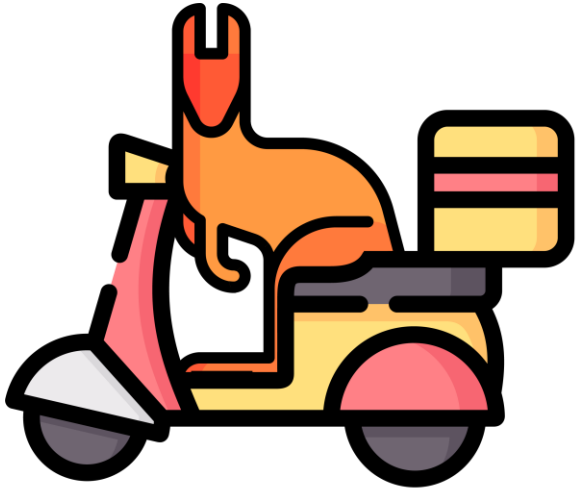


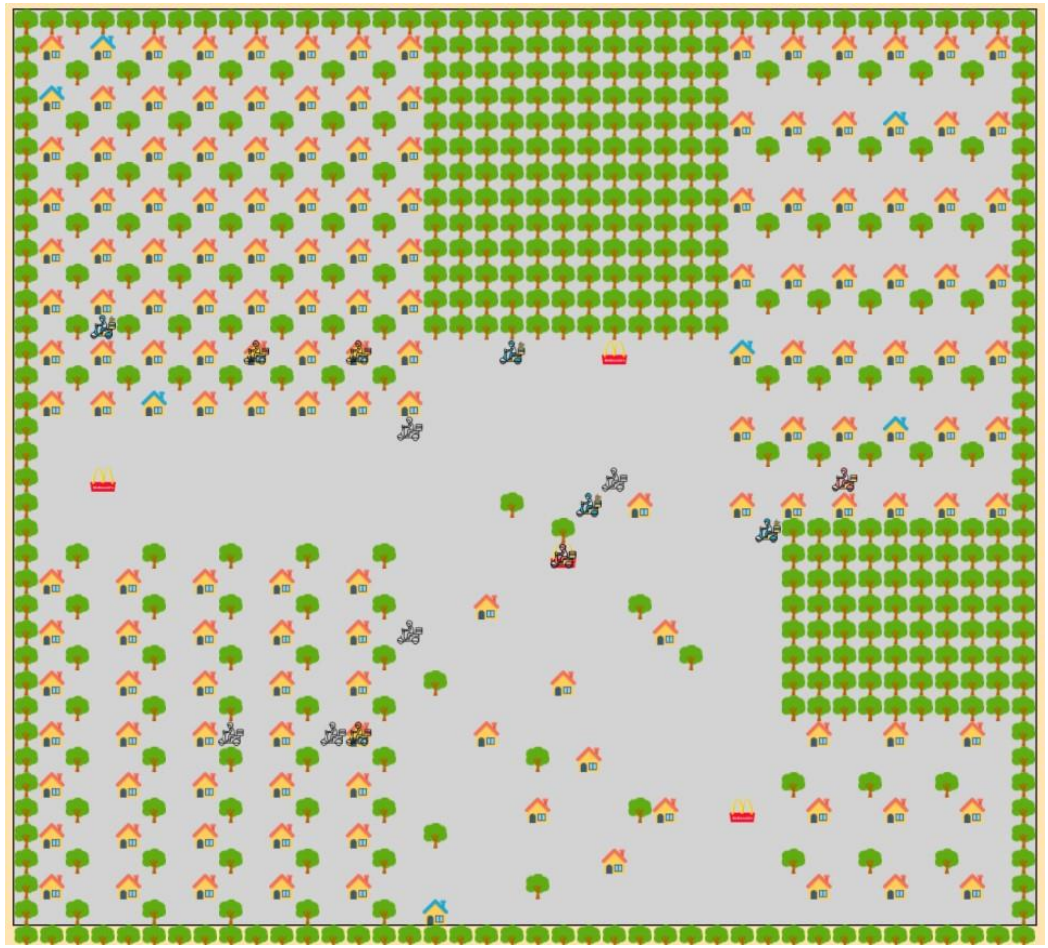
MCDELIVEROO

Simulating online food delivery system in Singapore

MCDELIVEROO

Environment and Agents

City



= Has not ordered

= Has ordered

= McDonald's Outlet

= Tree

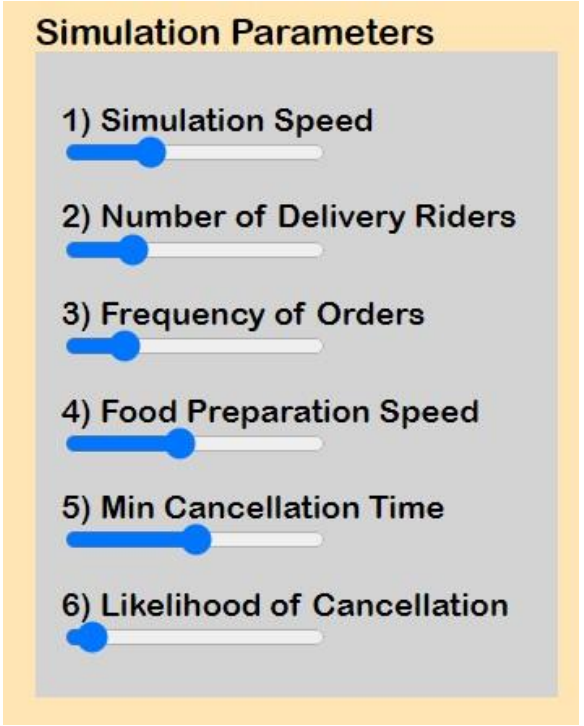
= Has not received an order

= Picking up order

= Delivering order

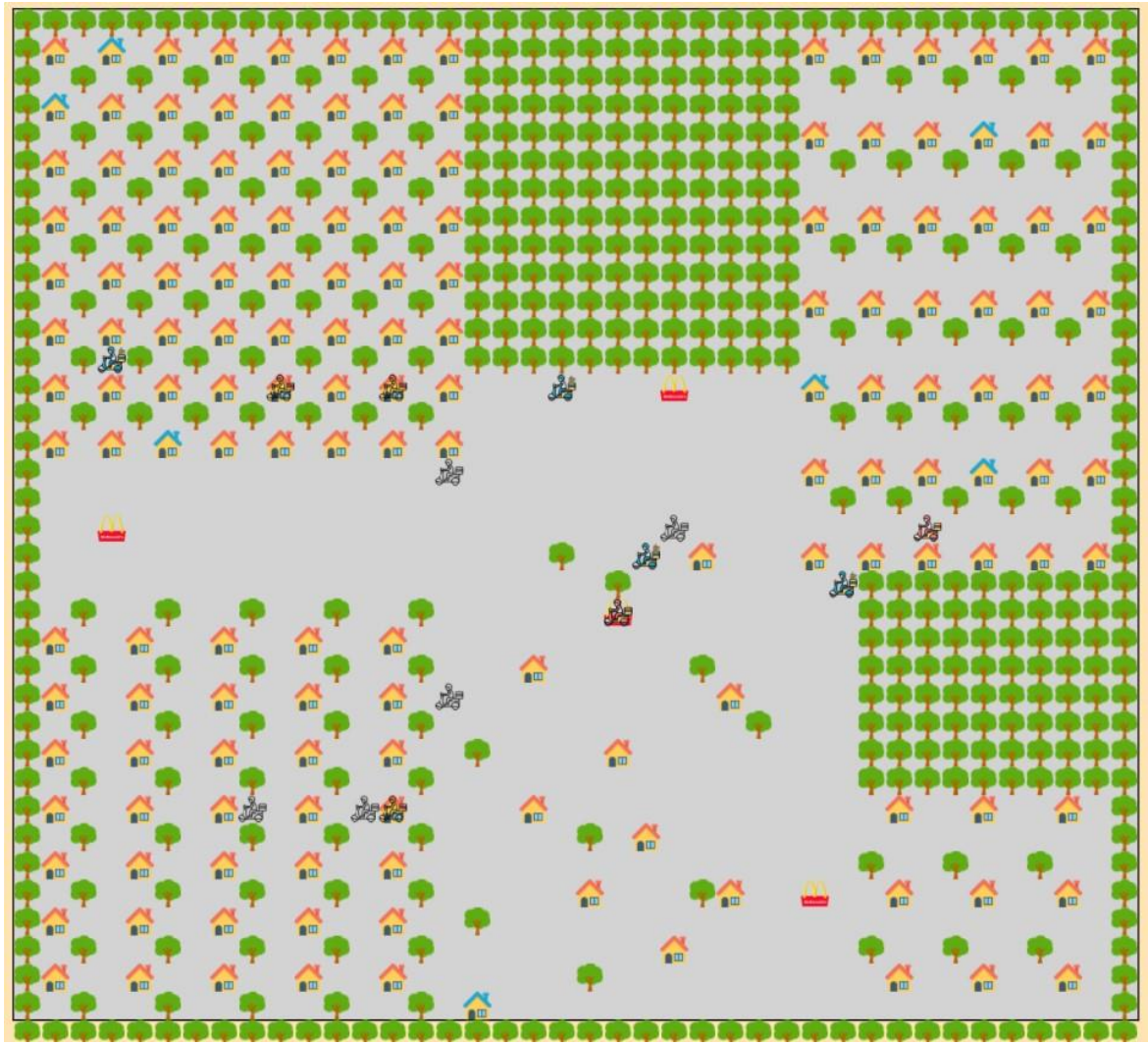
Simulation Parameters

- **Simulation Speed:** Allows user to vary the speed of simulation
- **Number of Delivery Riders:** Sets the number of available riders to be in the region
- **Frequency of Orders:** Sets the likelihood of the arrival of an order
- **Food Preparation Speed:** Sets the time taken for a restaurant to process an order and finish preparing the food.
- **Min Cancellation Time:** A threshold that allows customer to cancel their order after a certain time.
- **Likelihood of Cancellation:** The probability of a customer cancelling their order per timestep above the min threshold.



Riders Path

- Riders **appear at random initial positions**
- The number of available riders would appear based on the set parameter
- Riders can **only move in 2 directions** - along the x and y axis. (No diagonal movement)
- The **trees behave as obstacles** that riders cannot go through.
- Riders would roam around until an order is assigned to them.



Simulation Rules - Delivery

- **Orders have a certain probability to arrive** per cycle/1 min
- If a **rider** is **not** assigned to any orders, it will **continue to roam around**
- Orders would be **assigned to any** available delivery riders (randomly)
  - This method would be fairer from the rider's perspective
  - Even if a rider is not the nearest one from the customer, one can still receive the order.
- Once assigned, riders would be making their way to the **nearest McDonalds from the rider**.
- At the same time, the restaurant would prepare the food **within the set time duration**.
- The rider would then deliver the food to the customer's house **via shortest route**.

Simulation Rules – Cancellation

- Orders **have a certain probability to get cancelled** by the customer
- The **customer's patience can be visualised as** the ***'Min Cancellation Time'***
- **After which,** there would be **a probability each min that the customer may cancel** their order.
- This **probability** parameter is set as the ***'Likelihood of Cancellation'***

Assets

MCDELIVEROO

```
13    //urls for sprites
14    const urlWithoutorder = "images/Withoutorder.png";
15    const urlPickup = "images/Pickup.png";
16    const urlDelivery = "images/Delivery.png";
17    const urlHouse = "images/House.png";
18    const urlHouse_Ordered = "images/House_Ordered.png";
19    const urlMcd = "images/Mcd.png";
20    const urlBuilding = "images/Building.png";
```

States Tracking

MCDELIVEROO

```
22  //Delivery rider states
23  const WAITING_ORDER = 0;
24  const COLLECTING_ORDER = 1;
25  const DELIVERING_FOOD = 2;
26  const EXITED = 3;
27
28  //Order status of each house
29  const UNORDERED = 2;
30  const ORDERED = 1;
31
32  //for tracking state of food preparation
33  const notprepared = 0
34  const prepared = 1
```

Mass Generation of Customer Houses

```
67  //houses at top left corner
68  for (i=srow+1;i<srow+16;i+=2){
69      for (j=scol+1;j<scol+16;j+=2){
70          var newhouse = {"id":house_count,"location":{"row":i,"col":j},"state":UNORDERED};
71          house_count++;
72          houses.push(newhouse);
73      }
74  }
75
76  //houses at bottom left corner
77  for (i=nrow-6;i<nrow+2;i+=3){
78      for (j=ncol-7;j<ncol+1;j+=3){
79          var newhouse = {"id":house_count,"location":{"row":i,"col":j},"state":UNORDERED};
80          house_count++;
81          houses.push(newhouse);
82      }
83  }
84
85  //houses at top right corner
86  for (i=srow+1;i<srow+20;i+=3){
87      for (j=ncol-10;j<ncol+1;j+=2){
88          var newhouse = {"id":house_count,"location":{"row":i,"col":j},"state":UNORDERED};
89          house_count++;
90          houses.push(newhouse);
91      }
92  }
93
94  //houses at bottom left corner
95  for (i=nrow-12;i<nrow+2;i+=2){
96      for (j=scol+1;j<scol+16;j+=3){
97          var newhouse = {"id":house_count,"location":{"row":i,"col":j},"state":UNORDERED};
98          house_count++;
99          houses.push(newhouse);
100     }
101 }
```

*Similarly, for the generation of trees as obstacles

Randomly Shuffle Riders

```
188  //function to randomly shuffle riders so everyone gets an equal chance of receiving an order
189  function shuffle(array) {
190      var currentIndex = array.length, temporaryValue, randomIndex;
191
192      // While there remain elements to shuffle...
193      while (0 !== currentIndex) {
194
195        // Pick a remaining element...
196        randomIndex = Math.floor(Math.random() * currentIndex);
197        currentIndex -= 1;
198
199        // And swap it with the current element.
200        temporaryValue = array[currentIndex];
201        array[currentIndex] = array[randomIndex];
202        array[randomIndex] = temporaryValue;
203      }
204
205      return array;
206  }
```

Initialize Variables

```
208   //initialization variables
209   var probMovement = 0.2; //probability that riders change direction/target when roaming
210   var probCancel =  0.061; //probability that an order is cancelled after min cancellation time (per cycle/1min)
211   var timeCancel = 60; //minimum cancellation time
212   var currentTime = 0; //time variable for tracking
213   var probFoodArrival = 0.2; //probability that a food for an order is done preparing (per cycle/1min)
214   var probOrderArrival = 0.002; //probability that a house orders McD (per cycle/1min)
215   var order_counter = 0; //counter variable for tracking total orders
216   var delivered_counter = 0; //counter variable for tracking total number of deliveries
217   var cancel_counter = 0; //counter variable for tracking total number of cancelations
218   var citizens = []; //list variable for storing information on all delivery riders
219   var num_citizens = 15; //number of delivery rider intialized at the start
220   var order_time = []; //list variable for storing information on delivery time for each order
221   var orders = []; //list variable for storing information on all orders
222
```

Statistics Calculation

```
225  //function to calculate cancellation rate
226  function cancellation_rate() {
227      if (order_counter == 0){
228          return 0
229      } else {
230          return (cancel_counter/order_counter)*100
231      }
232  }
233
234  //function to calculate average delivery time per order
235  function average(order_time, sum){
236      if (order_time.length > 0){
237          for (i=0; i<order_time.length; i++){
238              sum += order_time[i]
239          }
240          var avg = sum/(order_time.length)
241          return avg
242      }   else {
243          var avg = 0
244          return avg
245      }
246  }
247
```

```
372                  cancel_counter++;
```

```
691                  delivered_counter++;
```

Statistics Display - 1

```
401  var statistics = [
402      {"name":"Number of Delivery Riders: ","location":{"row":21,"col":scol+ncol+1},"stat":0}, //num_citizens
403      {"name":"Time Elapsed(min): ","location":{"row":22,"col":scol+ncol+1},"stat":0}, //currentTime
404      {"name":"Number of Completed Deliveries: ","location":{"row":23,"col":scol+ncol+1},"stat":0}, //delivered_counter
405      {"name":"Number of Cancelled Deliveries: ","location":{"row":24,"col":scol+ncol+1},"stat":0}, //cancel_counter
406      {"name":"% of Orders Cancelled: ","location":{"row":25,"col":scol+ncol+1},"stat":0},            //cancelled_percent
407      {"name":"Min Cancellation Time(min): ","location":{"row":26,"col":scol+ncol+1},"stat":0},           //timeCancel = 60;
408      {"name":"Average Delivery Time per Order(min): ","location":{"row":27,"col":scol+ncol+1},"stat":0}, //delivery_average_time
409      {"name":"Average Number of Deliveries per Rider per Hour: ","location":{"row":28,"col":scol+ncol+1},"stat":0}
410      ];
```

Statistics Display - 2

```
778 function simStep(){
779     //This function is called by a timer; if running, it executes one simulation step
780     //The timing interval is set in the page initialization function near the top of this file
781     if (isRunning){ //the isRunning variable is toggled by toggleSimStep
782         currentTime++;
783         updateOrders();
784         checkriderAvailability();
785         updatedfoodArrival();
786         updateDynamicAgents();
787         removeOrders();
788         cancelOrders();
789         //update statistics
790         statistics[0].stat = num_citizens;
791         statistics[1].stat = currentTime;
792         statistics[2].stat = delivered_counter;
793         statistics[3].stat = cancel_counter;
794         statistics[4].stat = Math.round(cancellation_rate());
795         statistics[5].stat = timeCancel;
796         statistics[6].stat = average(order_time,0).toFixed(1);
797         statistics[7].stat = ((delivered_counter/(currentTime/60))/num_citizens).toFixed(1);
798     }
```

Refresh orders per cycle

```
273  //function to create/refresh all delivery orders (per cycle/1min)
274  function updateOrders(){
275      for (a=0;a<orders.length;a++){
276          orders[a].orderposition = a;
277          orders[a].time++;
278      }
279      var houseorders = houses.filter(function(d){return d.state == UNORDERED});
280      if (houseorders.length > 0) {
281          for (i = 0;i < houseorders.length; i++) {
282              if (Math.random()<probOrderArrival) {
283                  var neworder = {"orderid":order_counter,"orderposition":orders.length,"houseid":houseorders[i].id,"rider":null,"state":notprepared,"time":0, "c
284                  orders.push(neworder);
285                  order_counter++;
286                  for (j = 0; j<houses.length; j++) {
287                      if (houses[j].id == neworder.houseid){
288                          houses[j].state = ORDERED;
289                      }
290                  }
291              }
292          }
293      }
294  }
```

Assigning Orders to Random Available Riders

MCDELIVEROO

```
296  //function to assign orders to available delivery riders (per cycle/1min)
297  function checkriderAvailability(){
298      var availableriders = citizens.filter(function(d){return d.available == 1});
299      var availableorders = orders.filter(function(d){return d.rider == null});
300      shuffle(availableorders)
301      shuffle(availableriders)
302      const no_availableriders = availableriders.length;
303      const no_availableorders = availableorders.length;
304      if (no_availableriders > 0 && no_availableorders > 0) {
305          if (no_availableriders > no_availableorders) {
306              for (i=0;i<no_availableorders;i++){
307                  index = availableorders[i].orderposition
308                  orders[index].rider = availableriders[i].id
309              }
310          } else {
311              for (i=0;i<no_availableriders;i++){
312                  index = availableorders[i].orderposition
313                  orders[index].rider = availableriders[i].id
314              }
315          }
316      }
317  }
```

Food Preparation

```
319   //function to determine whether food is prepared (per cycle/1min)
320   function updatedfoodArrival(){
321       var foodpreparation = orders.filter(function(d){return d.rider != null});
322       if (foodpreparation.length > 0) {
323           for (i = 0; i < foodpreparation.length; i++) {
324               if (Math.random()<probFoodArrival) {
325                   var assigned_orderid = foodpreparation[i].orderid;
326                   for (j = 0; j < orders.length; j++) {
327                       if (orders[j].orderid == assigned_orderid) {
328                           orders[j].state = prepared;
329                       }
```

Removing Delivered Orders

```
336   //function to remove orders once they are delivered (per cycle/1min)
337 function removeOrders(){
338     delivered_citizens = citizens.filter(function(d){return d.available == 2});
339     for (i=0;i<delivered_citizens.length;i++){
340         for (j=0;j<orders.length;j++){
341             if (orders[j].orderid == delivered_citizens[i].orderid){
342                 order_time.push(orders[j].time);
343                 orders = orders.filter(function(d){return d.orderid != orders[j].orderid});
344             }
345         }
346         for (k=0;k<houses.length;k++){
347             if (houses[k].id == delivered_citizens[i].houseid){
348                 houses[k].state = UNORDERED;
349             }
350         }
351         for (l=0;l<citizens.length;l++){
352             if (citizens[l].id == delivered_citizens[i].id){
353                 citizens[l].available = 1;
354                 citizens[l].houseid = null;
355                 citizens[l].houseloc = null;
356                 citizens[l].orderid = null;
357                 var targetrow=Math.floor(Math.random() * ((nrow+srow) - srow) +srow);
358                 var targetcol=Math.floor(Math.random() * ((ncol+scol) - scol) +scol);
359                 citizens[l].target.row = targetrow;
360                 citizens[l].target.col = targetcol;
```

Removing Cancelled Orders

```
366   //function to remove cancelled orders (per cycle/1min)
367   function cancelOrders(){
368       for (a=0;a<orders.length;a++){
369           if (orders[a].time > timeCancel){
370               if (Math.random() < probCancel) {
371                   orders[a].cancelled = 1;
372                   cancel_counter++;
373                   console.log('CANCELLED')
374               }
375           }
376       }
377       cancelled_orders = orders.filter(function(d){return d.cancelled == 1});
378       for (i=0;i<cancelled_orders.length;i++){
379           for (j=0;j<citizens.length;j++){
380               if (citizens[j].orderid == cancelled_orders[i].orderid){
381                   citizens[j].available = 1;
382                   citizens[j].houseloc = null;
383                   citizens[j].orderid = null;
384                   var targetrow=Math.floor(Math.random() * ((nrow+srow) - srow) +srow);
385                   var targetcol=Math.floor(Math.random() * ((ncol+scol) - scol) +scol);
386                   citizens[j].target.row = targetrow;
387                   citizens[j].target.col = targetcol;
388                   citizens[j].state = WAITING_ORDER;
389               }
390           }
391           for (k=0;k<houses.length;k++){
392               if (houses[k].id == cancelled_orders[i].houseid){
393                   houses[k].state = UNORDERED;
394               }
395           }
396       }
397       orders = orders.filter(function(d){return d.cancelled != 1});
```

Removing Cancelled Orders

```
366   //function to remove cancelled orders (per cycle/1min)
367   function cancelOrders(){
368       for (a=0;a<orders.length;a++){
369           if (orders[a].time > timeCancel){
370               if (Math.random() < probCancel) {
371                   orders[a].cancelled = 1;
372                   cancel_counter++;
373                   console.log('CANCELLED')
374               }
375           }
376       }
377       cancelled_orders = orders.filter(function(d){return d.cancelled == 1});
378       for (i=0;i<cancelled_orders.length;i++){
379           for (j=0;j<citizens.length;j++){
380               if (citizens[j].orderid == cancelled_orders[i].orderid){
381                   citizens[j].available = 1;
382                   citizens[j].houseloc = null;
383                   citizens[j].orderid = null;
384                   var targetrow=Math.floor(Math.random() * ((nrow+srow) - srow) +srow);
385                   var targetcol=Math.floor(Math.random() * ((ncol+scol) - scol) +scol);
386                   citizens[j].target.row = targetrow;
387                   citizens[j].target.col = targetcol;
388                   citizens[j].state = WAITING_ORDER;
389               }
390           }
391           for (k=0;k<houses.length;k++){
392               if (houses[k].id == cancelled_orders[i].houseid){
393                   houses[k].state = UNORDERED;
394               }
395           }
396       }
397       orders = orders.filter(function(d){return d.cancelled != 1});
```

Riders Behaviours – No Order

```
621     // Behavior of citizen depends on his or her state
622     switch(state){
623         case WAITING_ORDER:
624             if (hasArrived){
625                 //this section of the code makes the rider roam if he has no order)
626                 var targetrow=Math.floor(Math.random() * ((nrow+srow) - srow) +srow);
627                 var targetcol=Math.floor(Math.random() * ((ncol+scol) - scol) +scol);
628                 var targetisbuilding=Buildings.filter(function(d){return d.row==targetrow && d.col==targetcol;});
629                 while (targetisbuilding.length>0){
630                     var targetrow=Math.floor(Math.random() * ((nrow+srow) - srow) +srow);
631                     var targetcol=Math.floor(Math.random() * ((ncol+scol) - scol) +scol);
632                     var targetisbuilding=Buildings.filter(function(d){return d.row==targetrow && d.col==targetcol;});
633                 }
634                 citizen.target.row = targetrow;
635                 citizen.target.col = targetcol;
636                 } else {
637                 //this section of the code makes the rider roam if he has no order)
638                     if (Math.random()<probMovement){
639                         var targetrow=Math.floor(Math.random() * ((nrow+srow) - srow) +srow);
640                         var targetcol=Math.floor(Math.random() * ((ncol+scol) - scol) +scol);
641                         var targetisbuilding=Buildings.filter(function(d){return d.row==targetrow && d.col==targetcol;});
642                         while (targetisbuilding.length>0){
643                             var targetrow=Math.floor(Math.random() * ((nrow+srow) - srow) +srow);
644                             var targetcol=Math.floor(Math.random() * ((ncol+scol) - scol) +scol);
645                             var targetisbuilding=Buildings.filter(function(d){return d.row==targetrow && d.col==targetcol;});
646                         }
647                         citizen.target.row = targetrow;
648                         citizen.target.col = targetcol;
649                     }
650                 }
651             for (i=0;i<orders.length;i++){
652                 if (orders[i].rider == citizen.id){
```

Riders Behaviours – Collecting Order

```
674         case COLLECTING_ORDER:
675             if (hasArrived){
676                 for (i=0; i<orders.length; i++){
677                     if (orders[i].rider == citizen.id) {
678                         if (orders[i].state == prepared) {
679                             citizen.state = DELIVERING_FOOD;
680                             citizen.target.row = citizen.houseloc.row;
681                             citizen.target.col = citizen.houseloc.col;
682                             }
683                         }
684                     }
685                 }
686         break;
```

Riders Behaviours – Delivering Order

```
687             case DELIVERING_FOOD:
688                 if (hasArrived){
689                     citizen.state = WAITING_ORDER;
690                     citizen.available = 2;
691                     delivered_counter++;
692                 }
693             break;
```

Riders Movement – 1

```
698     // set the current row and column of the citizen
699     var currentrow=citizen.location.row;
700     var currentcol=citizen.location.col;
701
702     // set the destination row and column
703     var targetRow = citizen.target.row;
704     var targetCol = citizen.target.col;
705
706     //Compute all possible directions for a citizen
707     nextsteps=[];
708      for(const dx of [-1, 0, 1]) {
709         for(const dy of [-1, 0, 1]) {
710             for (const dz of [-1,1]) {
711           if(dx === 0 && dy === 0) continue;
712             if (currentrow + dx == currentrow && currentcol + dy != currentcol){
713                 nextsteps.push({ row: currentrow + 0, col: currentcol + dz })
714             }
715             else if (currentrow + dx != currentrow && currentcol + dy == currentcol) {
716                 nextsteps.push({ row: currentrow + dz , col: currentcol + 0 })
717             }
718
719             }
720         }
721      }
```

Riders Movement – 2

```
722     // Compute distance of each possible step to the destination
723     stepdistance=[]
724     for (i = 0; i < nextsteps.length-1; i++) {
725         var nextstep=nextsteps[i];
726         var nextrow=nextstep.row
727         var nextcol=nextstep.col
728         stepdistance[i]=Math.sqrt((nextrow-targetRow)*(nextrow-targetRow)+(nextcol-targetCol)*(nextcol-targetCol));
729     }
730
731     //identify if the best next step (i.e. the step with the shortest distance to the target) is a building
732     var indexMin = stepdistance.indexOf(Math.min(...stepdistance));
733     var minnexstep=nextsteps[indexMin];
734     var nextsteprow=minnexstep.row;
735     var nextstepcol=minnexstep.col;
736
737     var nextstepisbuilding=Buildings.filter(function(d){return d.row==nextsteprow && d.col==nextstepcol;});
738     //If the best next step is a building, then we analyze the 2nd best next step...etc, until the next step is not a building
739     //Citizens cannot move through the buildings!
740     while (nextstepisbuilding.length>0){
741         nextsteps.splice((indexMin), 1);
742         stepdistance.splice((indexMin), 1);
743         var indexMin = stepdistance.indexOf(Math.min(...stepdistance));
744         var minnexstep=nextsteps[indexMin];
745         var nextsteprow=minnexstep.row;
746         var nextstepcol=minnexstep.col;
747         var nextstepisbuilding=Buildings.filter(function(d){return d.row==nextsteprow && d.col==nextstepcol;});
748     }
749
750     // compute the cell to move to
751     var newRow = nextsteprow;
752     var newCol = nextstepcol;
```

Visuals – Sliders, Legend and Logo

```
<div class = "placeholder-box">
    <p style="position:absolute;top:2.5%;left:5%"> 1) Simulation Speed </p>
    <p><input id="slider1" type="range" min="0" value="300" max="1000" step="10" onchange="redrawWindow();" style="position:absolute;top:12.5%
    <p style="position:absolute;top:17.5%;left:5%"> 2) Number of Delivery Riders</p>
    <p><input id="slider2" type="range" min="5" value="15" max="50" step="1" onchange="redrawWindow();" style="position:absolute;top:27.5%;lef
    <p style="position:absolute;top:32.5%;left:5%"> 3) Frequency of Orders</p>
    <p><input id="slider3" type="range" min="0.0001" value="0.002" max="0.01" step="0.0001" onchange="redrawWindow();" style="position:absolut
    <p style="position:absolute;top:47.5%;left:5%"> 4) Food Preparation Speed</p>
    <p><input id="slider4" type="range" min="0.05" value="0.2" max="0.4" step="0.01" onchange="redrawWindow();" style="position:absolute;top:5
    <p style="position:absolute;top:62.5%;left:5%"> 5) Min Cancellation Time</p>
    <p><input id="slider5" type="range" min="30" value="60" max="90" step="1" onchange="redrawWindow();" style="position:absolute;top:72.5%;le
    <p style="position:absolute;top:77.5%;left:5%"> 6) Likelihood of Cancellation</p>
    <p><input id="slider6" type="range" min="0.01" value="0.06" max="1" step="0.05" onchange="redrawWindow();" style="position:absolute;top:87
</div>
<div class = "legend-box">
    <p style="position:absolute;top:1%;left:5%"> Legend </p>
    <img src="images/House.png" width="30" height="30" style="position:absolute;top:20%;left:5%">
    <p style="position:absolute;top:17%;left:12%"> = Has not ordered </p>
    <img src="images/House_Ordered.png" width="30" height="30" style="position:absolute;top:40%;left:5%">
    <p style="position:absolute;top:37%;left:12%"> = Has ordered </p>
    <img src="images/Mcd.png" width="30" height="30" style="position:absolute;top:60%;left:5%">
    <p style="position:absolute;top:57%;left:12%"> = McDonald's Outlet </p>
    <img src="images/Building.png" width="30" height="30" style="position:absolute;top:80%;left:5%">
    <p style="position:absolute;top:77%;left:12%"> = Tree </p>
    <img src="images/Withoutorder.png" width="30" height="30" style="position:absolute;top:20%;left:50%">
    <p style="position:absolute;top:17%;left:57%"> = Has not received an order </p>
    <img src="images/Pickup.png" width="30" height="30" style="position:absolute;top:40%;left:50%">
    <p style="position:absolute;top:37%;left:57%"> = Picking up order </p>
    <img src="images/Delivery.png" width="30" height="30" style="position:absolute;top:60%;left:50%">
    <p style="position:absolute;top:57%;left:57%"> = Delivering order </p>
</div>
<div>
    <img src="images/logo1.png" width="150" height="150" style="position:absolute;top:20%;left:80%">
</div>
```