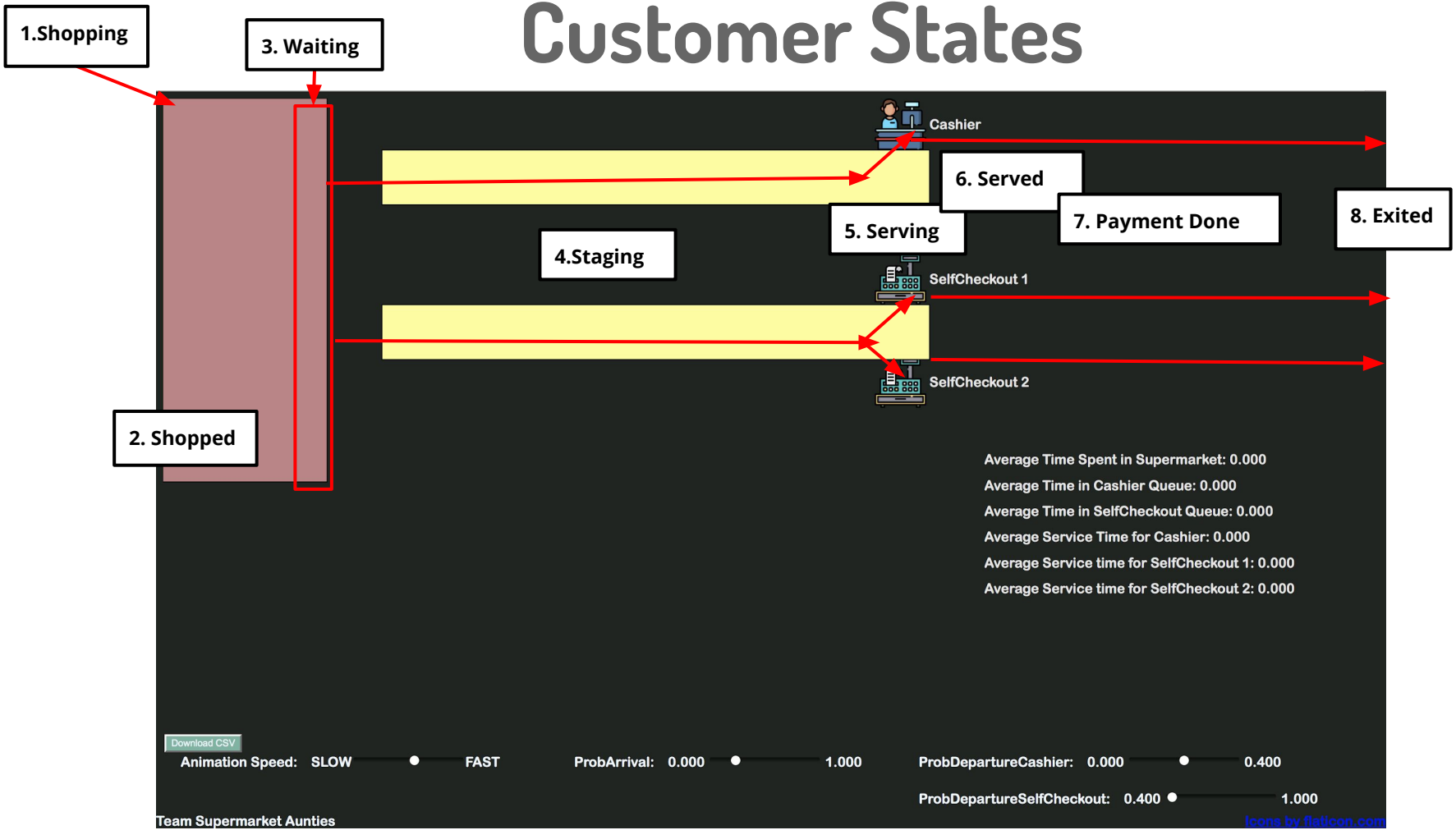
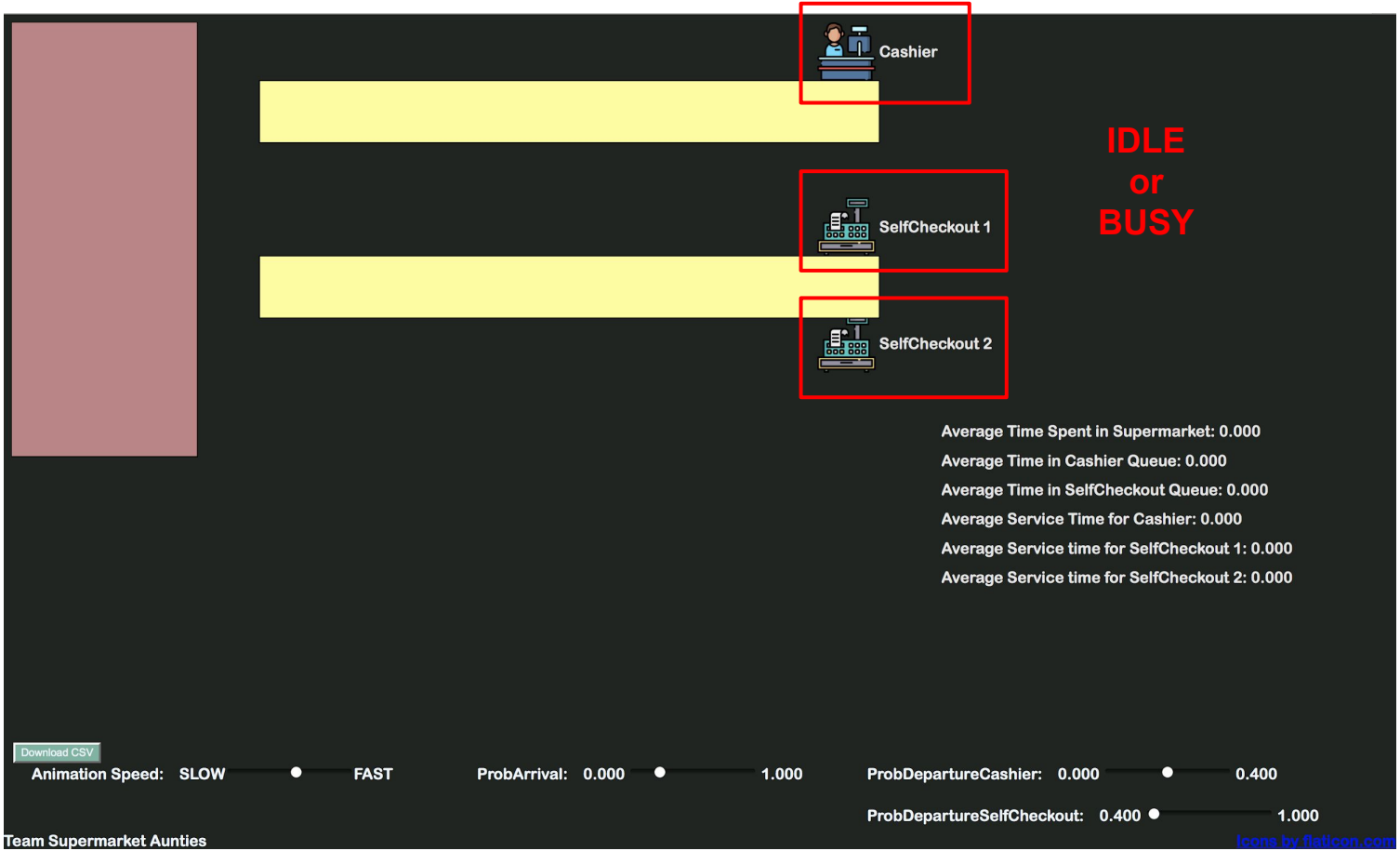
Customer Path



Customer States



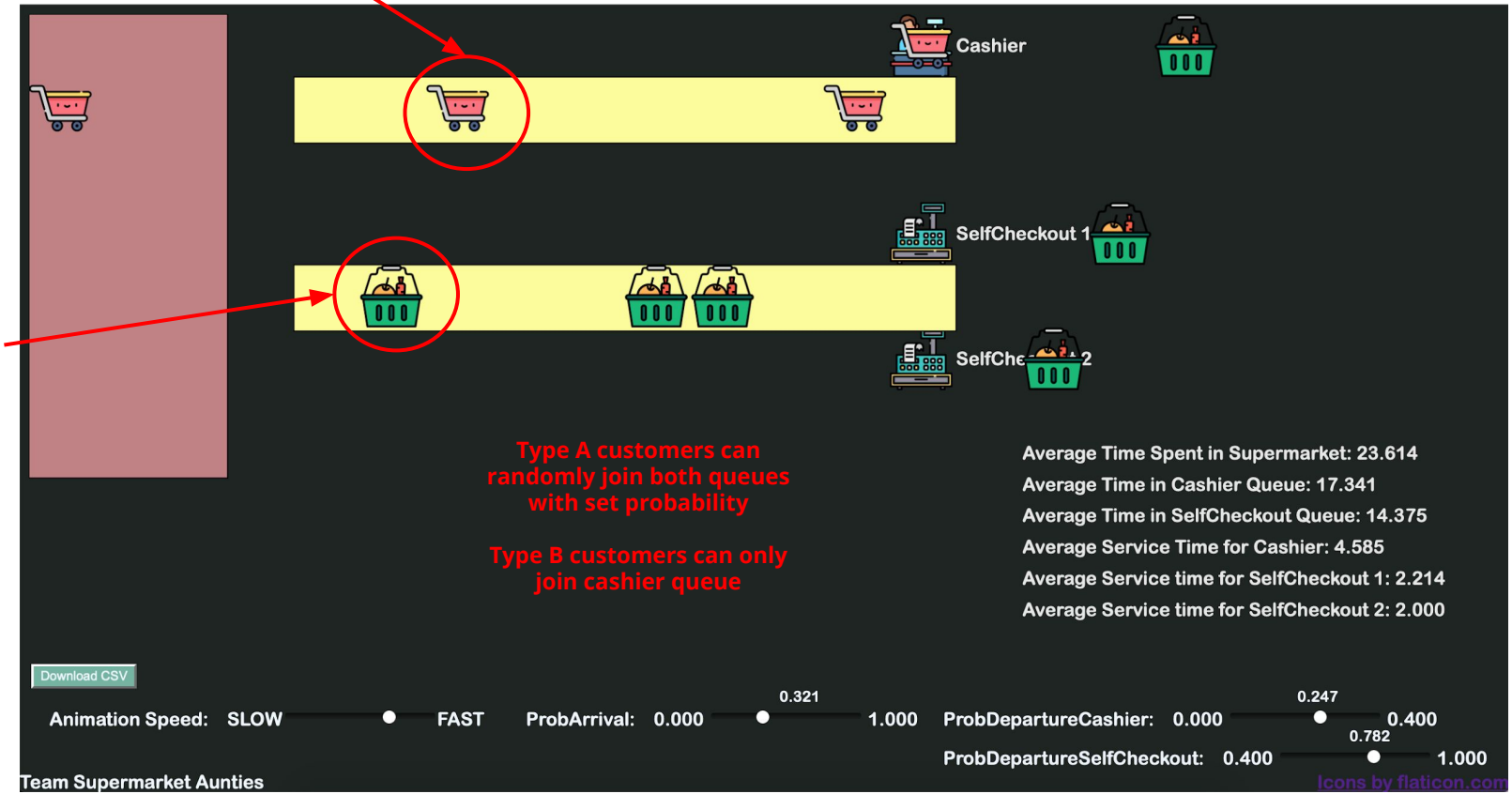
Server States



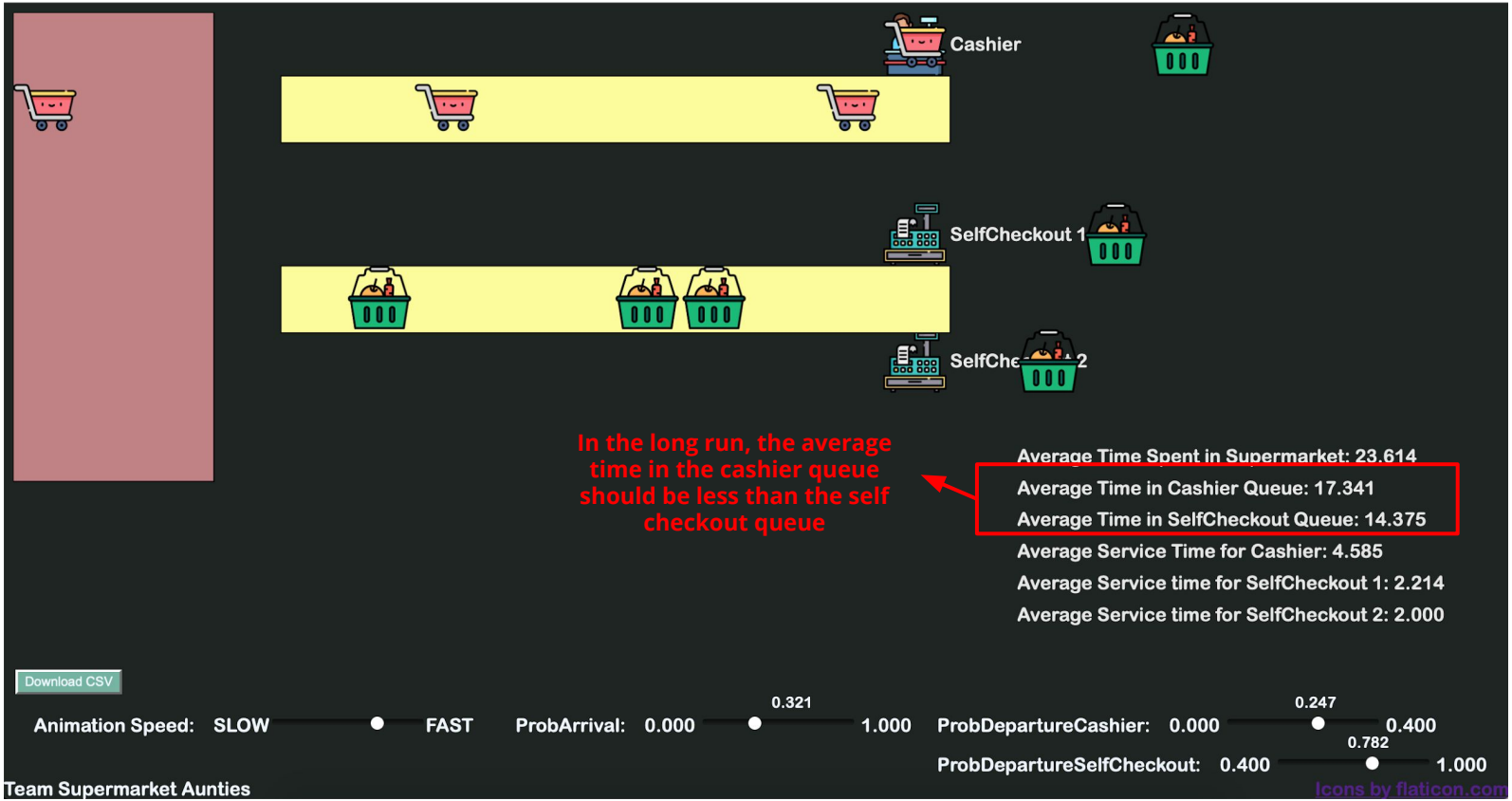
Customer Types

B: Customers with Trolley

A: Customers with Basket



Relative Performance of Queues



SupermarketSimulationModel.html

```html
<!-- The surface is the main playing field for the game -->
<svg  id="surface" style="width:100% height:100%" xmlns="http://www.w3.org/2000/svg" version="1.1" onclick="toggleSimStep();">
</svg>

<div id="title"  style="position:absolute;bottom:0;left:0">Supermarket Simulation Model</div>

<div id="slider_speed" >Animation Speed:   
SLOW<input id="slider1" type="range" min="0" value="275" max="500" step="10" onchange="redrawWindow();" />FAST
</div>

<span id="slider_value2" font-weight:bold;></span><br>
<div id="slider_parr" >ProbArrival:   
0.000 <input  id="slider2" type="range" min="0" max="1.0" step="0.001" name="sld6" value="0.200"
oninput="show_value2(this.value)"> 1.000
</div>


<span id="slider_value3" font-weight:bold;></span><br>
<div id="slider_pdep" >ProbDepartureCashier:   
0.000 <input  id="slider3" type="range" min="0" max="0.4" step="0.001" name="sld6" value="0.200"
oninput="show_value3(this.value)"> 0.400
</div>

<span id="slider_value4" font-weight:bold;></span><br>
<div id="slider_pdepself" >ProbDepartureSelfCheckout:   
0.400 <input  id="slider4" type="range" min="0.4" max="1.0" step="0.001" name="sld6" value="0.200"
oninput="show_value4(this.value)"> 1.000
</div>
<span id
<div id="csvwrapper">
  <button id = "csv" onclick="download_csv()">Download CSV</button>
</div>


<a id="credits" href="http://www.flaticon.com" style="position:absolute;bottom:0;right:0">Icons by flaticon.com</a>
```

Basic Elements of the webpage

- “surface”: an svg element for drawing graphics
- “title”: title label for the page
- “slider_speed”: to control animation speed
- “slider_value2/3/4”: to control different probability rates
- “credits”: a hyperlink to the source of the icons

Imports to html

```
<!-- d3 is for data visualization -->
<script type="text/javascript" src="lib/d3.min.js"></script>


<!-- custom styles for this application -->
<link rel="stylesheet" href="styles/supermarketsimulationmodel.css" media="screen">
```

SupermarketSimulationModel.js is loaded after the page elements have been created

```
<script type="text/javascript" src="lib/SupermarketSimulationModel.js"></script>
```

States and Types

```
//a customer enters the supermarket SHOPPING; he or she then is QUEUEING to be SERVED by a server1;
// then SERVING with the server1; then SERVED;
// When the customer is PAYMENTDONE he or she leaves the supermarket immediately at that point.
const SHOPPING=0;
const SHOPPED=1
const WAITING=2;
const STAGING=3;
const SERVING =4;
const SERVED=5;
const PAYMENTDONE=6;
const EXITED = 7;

// The server1 can be either BUSY serving a customer, or IDLE, waiting for a customer
const IDLE = 0;
const BUSY = 1;

// There are two types of servers in our system: server and eserver
const SERVER = 0;
const ESERVER = 1;
```

Basic Rates

```
// The probability of a customer arrival needs to be less than the probability of a departure, else an infinite queue will build.
// You also need to allow travel time for customers to move from their seat in the waiting room to get close to the server.
// So don't set probDeparture too close to probArrival.
var probArrival = document.getElementById("slider2").value;
// Probability that determines service rate of Cashier
var probDeparture = document.getElementById("slider3").value;
// Probability that determines service rate of Self Checkout
var probDepartureSelfCheckout = document.getElementById("slider4").value;
// Probability that customers finish shopping and proceed to queues
var probShopped = 0.25;
// Probability that basket goes to Self Checkout queue
var probQueue2 = 0.7;

//Different Type of Customers: A(Customers with Baskets), B(Customers with Shopping Carts) according to a probability, probTypeA.
var probTypeA = 0.4;
```

Location Information

```
var serverRow = 1;
var serverCol = 14;
var marketRow = 1;
var marketCol = 1;
```

```
// customers is a dynamic list, initially empty
var customers = [];
// servicers is a static list, populated with a receptionist and a server1
var servicers = [
    {"type":SERVER,"label":"Cashier","location":{"row":serverRow,"col":serverCol},"state":IDLE},
  {"type":ESERVER,"label":"SelfCheckout 1","location":{"row":serverRow+3,"col":serverCol},"state":IDLE},
  {"type":ESERVER,"label":"SelfCheckout 2","location":{"row":serverRow+5,"col":serverCol},"state":IDLE},

];
var server1 = servicers[0]; // the server1 is the first element of the servicers list.
var server2 = servicers[1];
var server3 = servicers[2];


// We can section our screen into different areas. In this model, the waiting area and the staging area are separate.
var areas =[
 {"label":"Waiting Area","startRow":marketRow,"numRows":7,"startCol":marketCol,"numCols":3,"color":"#c28285"},
 {"label":"Staging Area","startRow":serverRow+1,"numRows":1,"startCol":marketCol+4,"numCols":10,"color":"#fdfd96"},
 {"label":"Staging Area","startRow":serverRow+4,"numRows":1,"startCol":marketCol+4,"numCols":10,"color":"#fdfd96"},
]
var supermarket = areas[0]; // the waiting room is the first element of the areas array
```

Location Information

```
var positions= [];
for (i = 0; i < supermarket.numCols; i++) {
    for (j = 0; j < supermarket.numRows; j++) {
        positions.push({"location":{"row":supermarket.startRow + j,"col":supermarket.startCol + i}});
    }
};

var queue_1= [];
for (i = 0; i < areas[1].numCols-1; i++) {
    queue_1.push({"location":{"row":areas[1].startRow,"col":areas[1].startCol + i}});
};

var queue_2= [];
for (i = 0; i < areas[1].numCols-1; i++) {
    queue_2.push({"location":{"row":areas[2].startRow,"col":areas[2].startCol + i}});
};

var queue = [queue_1,queue_2];
```

```
var currentTime = 0;
var statistics = [
{"name":"Average Time Spent in Supermarket: ","location":{"row":7,"col":serverCol+1},"cumulativeValue":0,"count":0},
{"name":"Average Time in Cashier Queue: ","location":{"row":7.75,"col":serverCol+1},"cumulativeValue":0,"count":0},
{"name":"Average Time in SelfCheckout Queue: ","location":{"row":8.5,"col":serverCol+1},"cumulativeValue":0,"count":0},
{"name":"Average Service Time for Cashier: ","location":{"row":9.25,"col":serverCol+1},"cumulativeValue":0,"count":0},
{"name":"Average Service time for SelfCheckout 1: ","location":{"row":10,"col":serverCol+1},"cumulativeValue":0,"count":0},
{"name":"Average Service time for SelfCheckout 2: ","location":{"row":10.75,"col":serverCol+1},"cumulativeValue":0,"count":0}
];
```

Basic Simulation Step

```
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();
    dataCollected.push([
          statistics[0].cumulativeValue, statistics[0].count,
          statistics[1].cumulativeValue, statistics[1].count,
          statistics[2].cumulativeValue, statistics[2].count,
          statistics[3].cumulativeValue, statistics[3].count,
          statistics[4].cumulativeValue, statistics[4].count,
          statistics[5].cumulativeValue, statistics[5].count]);
    console.log(dataCollected);

    console.log("Server1 state");
    console.log(server1.state);
    console.log("Server2 state");
    console.log(server2.state);
    console.log("Server3 state");
    console.log(server3.state);
  }
}
```

At every timestep, push statistics into dataCollected

In the page initialization code, we set a timer to call simStep repeatedly

```
simTimer = window.setInterval(simStep, animationDelay);
```

Add Dynamic Agents

```
function addDynamicAgents(){
  // Customers are dynamic agents: they enter the supermarket, wait, get SERVED, and then leave
  // We have entering customers of two types "A" and "B"
  // We could specify their probabilities of arrival in any simulation step separately
  // Or we could specify a probability of arrival of all customers and then specify the probability of a Type A arrival.
  // We have done the latter. probArrival is probability of arrival a customer and probTypeA is the probability of a type A customer who arrives.
  // First see if a customer arrives in this sim step.
  if (Math.random()< probArrival){


    var newcustomer = {"id":1,"type":"A","location":{"row":0,"col":0},
    "target":{"row":0,"col":0},"state":SHOPPING,"timeEntered":0, "queue":0, "timeQueued":0,"timeServed":0};
    if (Math.random()<probTypeA) newcustomer.type = "A";
    else newcustomer.type = "B";
    customers.push(newcustomer);
  }
```

Every simStep, we add one customer according to the probArrival and assign either “A” or “B” according to probTypeA

Update Dynamic Agents

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

Every simStep we loop over each Dynamic agent to update its state and location

We call updateSurface() to animate the display
We call updateCustomer() to update customer state and ID

Remove Dynamic Agents

```
function removeDynamicAgents(){
  // We need to remove customers who have been PAYMENTDONE.
  //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 SERVEDcustomers = 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
  SERVEDcustomers.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
}
```

Remove patients who have EXITED at every simStep

Update a Dynamic Agents

```
function updateCustomer(customerIndex){
  //customerIndex is an index into the customers data array
  customerIndex = Number(customerIndex); //it seems customerIndex was coming in as a string
  var customer = customers[customerIndex];
  // get the current location of the customer
  var row = customer.location.row;
  var col = customer.location.col;
  var type = customer.type;
  var state = customer.state;
  var queue = customer.queue;
    var chosen;
  // console.log(positions);

  // determine if customer has arrived at destination
  var hasArrived = (Math.abs(customer.target.row-row)+Math.abs(customer.target.col-col))==0;
```

Structure of Update Code

```
switch(state){
  case SHOPPING:
    if (hasArrived){⋯
  break;

  case SHOPPED:
    if (Math.random() < probShopped){⋯
    break;

  case WAITING:
  if (hasArrived){⋯
  break;
  case STAGING:
    if (hasArrived){⋯
  break;

  case SERVING:
    var stats4;
    var stats5;
    var stats6;
    switch(queue){
      case 0:
      if (Math.random()< probDeparture){⋯
      break;
      case 1:
      if (Math.random()< probDepartureSelfCheckout){⋯
      break;
    }
  break;
```

```
  case SERVED:
    if (hasArrived){⋯
  break;
  case PAYMENTDONE:
    if (hasArrived){⋯
  break;
  default:
  break;
}
```

Code to Handle Movement

```
// set the destination row and column
var targetRow = customer.target.row;
var targetCol = customer.target.col;
// compute the distance to the target destination
var rowsToGo = targetRow - row;
var colsToGo = targetCol - col;
// set the speed
var cellsPerStep = 1;
// compute the cell to move to
var newRow = row + Math.min(Math.abs(rowsToGo),cellsPerStep)*Math.sign(rowsToGo);
var newCol = col + Math.min(Math.abs(colsToGo),cellsPerStep)*Math.sign(colsToGo);
// update the location of the customer
customer.location.row = newRow;
customer.location.col = newCol;
```

Handle State SHOPPING

```
switch(state){
  case SHOPPING:
    if (hasArrived){
      customer.timeEntered = currentTime;
              // var stats = statistics[2];
              // stats.count = stats.count + 1;
              if (positions.length > 0) {
                  customer.state = SHOPPED;
                  // pick a random spot in the waiting area to queue
                  // chosen = positions.splice(positions[Math.floor(Math.random()*positions.length)],1);
                  // customer.target.row = chosen[0].location.row;
                  // customer.target.col = chosen[0].location.col;
        customer.target.row = supermarket.startRow+Math.floor(Math.random()*supermarket.numRows);
        customer.target.col = supermarket.startCol+Math.floor(Math.random()*supermarket.numCols);
                  // receptionist assigns a sequence number to each customer to govern order of treatment
              }
      else {
                  customer.state = DISCHARGED;
                  // stats.cumulativeValue = stats.cumulativeValue+100;
                  customer.target.row = 1;
          customer.target.col = 0;
              }
    }
  break;
```

Handle State SHOPPED

```
case SHOPPED:
  if (Math.random() < probShopped){
    if (customer.type=="A") {
      if (Math.random() > probQueue2){
        customer.target.col = marketCol+3;
        customer.queue = 0;
        customer.target.row = 2;
        customer.id = ++nextCustomerID_1;
      }
      else {
        customer.queue = 1;
        customer.target.row = 5;
        customer.id = ++nextCustomerID_2;
      }
    }

          else {
      customer.target.col = marketCol+3;
      customer.queue = 0;
      customer.target.row = 2;
      customer.id = ++nextCustomerID_1;
    }
    customer.state = WAITING;
    // positions.push({"location":{"row":customer.target.row,"col":customer.target.col}});
  }
  break;
```

Handle State WAITING

```
case WAITING:
if (hasArrived){
    customer.timeQueued = currentTime;
    switch (queue){
      case 0:
        for (i=0; i < 10; i++){
          if (customer.id == nextServicedCustomerID_1+i){
            customer.state = STAGING;
            customer.target.col = serverCol-i
          }
        }

      break;
      case 1:
        for (j=0; j < 10; j++){
          if (customer.id == nextServicedCustomerID_2+j){
            customer.state = STAGING;
            customer.target.col = serverCol-j

          }
        }
      break;

    }
  }
break;
```

Handle State STAGING

```
case STAGING:
  // Queueing behavior depends on the customer priority
  // For this model we will give access to the server1 on a first come, first served basis
  if (hasArrived){
    switch(queue){
      case 0:

      //The customer is staged right next to the server1
        if (server1.state == IDLE){
          // the server1 is IDLE so this customer is the first to get access
          server1.state = BUSY;
          customer.state = SERVING;
          customer.target.col = serverCol;
          customer.target.row = serverRow;
          nextServicedCustomerID_1++;
          customer.timeServed = currentTime

        }
      break;
```

```
      case 1:
      //The customer is staged right next to the server1
        if (server2.state == IDLE){
          // the server1 is IDLE so this customer is the first to get access
          server2.state = BUSY;
          customer.state = SERVING;
          customer.target.col = server2.location.col;
          customer.target.row = server2.location.row;
          nextServicedCustomerID_2++;
          customer.timeServed = currentTime
        }
        else if (server3.state == IDLE){
          server3.state = BUSY;
          customer.state = SERVING;
          customer.target.col = server3.location.col;
          customer.target.row = server3.location.row;
          nextServicedCustomerID_2++;
          customer.timeServed = currentTime
        }
      break;
    }
  }
break;
```

Handle State SERVING

```
case SERVING:
  // Complete treatment randomly according to the probability of departure
  var stats4;
  var stats5;
  var stats6;
  switch(queue){
    case 0:
    if (Math.random()< probDeparture){
      customer.state = SERVED;
      server1.state = IDLE;
      stats4 = statistics[3]
      stats4.cumulativeValue = stats4.cumulativeValue+currentTime - customer.timeServed;
      stats4.count = stats4.count + 1;
    }
    break;
    case 1:
    if (Math.random()< probDepartureSelfCheckout){
      customer.state = SERVED;
      if (customer.location.row == server2.location.row){
        server2.state = IDLE;
        stats5 = statistics[4]
        stats5.cumulativeValue = stats5.cumulativeValue+currentTime - customer.timeServed;
        stats5.count = stats5.count + 1;
      }
      else if (customer.location.row == server3.location.row){
        server3.state = IDLE;
        stats6 = statistics[5]
        stats6.cumulativeValue = stats6.cumulativeValue+currentTime - customer.timeServed;
        stats6.count = stats6.count + 1;
      }
    }
    break;
```

Handle State SERVED

```
case SERVED:
  if (hasArrived){
    customer.state = DISCHARGED;
    // customer.target.row = serverCol;
    customer.target.col = serverCol+5;
    // compute statistics for discharged customer
    var timeInSupermarket = currentTime - customer.timeEntered;
    var timeInQueue = currentTime - customer.timeQueued;
    var stats;

    if (customer.queue == 0){
      stats = statistics[1];
    }
    else if(customer.queue == 1){
      stats = statistics[2];
    }
    stats.cumulativeValue = stats.cumulativeValue+timeInQueue;
    stats.count = stats.count + 1;
    var stats2

    stats2 = statistics[0];
    stats2.cumulativeValue = stats2.cumulativeValue+timeInSupermarket;
    stats2.count = stats2.count + 1;
  }
break;
```

Handle State PAYMENTDONE

```
    case PAYMENTDONE:
      if (hasArrived){
        customer.state = EXITED;
      }
    break;
    default:
    break;
  }
```

Data Initialization

```
nextCustomerID_1 = 0;
nextCustomerID_2 = 0;
nextServicedCustomerID_1 = 1;
nextServicedCustomerID_2 = 1;
currentTime = 0;
server1.state=IDLE;
server2.state=IDLE;
server3.state=IDLE;
statistics[0].cumulativeValue=0;
statistics[0].count=0;
statistics[1].cumulativeValue=0;
statistics[1].count=0;
statistics[2].cumulativeValue=0;
statistics[2].count=0;
statistics[3].cumulativeValue=0;
statistics[3].count=0;
statistics[4].cumulativeValue=0;
statistics[4].count=0;
statistics[5].cumulativeValue=0;
statistics[5].count=0;
customers = [];
dataCollected = [];
```

increment this and assign it to the next entering customer

id of the next customer

Data Download

```
// Downloads output statistics into csv format
function download_csv() {
    var csv = 'CumVal_0,Count_0,CumVal_1,Count_1,CumVal_2,Count_2,CumVal_3,Count_3,CumVal_4,Count_4,CumVal_5,Count_5\n';
    dataCollected.forEach(function(row) {
            csv += row.join(',');
            csv += "\n";
    });

    var hiddenElement = document.createElement('a');
    hiddenElement.href = 'data:text/csv;charset=utf-8,' + encodeURI(csv);
    hiddenElement.target = '_blank';
    hiddenElement.download = 'Output_statistics.csv';
    hiddenElement.click();
}
```

REFERENCES

Week 3 Lecture 5 Agent-Based Modeling.pdf
Peter L Jackson