Model Documentation & Programming Logic

GRID: ACTS AS A CENTRAL MAP THAT TRACKS THE LOCATION OF ALL ENTITIES

CONTAINS HELPER METHODS

MAKE2DARRAY – INITIALIZES 2D GRID AS

ISEMPTY/ISFULL – CHECKS THE STATE

FILLLOCATION/FREELOCATION – MODIFIES SPECIFIED LOCATION

You, a few seconds ago | 1 author (You)

```javascript
class Grid {
    constructor(numRows, numCols, isempty=true) {
    this.numRows = numRows;
    this.numCols = numCols;
        this.locations = this.make2dArray(numRows, numCols, isempty);
    }

    isEmpty(location) {
        let row = location.row - 1;
        let col = location.col - 1;
        return this.locations[row][col];
    }

    isFull(location) {        You, a few seconds ago • Uncommitted changes
        return !this.isEmpty(location);
    }

    fillLocation(location) {
        let row = location.row - 1;
        let col = location.col - 1;
        this.locations[row][col] = false;
    }

    freeLocation(location) {
        let row = location.row - 1;
        let col = location.col - 1;
        this.locations[row][col] = true;
    }

  fillLocations(startRow, numRows, startCol, numCols) {
    for (let row = startRow; row < startRow + numRows; row++) {
            for (let col = startCol; col < startCol + numCols; col++) {
        let location = {"row": row, "col": col};
                this.fillLocation(location);
            }
        }
    }

    make2dArray(numRows, numCols, value) {
        let arr = new Array();
        for (let row = 0; row < numRows; row++) {
            arr[row] = new Array(numCols).fill(value);
        }
        return arr;
    }
}
```

```
133
    You, 4 hours ago | 1 author (You)
134 class NonCollidingArea  {
135     constructor(label,  numRows, numCols, grid, url,relativePosition,addRow,addCol,
136                 fillColor='white', outlineColor='black', outlineWidth=1) {
137
138     //super(label,  numRows,  numCols);
139     this.label = label
140     this.numRows = numRows
141     this.numCols = numCols
142
143     this.grid = grid;
144     this.url = url;
145
146     this.relativePosition = relativePosition
147     console.log(this.relativePosition.row)
148     this.addRow = addRow
149     this.addCol = addCol
150     this.position = insertPosition(this.relativePosition,this.addRow,this.addCol);
151     console.log(this.position.startRow)
152     this.grid.fillLocations(this.position.startRow, this.numRows, this.position.startCol, this.numCols,window.numRows);
153
154     }
155   }
156
157
```

NONCOLLIDINGAREA LINKS TO GLOBAL GRID – USED TO CREATE LAYOUT FOR SUPERMARKET

You, a minute ago | 1 author (You)

```
159 class NonCollidingAgent {
160     constructor(id, type, row, col, grid, url, timeEntered) {
161         this.id = id;
162         this.type = type;
163         this.location = {
164             "row": row,
165             "col": col,
166         }
167         this.grid = grid;
168         this.url = url;
169         this.timeEntered = timeEntered;
170         this.fillGrid();
171     }
172 
173     fillGrid() {
174         this.grid.fillLocation(this.location);
175     }
176 
177     freeGrid() {
178         this.grid.freeLocation(this.location);
179   }
180 
181   getWeights(row, col) {
182     // simple zoning, divide into 4 quarters
183     let nrows = this.grid.numRows;
184     let ncols = this.grid.numCols;
185     let zone;
```

NONCOLLIDINGAGENT LINKS TO GLOBAL GRID – USED TO CREATE MOVING CUSTOMERS

```
grid = new Grid(numRows, numCols);


let walls = new NonCollidingArea('Walls',scale(3),maxCols ,grid,"images/

let rightPole = new NonCollidingArea('rightPole',Math.ceil((10/23)*numRo
let leftPole = new NonCollidingArea('leftPole', Math.ceil((10/23)*numRow


let cashier1 = new NonCollidingArea('cashier1', Math.ceil((2/23)*numRows
let cashier2 = new NonCollidingArea('cashier2', scale(2), 2, grid,"image

let midLaneBlocker = new NonCollidingArea('midLaneBlocker', Math.ceil((5
let leftLaneBlocker = new NonCollidingArea('leftLaneBlocker', Math.ceil(

// Reference cashier
right_cashier = new NonCollidingArea('right_cashier', scale(2), 2, grid,
```

```
function addDynamicAgents() {

  //
  let arrivalApproved = false;

  if (nextArrivalTime == currentTime) {
    arrivalApproved = thinPoisson(thinRate);
    nextArrivalTime += generateDiscreteExpTime(rate);
  }

  if (arrivalApproved) {
    let initialRow = bottomRow - 1;
    let doorStartCol = 0;
    let doorLength = 3;
    let initialCol = Math.floor(Math.random() * doorLength + doorStartCol);

    let newcustomer = new NonCollidingAgent(1, "A", initialRow, initialCol, grid,"images/girl


    let customerType = Math.floor(Math.random()*5);
    switch (customerType) {
        case 0:
            newcustomer.type = "A";
            newcustomer.url = "images/girl.png" ;
        break;

        case 1 :
            newcustomer.type = "B";
            newcustomer.url = "images/boy.png" ;
        break;

        case 2:
            newcustomer.type = "C";
            newcustomer.url = "images/old-woman.png" ;
        break;

        case 3 :
            newcustomer.type = "D";
            newcustomer.url = "images/minion.png" ;
        break;
        case 4 :
            newcustomer.type = "E";
            newcustomer.url = "images/family.png" ;

    }
    customers.push(newcustomer);
```

USAGE

Define grid at the start, then add all static objects to define layout

Customers are created dynamically with each simulation step

Entities will automatically interact with grid to avoid collision

```
284
285         let direction = this.generateDirection(weights);
286         switch (direction) {
287             //up
288             case 0:
289                 this.up();
290                 break;
291             //down
292             case 1:
293                 this.down();
294                 break;
295             case 2:
296                 // stay
297                 break;
298             //left
299             case 3:
300                 this.left();
301                 break;
302             //right
303             case 4:
304                 this.right();
305                 break;
306             default:
307                 break;
308         }
309     }
310
```

```
329
330     up() {
331         this.freeGrid();
332         this.location.row -= 1;
333         this.fillGrid();
334     }
335
336     down() {
337         this.freeGrid();
338         this.location.row += 1;
339         this.fillGrid();
340     }
341
342     left() {
343         this.freeGrid();
344         this.location.col -= 1;
345         this.fillGrid();
346     }
347
348     right() {
349         this.freeGrid();
350         this.location.col += 1
351         this.fillGrid();
352     }
```

PROBABILISTIC DIRECTIONS GENERATED

MOVEMENTS LINKED TO GLOBAL GRID

```javascript
getWeights(row, col) {
  // simple zoning, divide into 4 quarters
  let nrows = this.grid.numRows;
  let ncols = this.grid.numCols;
  let zone;
  if (col <= right_cashier.position.startCol && col >= right_cashier.position.startCol -
        You, 5 hours ago • Exit condition, fix location errors, cashier zone, payment
    if (row == right_cashier.position.startRow - 6) {
      this.timeQueued = currentTime;
    }
    if (row == right_cashier.position.startRow) {
      this.timePaying = currentTime;
    }

    // queue zone
    if (row < right_cashier.position.startRow && row > right_cashier.position.startRow -
      console.log(row);
      console.log(right_cashier.position.startRow - 8, right_cashier.position.startRow);
      return [0, 5, 7, 1, 1]
    }

    // cashier zone
    if (row <= right_cashier.position.startRow + 2 && row >= right_cashier.position.star
      console.log(row);
      console.log(right_cashier.position.startRow, right_cashier.position.startRow + 3);
      return [0, 1, cashierDelay, 0, 0]
    }
  }
  if (row < Math.floor(nrows/2)) {
    // Upper
    if (col <= Math.floor(ncols/2)) {
      // Left
      zone = 0;
    } else {
      zone = 1;
    }
  } else {
    if (col <= Math.floor(ncols/2)) {
      // Left
      zone = 2;
    } else {
      zone = 3;
    }
  }
  switch (zone) {
    case 0:
      // upper left, more right
      return [1, 1, 2, 1, 2];
    case 1:
      // upper right, more down
      return [1, 7, 2, 1, 9.5];
    case 2:
      // lower left, more up, right
      return [3, 1, 2, 1, 2];
    case 3:
      // lower right, no more up
      return [0, 2, 5, 0.2, 0.2];
  }
```

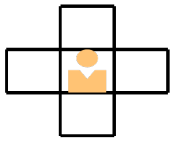
```javascript
generateDirection(weights) {
    let total = weights.reduce((x1, x2) => x1 + x2, 0); // i.e. [1, 4, 3, 2] => 10
    let normWeights  = weights.map(x => x / total); // i.e. [1, 4, 3, 2] => [0.1, 0.4, 0.3, 0.2]
    let cumulativeSum = [];
    for (let index in normWeights) {
        if (index == 0) {
            cumulativeSum[index] = normWeights[index];
        } else {
            cumulativeSum[index] = cumulativeSum[index - 1] + normWeights[index];
        }
    }
    let rng = Math.random();
    let direction = 0;
    while (rng > cumulativeSum[direction]) {
        direction += 1;
    }
    return direction;
}
```

However, weights are set to be non-colliding, hence directions generated are conditional on being non-colliding.

Non-Colliding Agent Movement Logic

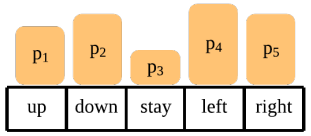
Visual	Logical Implementation in Javascript	Probability mass function (pmf)

Case 1: No non-colliding objects in agent's surroundings



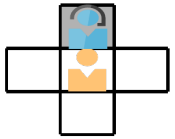
Case 1: The direction array contains its original direction weights

w_1	w_2	w_3	w_4	w_5
up	down	stay	left	right



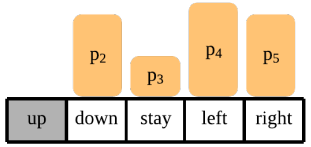
$$p_i = \frac{w_i}{\Sigma w_i}$$

Case 2: Non-colliding object(s) blocking certain direction(s)



Case 2: The "up" weight is set to 0, agent does not move up.

0	w_2	w_3	w_4	w_5
up	down	stay	left	right



$$p_i = \frac{w_i}{\Sigma w_i}, \quad w_1 = 0$$

The probabilites of moving in each direction follows the relative weights for each direction.

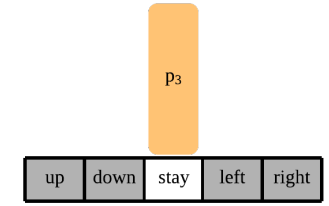
The pmf of each direction is then the direction weight divided by the sum of weights.

We generate the discrete pmf using the generalized inverse transform algorithmn from week 9.

Case 3: Non-colliding objects blocking agent in all directions

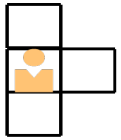
Case 3: All weights except "stay" are set to 0, agent just stays in place.

0	0	w_3	0	0
up	down	stay	left	right



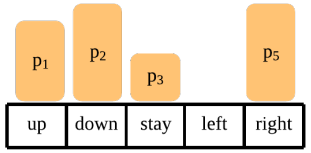
$$p_i = \frac{w_i}{\Sigma w_i}, \quad w_1, w_2, w_4, w_5 = 0$$

Case 4: Agent beside map's left border



Case 4: The weight in the direction of the border are set to 0, agent does not move left

w_1	w_2	w_3	0	w_5
up	down	stay	left	right



$$p_i = \frac{w_i}{\Sigma w_i}, \quad w_4 = 0$$

VISUAL MAPPING OF NON-COLLIDING LOGIC

```
}

function generateDiscreteExpTime(rate) {
  let U = Math.random();
  let time_delta = (-Math.log(1 - U)) / rate;
  let next_time = Math.max(1, Math.round(time_delta)) // ensure discrete time
  return next_time;
}

function thinPoisson(probAccept) {
  let U = Math.random();
  return probAccept > U;
}

function addDynamicAgents() {

  //
  let arrivalApproved = false;

  if (nextArrivalTime == currentTime) {
    arrivalApproved = thinPoisson(thinRate);
    nextArrivalTime += generateDiscreteExpTime(rate);
  }

  if (arrivalApproved) {
    let initialRow = bottomRow - 1;
    let doorStartCol = 0;
    let doorLength = 3;
    let initialCol = Math.floor(Math.random() * doorLength + doorStartCol);

```

POISSON PROCESS (APPROXIMATED TO DISCRETE DUE TO SIMULATION STEPS)

EXPONENTIAL TIME GENERATED DYNAMICALLY (TO PREVENT RAM BURSTING)

THINNING RATE DETERMINED BY SLIDER