Model Documentation & Programming Logic

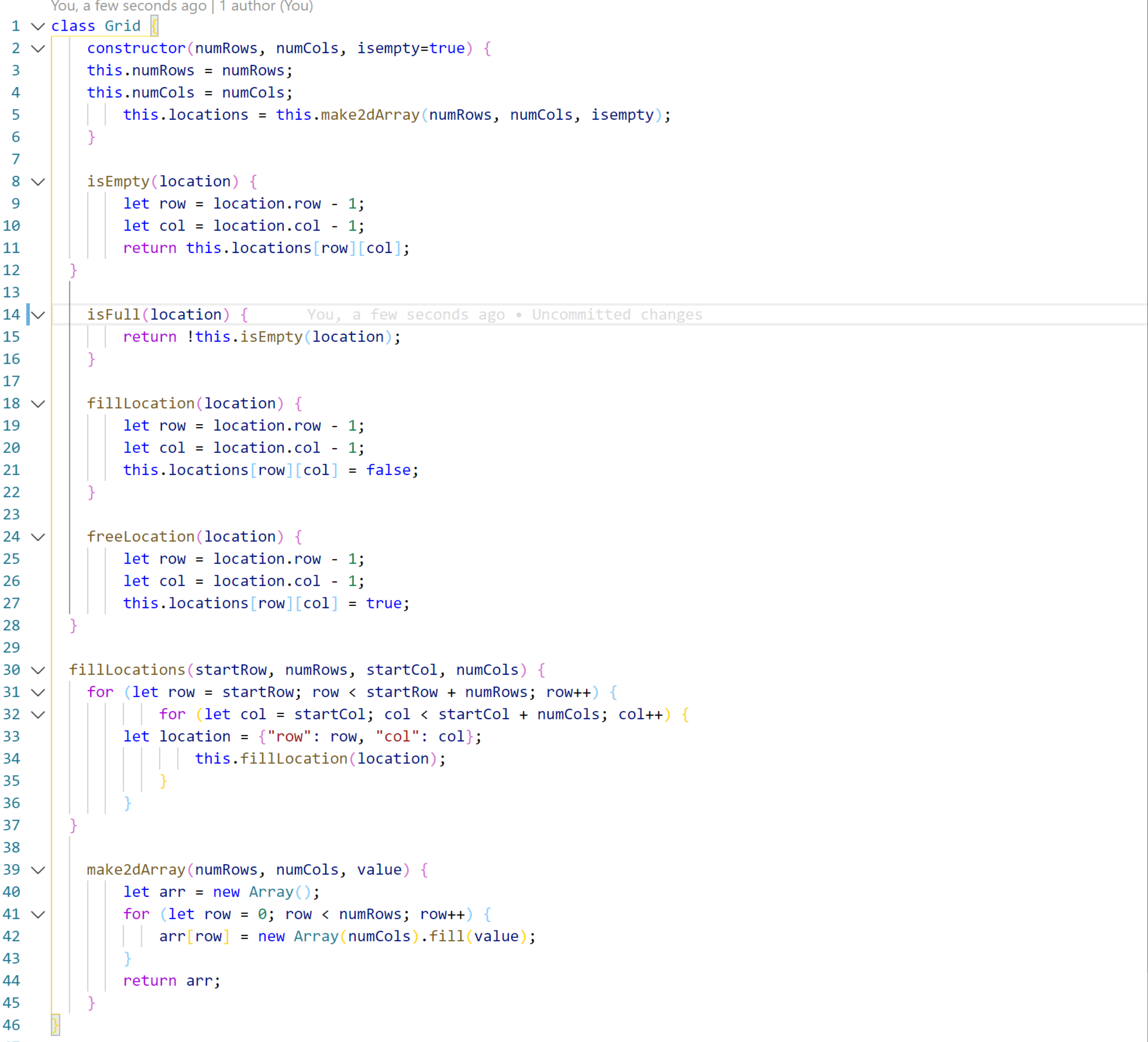
GRID: ACTS AS A CENTRAL MAP THAT TRACKS THE LOCATION OF ALL ENTITIES

CONTAINS HELPER METHODS

MAKE2DARRAY – INITIALIZES 2D GRID AS

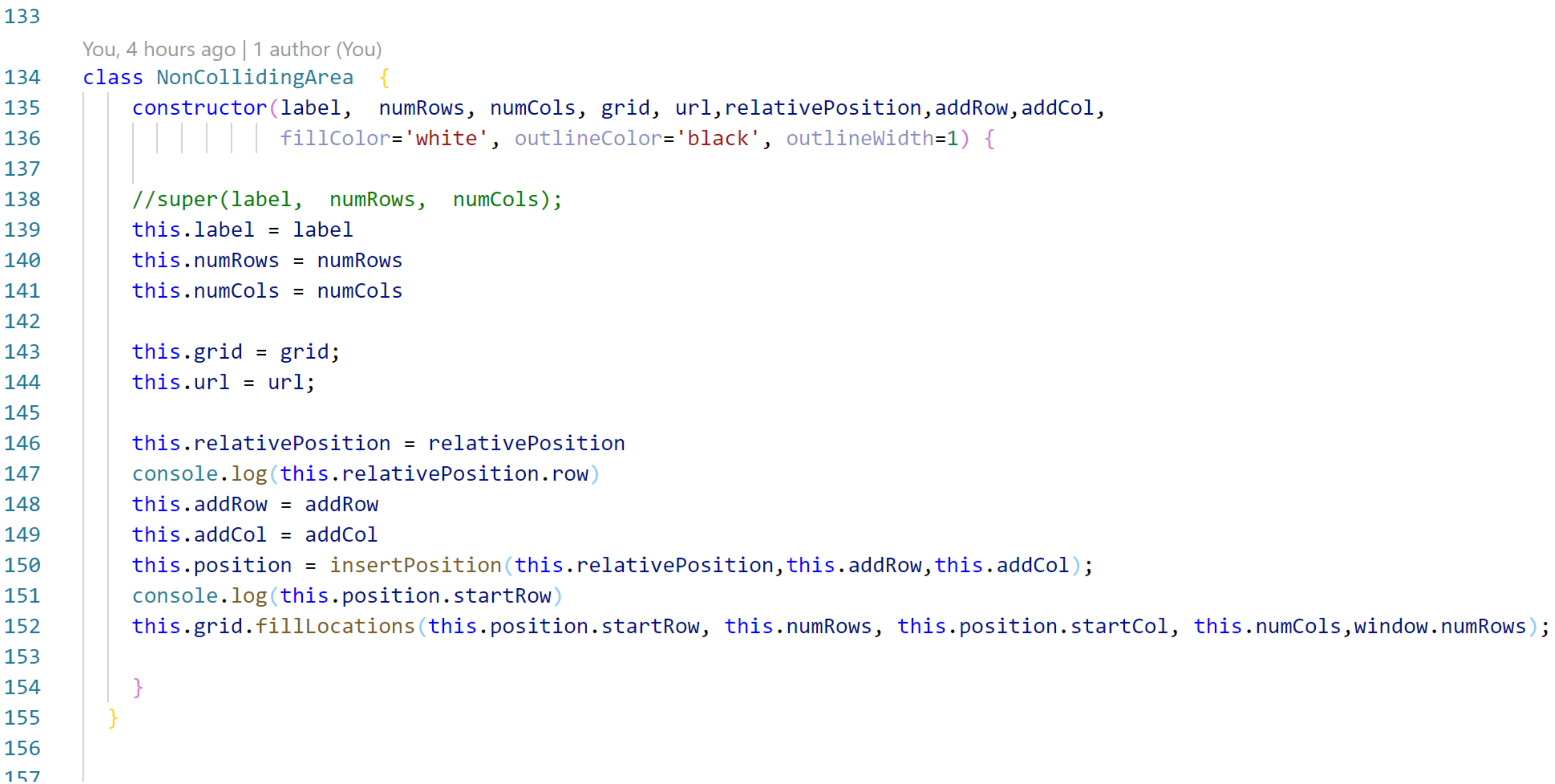
ISEMPTY/ISFULL – CHECKS THE STATE

FILLLOCATION/FREELOCATION – MODIFIES SPECIFIED LOCATION



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

```
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;
    }
}
```



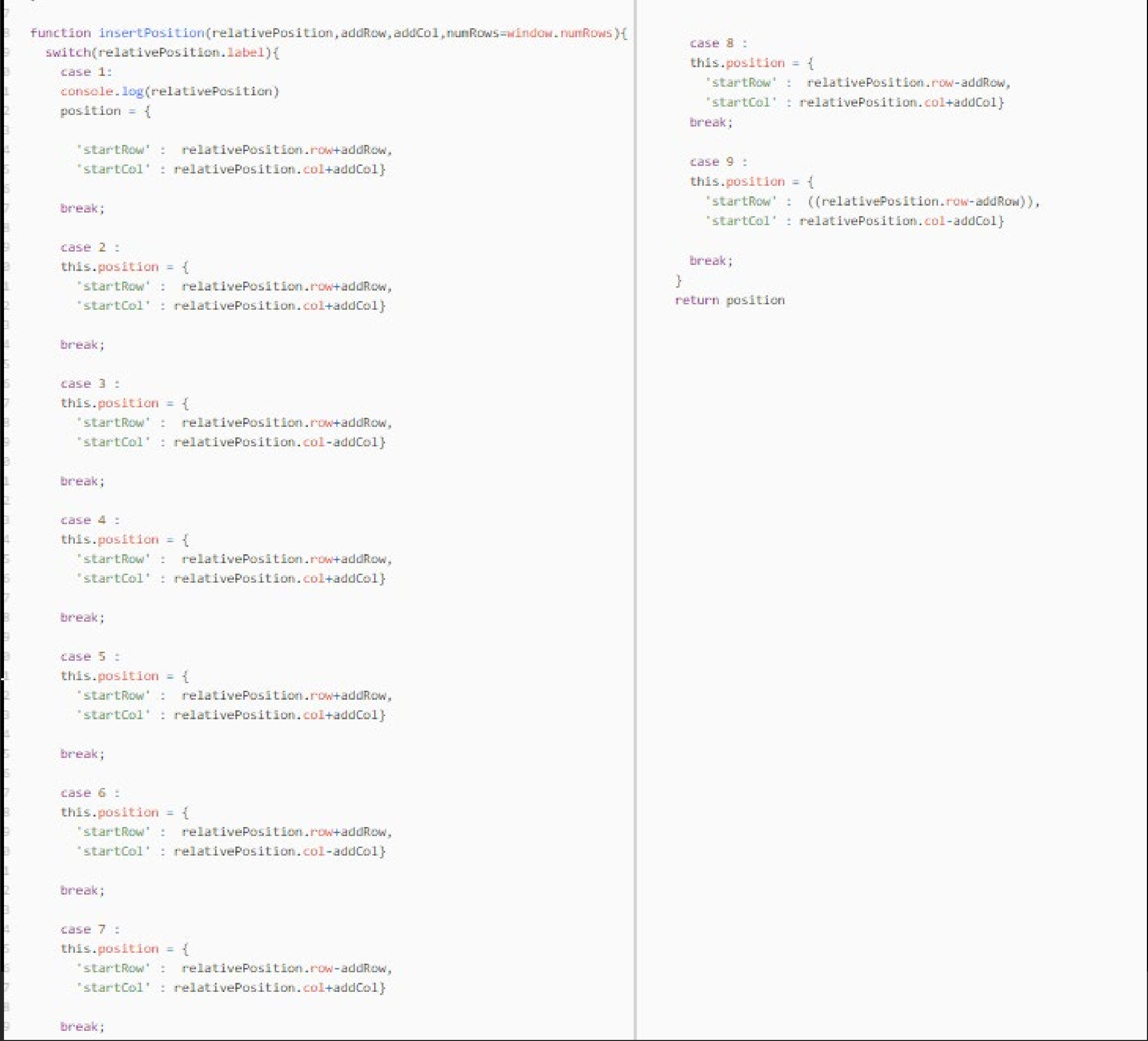
You, 4 hours ago | 1 author (You)

```
class NonCollidingArea  {
    constructor(label,  numRows, numCols, grid, url,relativePosition,addRow,addCol,
                fillColor='white', outlineColor='black', outlineWidth=1) {

    //super(label,  numRows,  numCols);
    this.label = label
    this.numRows = numRows
    this.numCols = numCols

    this.grid = grid;
    this.url = url;

    this.relativePosition = relativePosition
    console.log(this.relativePosition.row)
    this.addRow = addRow
    this.addCol = addCol
    this.position = insertPosition(this.relativePosition,this.addRow,this.addCol);
    console.log(this.position.startRow)
    this.grid.fillLocations(this.position.startRow, this.numRows, this.position.startCol, this.numCols,window.numRows);

    }
  }
```

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

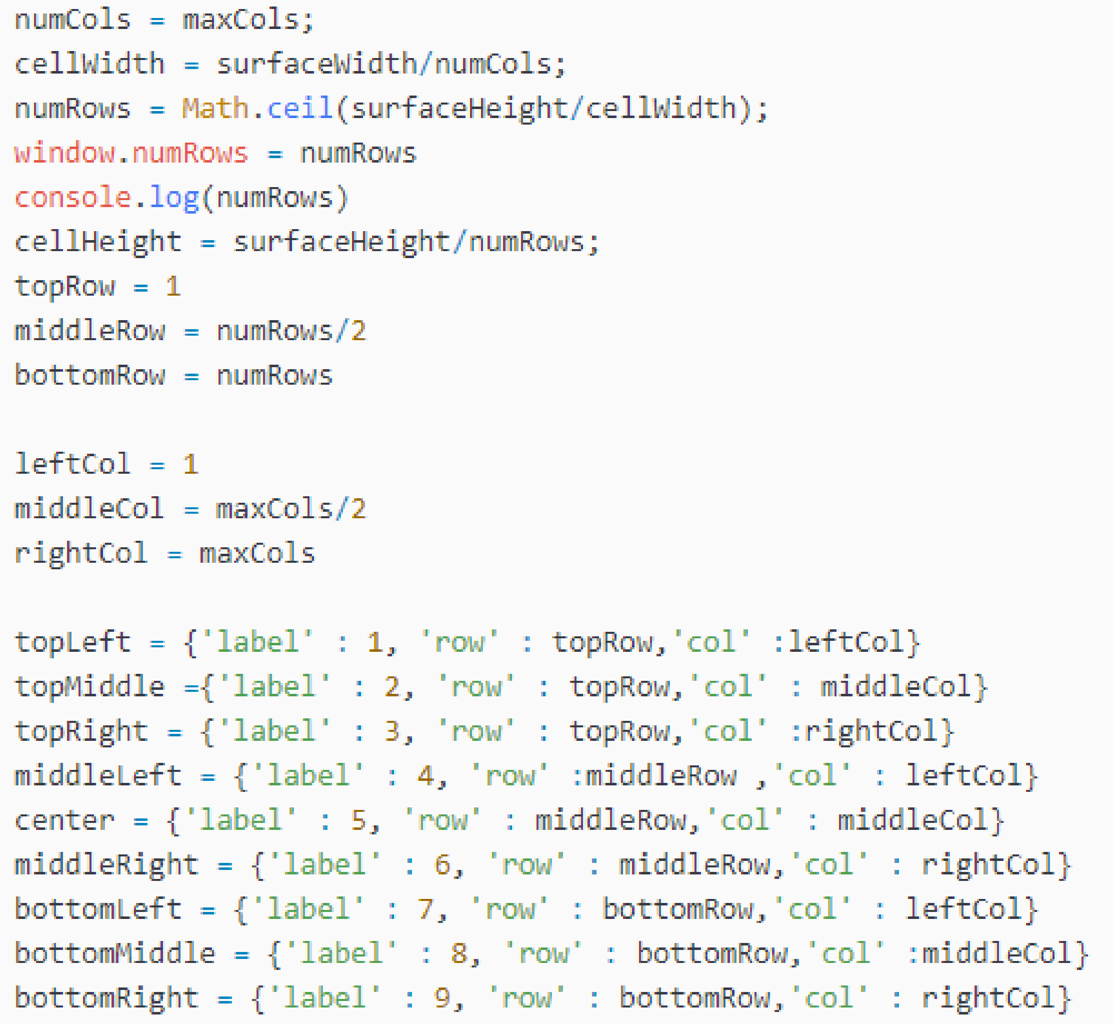
FIRST HELPER FUNCTION FOR THE POSITIONING OF THE NON-COLLIDING AREA IN THE LAYOUT OF THE SUPERMARKET

INSERTPOSITION- SPECIFIES THE START ROW AND START COLUMN BY USING THE RELATIVE POSITION AS THE INPUT



```
function insertPosition(relativePosition,addRow,addCol,numRows=window.numRows){
  switch(relativePosition.label){
    case 1:
    console.log(relativePosition)
    position = {

      'startRow' :  relativePosition.row+addRow,
      'startCol' : relativePosition.col+addCol}

    break;

    case 2 :
    this.position = {
      'startRow' :  relativePosition.row+addRow,
      'startCol' : relativePosition.col+addCol}

    break;

    case 3 :
    this.position = {
      'startRow' :  relativePosition.row+addRow,
      'startCol' : relativePosition.col-addCol}

    break;

    case 4 :
    this.position = {
      'startRow' :  relativePosition.row+addRow,
      'startCol' : relativePosition.col+addCol}

    break;

    case 5 :
    this.position = {
      'startRow' :  relativePosition.row+addRow,
      'startCol' : relativePosition.col+addCol}

    break;

    case 6 :
    this.position = {
      'startRow' :  relativePosition.row+addRow,
      'startCol' : relativePosition.col-addCol}

    break;

    case 7 :
    this.position = {
      'startRow' :  relativePosition.row-addRow,
      'startCol' : relativePosition.col+addCol}

    break;
```

```
    case 8 :
    this.position = {
      'startRow' :  relativePosition.row-addRow,
      'startCol' : relativePosition.col+addCol}
    break;

    case 9 :
    this.position = {
      'startRow' :  ((relativePosition.row-addRow)),
      'startCol' : relativePosition.col-addCol}

    break;
  }
  return position
```

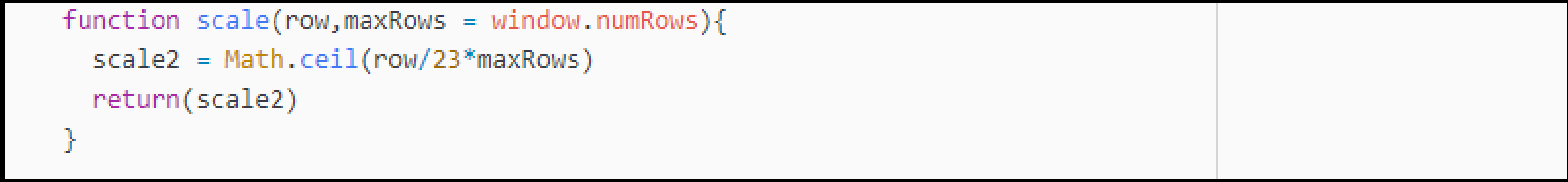


```
numCols = maxCols;
cellWidth = surfaceWidth/numCols;
numRows = Math.ceil(surfaceHeight/cellWidth);
window.numRows = numRows
console.log(numRows)
cellHeight = surfaceHeight/numRows;
topRow = 1
middleRow = numRows/2
bottomRow = numRows

leftCol = 1
middleCol = maxCols/2
rightCol = maxCols

topLeft = {'label' : 1, 'row' : topRow,'col' :leftCol}
topMiddle ={'label' : 2, 'row' : topRow,'col' : middleCol}
topRight = {'label' : 3, 'row' : topRow,'col' :rightCol}
middleLeft = {'label' : 4, 'row' :middleRow ,'col' : leftCol}
center = {'label' : 5, 'row' : middleRow,'col' : middleCol}
middleRight = {'label' : 6, 'row' : middleRow,'col' : rightCol}
bottomLeft = {'label' : 7, 'row' : bottomRow,'col' : leftCol}
bottomMiddle = {'label' : 8, 'row' : bottomRow,'col' :middleCol}
bottomRight = {'label' : 9, 'row' : bottomRow,'col' : rightCol}
```

RELATIVE POSITION WAS DECLARED AS THE INPUT FOR THE HELPER FUNCTION INSERTPOSITION

THE BOTTOMROW WAS SET TO BE THE MAXROW WHICH DEPENDS ON THE SIZE OF THE WINDOW

SECOND HELPER FUNCTION FOR THE POSITIONING OF THE NON-COLLIDING AREA IN THE LAYOUT

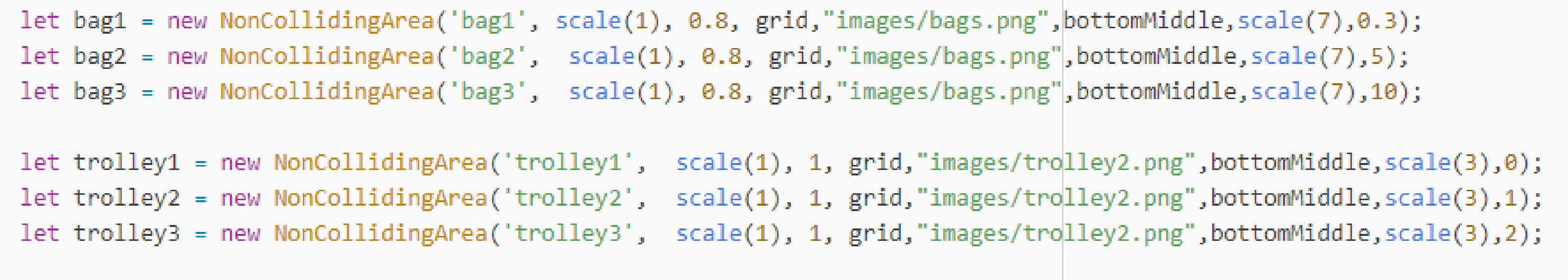


```
function scale(row,maxRows = window.numRows){
  scale2 = Math.ceil(row/23*maxRows)
  return(scale2)
}
```

SCALE- RETURNS THE APPROPRIATE NUMROWS BASED ON THE WINDOW SIZE

-THIS PREVENTS THE NUMROW THAT WE SPECIFIED TO BE UNDEFINED ON THE GRID DUE TO THE RESIZE OF THE WINDOW

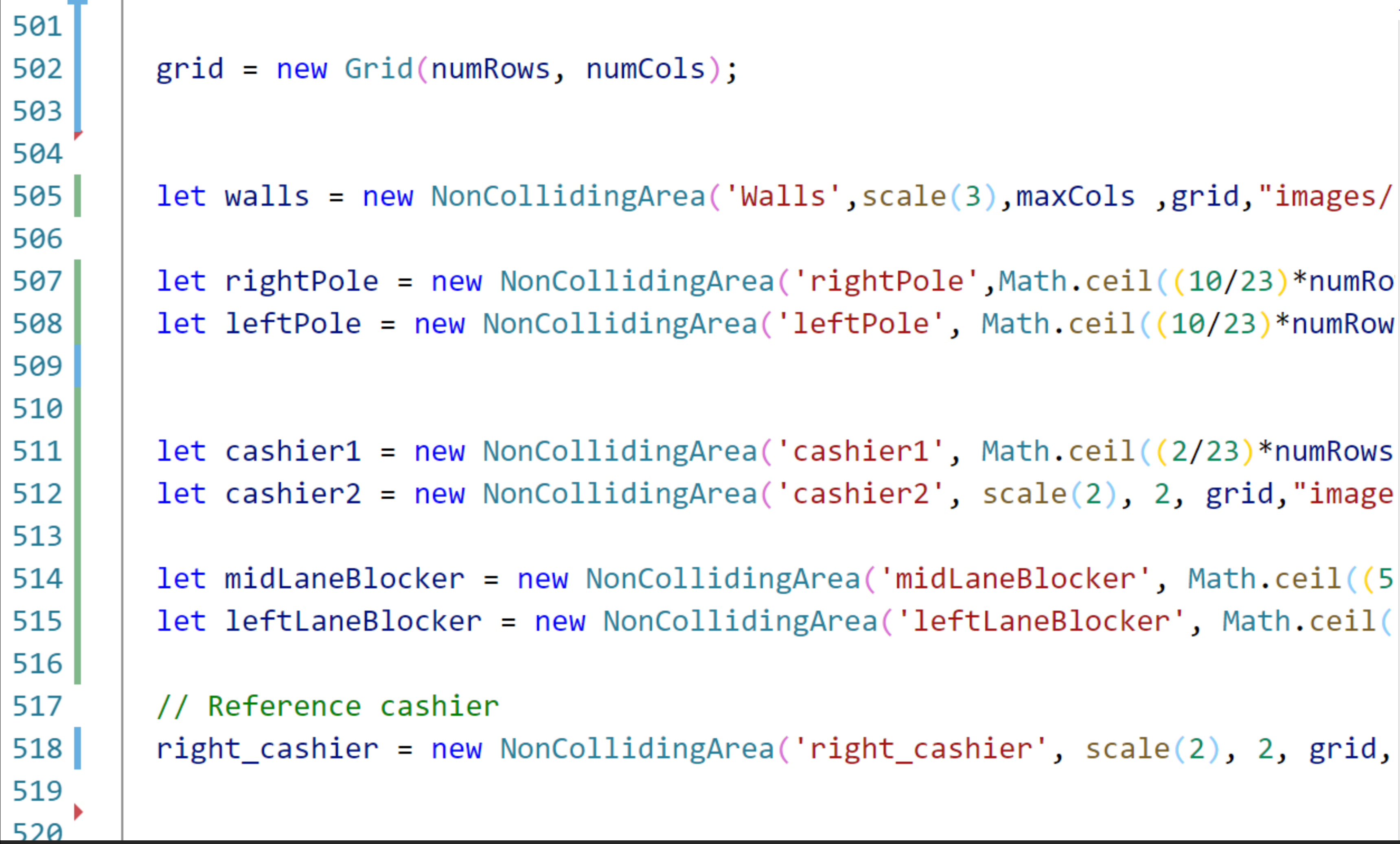
-ALLOWS THE IMAGE TO RESIZED ACCORDING TO THE WINDOW SIZE



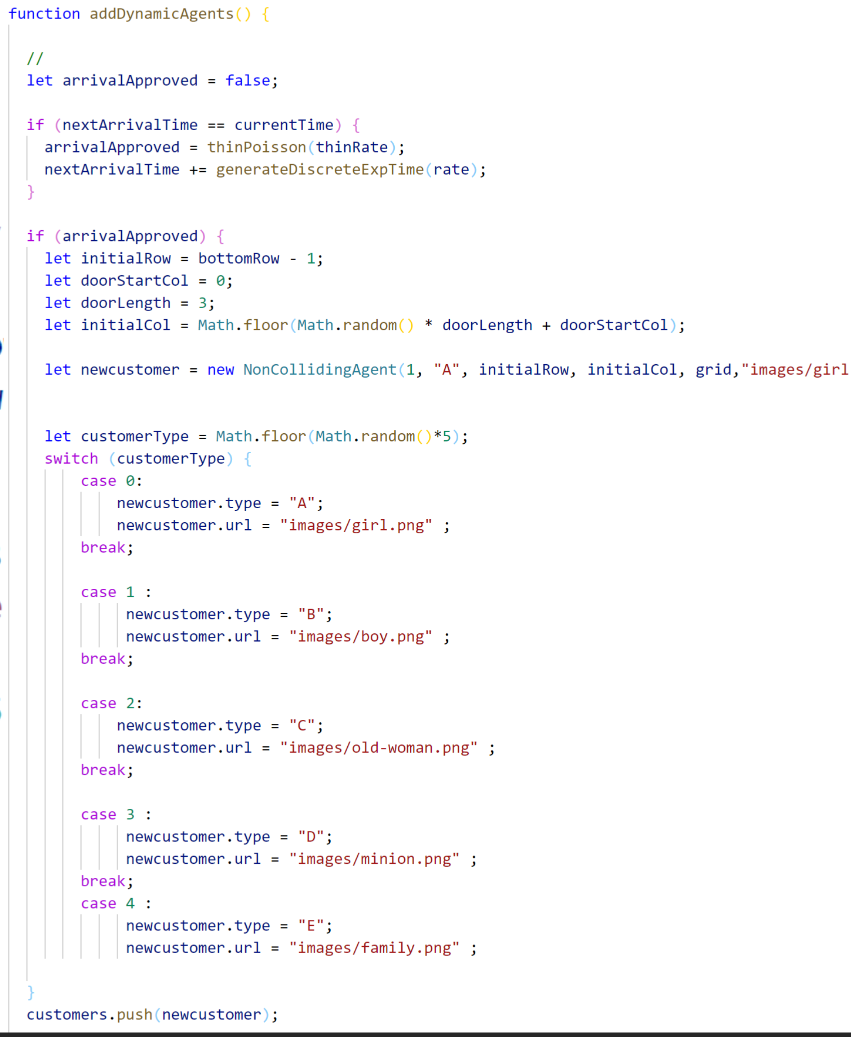
```
let bag1 = new NonCollidingArea('bag1', scale(1), 0.8, grid,"images/bags.png",bottomMiddle,scale(7),0.3);
let bag2 = new NonCollidingArea('bag2',  scale(1), 0.8, grid,"images/bags.png",bottomMiddle,scale(7),5);
let bag3 = new NonCollidingArea('bag3',  scale(1), 0.8, grid,"images/bags.png",bottomMiddle,scale(7),10);

let trolley1 = new NonCollidingArea('trolley1',  scale(1), 1, grid,"images/trolley2.png",bottomMiddle,scale(3),0);
let trolley2 = new NonCollidingArea('trolley2',  scale(1), 1, grid,"images/trolley2.png",bottomMiddle,scale(3),1);
let trolley3 = new NonCollidingArea('trolley3',  scale(1), 1, grid,"images/trolley2.png",bottomMiddle,scale(3),2);
```

THE SCALE FUNCTION IS USED FOR THE NUMROW AS SHOWN ABOVE

USAGE



```
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,
```

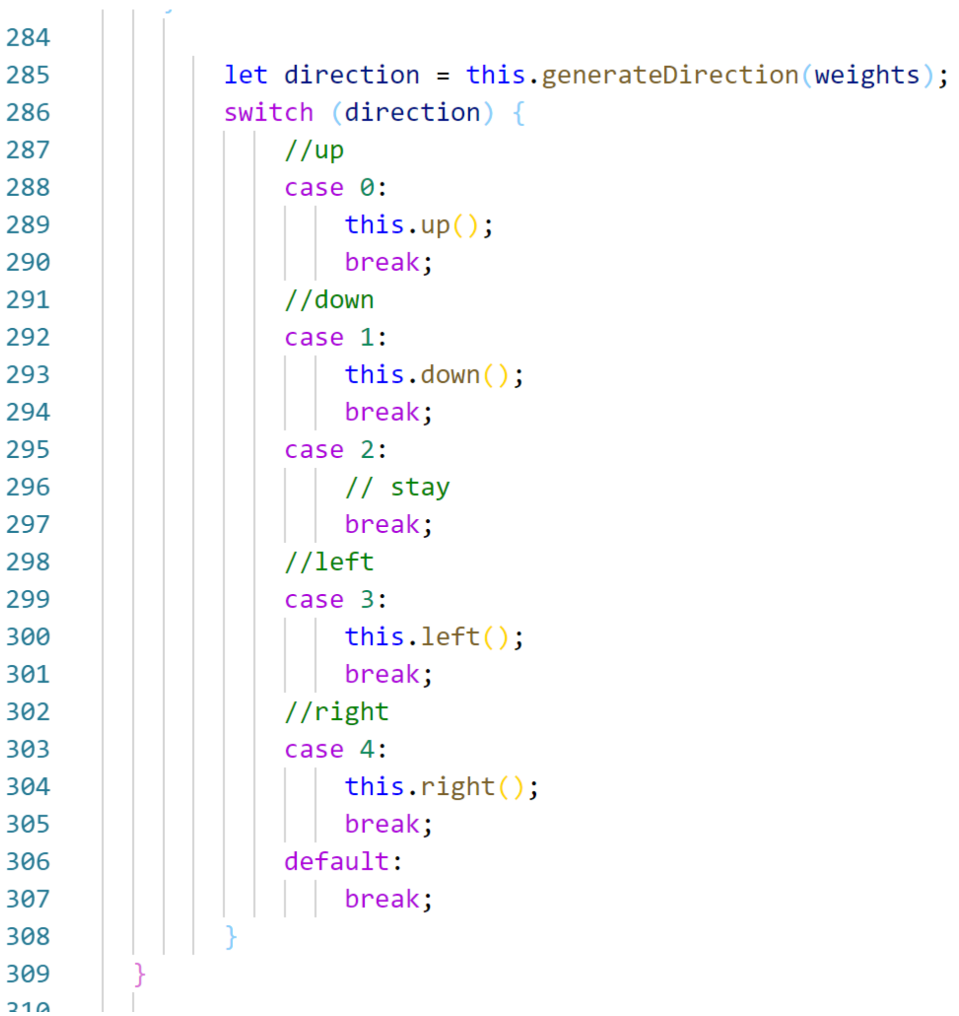


```
function addDynamicAgents() {

  //
  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);
```

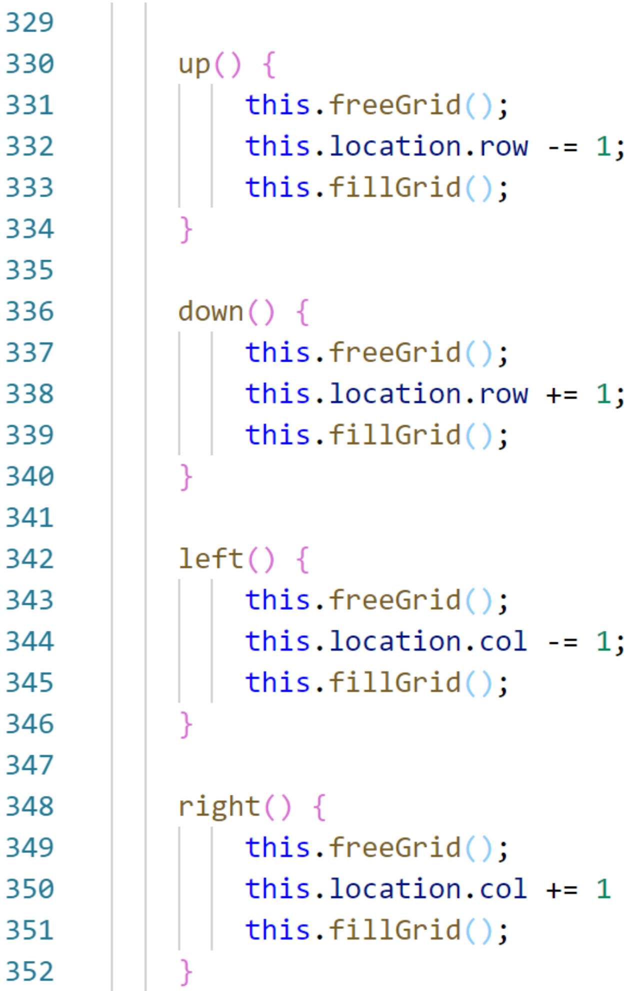
Define grid at the start, then add all static objects to define layout

Customers are created dynamically with each simulation step

Entities will automatically interact with grid to avoid collision



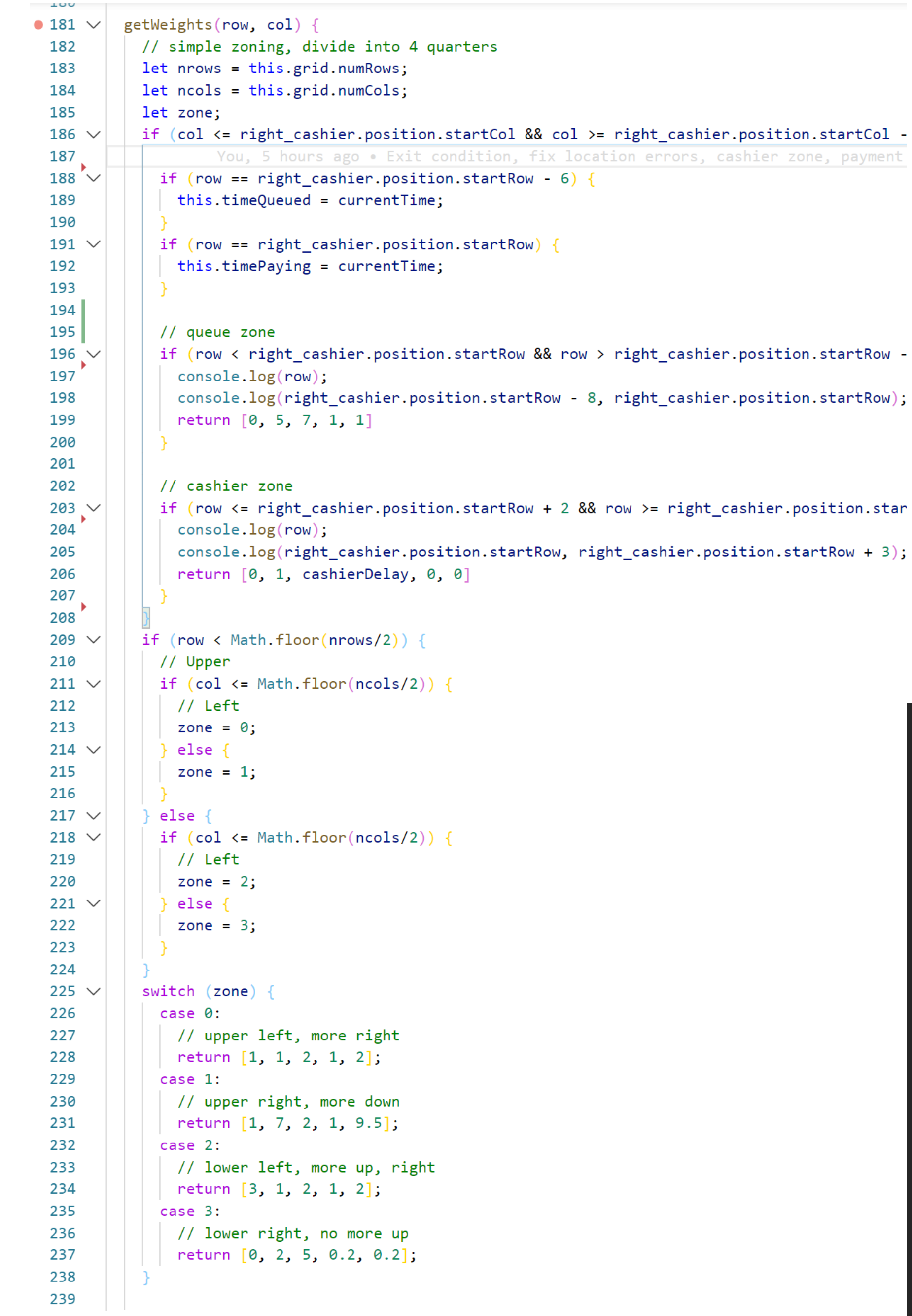
```

        let direction = this.generateDirection(weights);
        switch (direction) {
            //up
            case 0:
                this.up();
                break;
            //down
            case 1:
                this.down();
                break;
            case 2:
                // stay
                break;
            //left
            case 3:
                this.left();
                break;
            //right
            case 4:
                this.right();
                break;
            default:
                break;
        }
  }
```



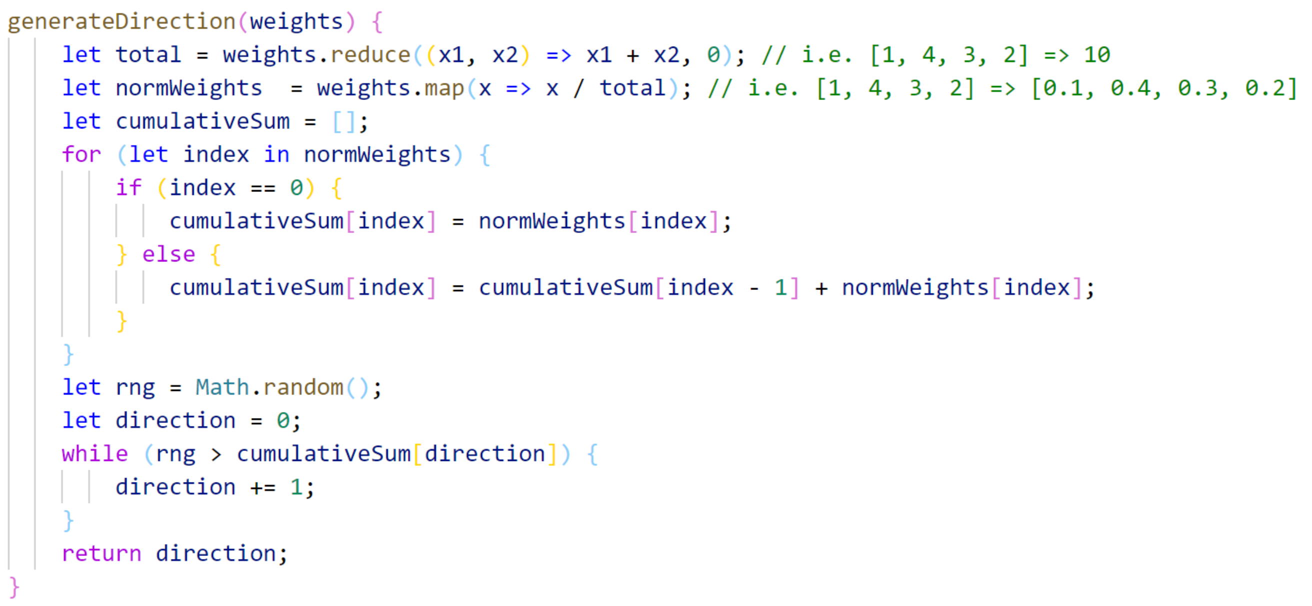
```

    up() {
        this.freeGrid();
        this.location.row -= 1;
        this.fillGrid();
    }

    down() {
        this.freeGrid();
        this.location.row += 1;
        this.fillGrid();
    }

    left() {
        this.freeGrid();
        this.location.col -= 1;
        this.fillGrid();
    }

    right() {
        this.freeGrid();
        this.location.col += 1
        this.fillGrid();
    }
```

PROBABILISTIC DIRECTIONS GENERATED

MOVEMENTS LINKED TO GLOBAL GRID



```
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];
  }

```



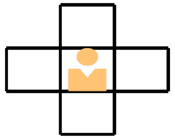
```
generateDirection(weights) {
    let total = weights.reduce((x1, x2) => x1 + x2, 0); // i.e. [1, 4, 3, 2] => 10
    let normWeights  = weights.map(x => x / total); // i.e. [1, 4, 3, 2] => [0.1, 0.4, 0.3, 0.2]
    let cumulativeSum = [];
    for (let index in normWeights) {
        if (index == 0)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

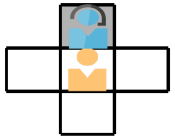
Non-Colliding Agent Movement Logic

Visual

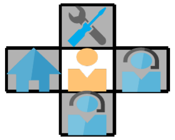
Case 1: No non-colliding objects in agent's surroundings



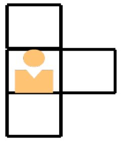
Case 2: Non-colliding object(s) blocking certain direction(s)



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



Case 4: Agent beside map's left border



Logical Implementation in Javascript

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

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

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

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

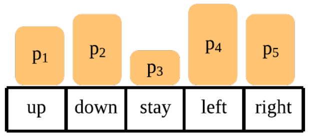
w_1	w_2	w_3	0	w_5
up	down	stay	left	right

Probability mass function (pmf)

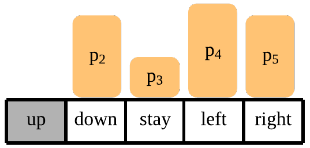
The probabilites of moving in each direction follows the relative weights for each direction.

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

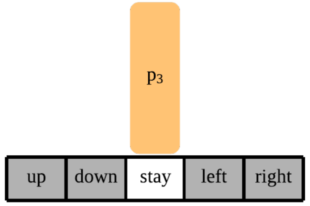
We generate the discrete pmf using the generalized inverse transform algorithmn from week 9.



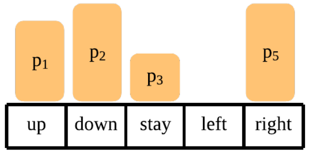
$$p_i = \frac{w_i}{\Sigma w_i}$$



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

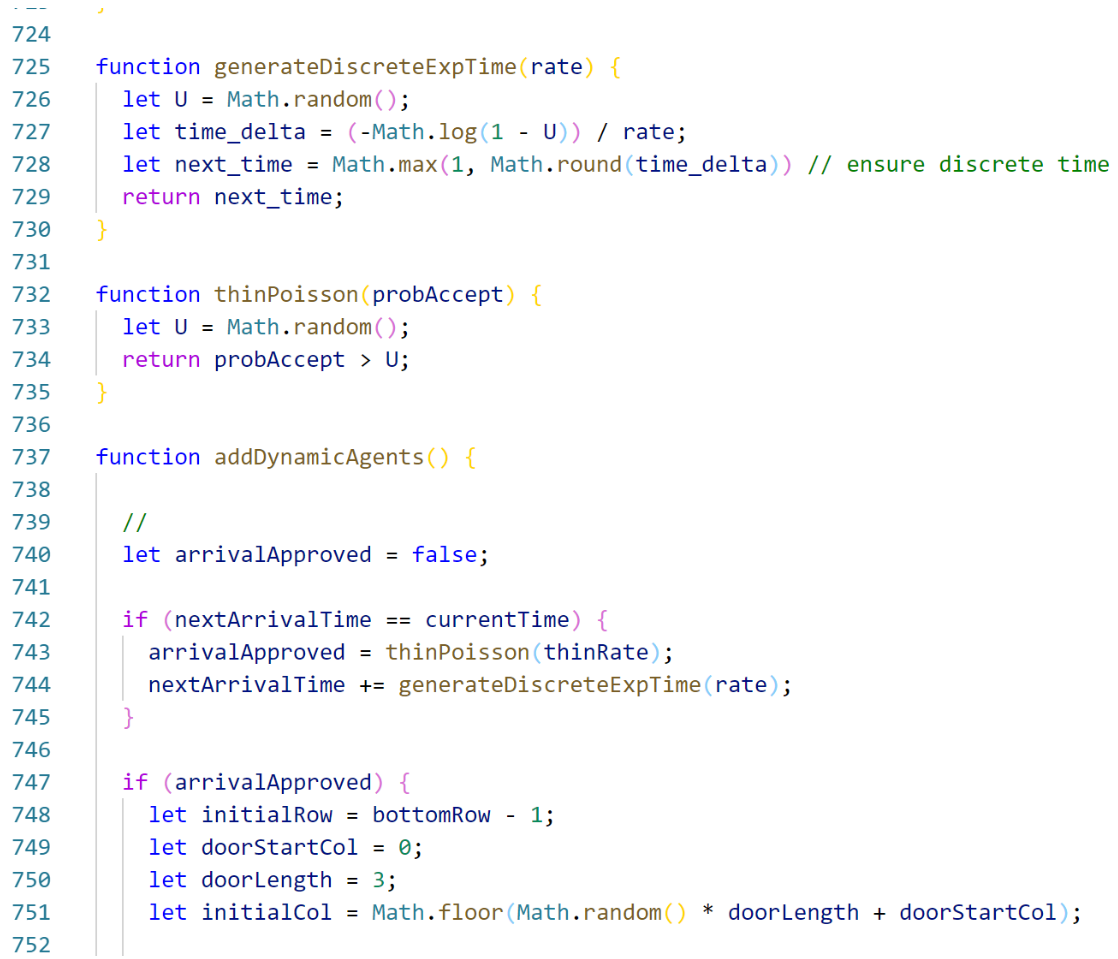


$$p_i = \frac{w_i}{\Sigma w_i} , \quad w_1, w_2, w_4 . w_5 = 0$$



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

VISUAL MAPPING OF NON-COLLIDING LOGIC



```javascript

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