TEAM 15

Model Documentation & Programming

Hospital Infection Model

Content Page

An overview of the components of the model code

- Initialization of the system
  - Constants and variables
  - Additional functions
- Agents
  - Healthcare Workers
  - Patients
- Update Simulation Display
- User Interface & Visualizations

Model Assumptions

General rules

- Infection is airborne
- Spread of infection is by proximity.
- The recovery or death of infected agents are not affected by other agents.
- All agents are susceptible to the infection at the start of the simulation.
- Agents may enter the system infected but undiagnosed.
- Undiagnosed infected agents can spread the infection.

Patients

- Uninfected patients have a higher chance of getting infected than healthcare workers as healthcare workers have safe practice protocols in place.
- Uninfected patients cannot die from their original cause of admission.
- Undiagnosed patients do not exit the simulation until they have been diagnosed.

Healthcare Workers

- Same team of workers remain in the system throughout the simulation.
- Not assigned to a specific ward for work.
- Cannot die from the infection, they remain in the system after recovering from the infection.
- After recovery, they are immune to it for a period of time.

Initialize the hospital system

Initialize environment, constants & variables

```
/////////////// Initialize variables and constants ////////////////

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 = 80
var cellWidth    //cellWidth is calculated in the redrawWindow function
var cellHeight   //cellHeight is calculated in the redrawWindow function
var numCols;     //numCols is calculated in the redrawWindow function
var numRows;     //numRows is calculated in the redrawWindow function

// heather added
const urlHealthyPatient = "images/YellowPatient.png";
const urlUndiagnosedPatient = "images/OrangePatient.png";
const urlInfectedPatient = "images/RedPatient.png"; // infected and diagnosed
const urlDischargedPatient = "images/GreenPatient.png";
const urlDeadPatient = "images/BlackPatient.png";
const urlHealthyDoctor = "images/GreenDoctor.png";
const urlUndiagnosedDoctor = "images/OrangeDoctor.png";
const urlInfectedDoctor = "images/RedDoctor.png";
const urlImmuneDoctor = "images/BlueDoctor.png";
const urlHealthyVisitor = "images/GreenVisitor.png";
const urlUndiagnosedVisitor = "images/OrangeVisitor.png";
// heather end
```

edited maxCols to 80 columns

referred from Prof's Code

link images for graphics of agents update icons

Initialize constants

```
// divya added
const exitRow = 13
const exitCol = 2 // leftmost
const deadCol = 78 // rightmost
const deadRow = 9 // center of top half of page
// divya end
```

specify rows and columns where agents leave the simulations

```
// divya added
// Patient States
const ENTER = 0;                // once entered system and not yet at ward
const STANDARDWARD = 1;         // at standard ward
const TOISOLATIONWAITING = 2;   // diagnosed and moving to isolation waiting
const ISOLATIONWAITING = 3;     // at isolation waiting area
const TOISOLATIONWARD = 4;      // diagnosed and moving to isolation ward
const ISOLATIONWARD = 5;        // reached isolation ward
const LEAVING = 6;              // once recovered
const EXITED = 7;               // once discharged or dead
// divya end
```

initialize patient states

```
// suzanne added
// Doctor States
const GOTOWARD = 0;        // going to a ward to treat patients
const TREATING = 1;        // treating a patient (stationary at ward)
const GOTOREST = 2;        // going to the communal area
const RESTING = 3;         // resting at the communal area
const GOTOISOLATION = 4;   // moving to the isolation ward (as they are infected)
const ISOLATED = 5;        // staying in the isolation ward (officially admitted as infected)
const GOTOWAIT = 6;        // going to isolation waiting room
const WAITING = 7;         // in the isolation waiting room
```

initialize healthcare worker states

```
//a ward can be AVAILABLE (has at least 1 bed available) or FULL (reached max capacity)
const AVAILABLE = 0;
const FULL = 1;
// suzanne end
```

initialize ward capacity states

Initialize variables

```
// Spontaneously created agents
// a dynamic list, initially empty
var patients = [];
var visitors = [];
var doctors = [];
var infectedwaitinglist = []

// Required ID variables
var nextPatientID = 0 // a value to ensure that each patient ha
var nextWaitingID = 1 // track the next unique value to be assi
var nextIsolationID = 1 // value for comparison to determine ne

/// Resources (to be toggled by user) ///
var numDoctors;         // total number of healthcare workers
var numBedsPerWard;     // ward capacity
```

initialize dynamic lists of agents

var **nextPatientID:**
a value to ensure that each patient has a unique ID number (increment before assignment)

resources to be toggled by user using sliders

Managing the isolation waiting area

Diagnosed infected agent is sent to the isolation waiting area and assigned a waiting ID

WAITING ID

var **nextWaitingID**

Represents the ticket system that issues the next patient their queue number

Updated (+ 1) whenever an infected agent is assigned the waiting id

When a bed is available in the isolation ward, check the nextIsolationID to see who is next in line

ISOLATION ID

var **nextIsolationID**

Represents the clinic display that shows the queue number of the next patient

Updated (+ 1) whenever an infected agent is assigned a bed in the isolation ward

Agent whose waiting ID = nextIsolationID gets assigned a place in the isolation ward

Managing the isolation waiting area

Diagnosed infected agent is sent to the isolation waiting area and assigned a waiting ID

WAITING LIST

infectedwaitinglist = [ ]

Each entry has:

{ "type": "doctor" OR "patient",

"id": agent's id number from agent list,

"waitingid": queue number }

When a bed is available in the isolation ward, check the nextIsolationID to see who is next in line

WAITING ID

var **nextWaitingID**

ISOLATION ID

var **nextIsolationID**

Define areas in the hospital

PER WARD

Define location of area

Define colour of area

Store capacity variable

Track the number of patients in the ward

```
var areas =[{ // areas[0]
    "label":"Standard Ward 1", // one ward at top left hand corner
    "startRow":2,
    "numRows":6,
    "startCol":4,
    "numCols":16,
    "color":"white",
    "capacity":numBedsPerWard,
    "numPatients":0
},
```

```
{ // areas[4]
    "label":"Standard Ward 2", // ward at bottom left hand corner
    "startRow":12,
    "numRows":6,
    "startCol":4,
    "numCols":16,
    "color":"white",
    "capacity":numBedsPerWard,
    "numPatients":0
},
```

```
{ // areas[2]
    "label":"Isolation Ward",
    "startRow":2,
    "numRows":6,
    "startCol":58,
    "numCols":20,
    "color":"white",
    "capacity":numBedsPerWard,
    "numPatients":0    // to count how many patients are currently in the ward
},
```

```
{ //areas[5]
    "label":"Standard Ward 3", // ward at bottom right hand corner
    "startRow":12,
    "numRows":6,
    "startCol":62,
    "numCols":16,
    "color":"white",
    "capacity":numBedsPerWard,
    "numPatients":0
},
```

Define areas in the hospital

WAITING AREA

Define location of area

Define colour of area

Track the number of patients in the ward

*Assume infinite capacity

```
{ // areas[1]
    "label":"Waiting Area",
    "startRow":2,
    "numRows":5,
    "startCol":42,
    "numCols":16,
    "color":"powderblue",
    "numPatients":0
},    // assume infinite capacity for waiting area
```

```
{   //areas[6]
    "label":"Entry point", // left side
    "startRow":5,
    "numRows":2,
    "startCol":2,
    "numCols":1,
    "color":"black",
},
{   //areas[7]
    "label":"Exit point", // left side
    "startRow":13,
    "numRows":2,
    "startCol":2,
    "numCols":1,
    "color":"black",
},
{   //areas[8]
    "label":"Dead point", // right side
    "startRow":9,
    "numRows":2,
    "startCol":78,
    "numCols":1,
    "color":"black",
}
];
```

ENTRY/EXIT POINTS & COMMUNAL AREA

Define location of area

Define colour of area

```
{ //areas[3]
    "label":"Communal Area",
    "startRow":10, // row 10 to 14
    "numRows":5,
    "startCol":28, // col 28 to 51
    "numCols":24,
    "color":"lightyellow"
},
```

Areas

```
var hospitalStartRow = areas[0].startRow - 1
var hospitalStartCol = areas[6].startCol
var hospitalEndRow = areas[4].startRow + areas[4].numRows + 1
var hospitalEndCol = areas[8].startCol + areas[8].numCols

var standardWard = areas[0]; // the standardWard is the first element of the areas array

var standardWards = [areas[0], areas[4], areas[5]]   //indexes 0, 1, 2 represent each ward
// in our code we define:
// var i = getRandomInt(0,2)
// var standardWard = standardWards[i]

var waitingArea = areas[1];

var isolationWard = areas[2];

var communalArea = areas[3]; // row 10 to 15, col 28 to 52

var standardWard2 = areas[4];

var standardWard3 = areas[5];

var entryPoint = areas[6];
```

specify rows and columns for the entire hospital area

store each area as a variable for easier access when coding

The defined hospital area serves as limits for agent movement

Initialize statistics

```
// Statistics Variables
var currentTime = 0
// At each simulation step we want to know how many individuals are infected,
// what is the number of patients turned away due to full capacity at standard and isolation ward,
// what is the total number of deaths
var statistics = [{
        "name":"Total Number of Acquired Infections: ",      // index 0
        "location":{
            "row":19,
            "col":24
        },
            "count":0
    },
    {
        "name":"No. of Patients Turned Away due to Full Capacity at Standard Ward: ",     // index 1
        "location":{
            "row":20,
            "col":24
        },
            "count":0
    },
    {
        "name":"No. of Patients Turned Away due to Full Capacity at Isolation Ward: ",    // index 2
        "location":{
            "row":21,
            "col":24
        },
        "count":0
    },
    {
        "name":"No. of Deaths: ",                                                        // index 3
        "location":{
            "row":23,
            "col":24
        },
        "count":0
    }, { // divya added: no. of people in waiting area at the present time
        "name": "No. of Patients at Waiting Area: ",                                     // index 4
        "location":{
            "row": 22,
            "col": 24
        },
        "count":0
    }
];
```

Initialize
currentTime = 0

Set up location of statistics

Statistic will be displayed near the last row of the hospital

At each simStep(), 5 statistics will be evaluated

EACH STATISTIC ENTRY

location
"row"
"col"

count
0

Variables that govern agent movement

PATIENTS

```
// The probability of a patient arriving, probArrivalP;
var probArrivalP;    // if generated no. < probArrival, arrival of patient occurs.
```

HEALTHCARE WORKERS

```
// The probability that the doctor is currently resting, probResting
// The probability that the doctor is currently treating a patient, probTreating
var probResting = 0.2; // rest at communal area
var probTreating = 0.7; // stay at ward location to treat patient
var probStandardTarget = 0.6 // probability that the doctor is going to a standard ward to administer treatment
                                                                    (instead of isolation ward)
```

Variables that govern infection

INFECTION & DIAGNOSIS

```
/////// For incoming patients or initialized doctor (initial state of health)
var probUndiagnosed = 0.1; // probability that an incoming patient or an initialized doctor is infected and undiagnosed

/////// For spread of infection and diagnosis of infection
// These variables define what is the probability of being undiagnosed (orange) when a notinfected citizen is near to an infected citizen.

// It also specifies the minimum distance for an infection opportunity to take place
var DistTransmission;
// once come into contact: infectionrate % chance of getting infected
var InfectionRate;  // Probability that someone gets infected upon close contact
// once infected: the rate of diagnosis governs whether the infected person is diagnosed or not

var DiagnosisRate; // Probability that an infected person gets diagnosed
```

RECOVERY, IMMUNITY & DEATH

```
// Doctors gain immunity to the infection after recovering
// This variable governs the probability that the immune doctors would lose their immunity at each time step
var probImmunityLoss = 0.01

// This variable defines what is the probability of recovering with no infection and exiting (for patient only), probRecoveryNo;
// and after getting infected and exiting, probRecoveryYes.
var probRecoveryNo = 0.7; // recover from injury aka reason for coming to hospital
var probRecoveryYes; // recover from hospital infection with no difference between doc and patient?
// heather end

// divya added
var probDeath = 0.2; // if generated no < probDeath, death of patient occurs.
// divya end
```

Define additional functions

OBTAIN SCREEN COORDINATES

```
/////////////// Additional defined functions /////////////
// The window is resizable, so we need to translate row and column coordinates into screen coordinates x and y
function getLocationCell(location) {
    var row = location.row
    var col = location.col
    var x = (col - 1) * cellWidth //cellWidth is set in the redrawWindow function
    var y = (row - 1) * cellHeight //cellHeight is set in the redrawWindow function
    return {
        "x": x,
        "y": y
    }
}
```

Referred from Prof's code

Define additional functions

```
// function to get a random integer between and inclusive of the defined upper and lower bounds
// used to randomly assign starting points and target locations
function getRandomInt(min, max) { // using whole ward as space instead of specific cell locations
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(Math.random() * (max - min + 1) + min);
}

// function to choose a random location in the wards (either standard or iso ward) for doctors to move to
// used to assign doctor to a target location for them to treat patients
function getRandomWardTarget(){

    var i = getRandomInt(0,2)
    var standardWard = standardWards[i]

    if (Math.random() < probStandardTarget){
        var wardRow = getRandomInt(standardWard.startRow, (standardWard.startRow + standardWard.numRows));
        var wardCol = getRandomInt(standardWard.startCol, (standardWard.startCol + standardWard.numCols));
    } else {
        var wardRow = getRandomInt(isolationWard.startRow, (isolationWard.startRow + isolationWard.numRows));
        var wardCol = getRandomInt(isolationWard.startCol, (isolationWard.startCol + isolationWard.numCols));
    }

    return [wardRow, wardCol]
}
```

Chooses a random integer between and inclusive of an input upper and lower bound

Chooses a random target location in one of the wards (for healthcare workers)

probStandardTarget is used to check if the doctor/nurse goes to the standard or isolation ward

Define additional functions

```
// function to find the next person in the waiting room that gets a spot in the isolation ward
// loops through infectedwaitinglist, checks if nextWaitingID == nextIsolationID
// finds the entry in the infectedwaitinglist where nextWaitingID == nextIsolationID
// stores that agent's "type" (patient or doctor), and original list id (patients.id or doctors.id)
function findNextIsoPerson(){

    for (let i = 0; i < infectedwaitinglist.length; i++){

        // initialize variables to store info
        var type;
        var id;
        var waitingID = infectedwaitinglist[i].waitingid;

        // find the next person to be assigned a spot in the isolation ward
        if (waitingID == nextIsolationID){
            type = infectedwaitinglist[i].type; // stores agent type
            id = infectedwaitinglist[i].id      // stores original id
        }
    }
    return [type, id]
    // when using function, call on findNextIsoPerson[0] and findNextIsoPerson[1] to get agent type and original id respectively
}
```

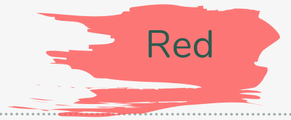
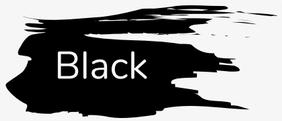
Finds the next infected agent in the waiting area that gets a place in the isolation ward

findNextIsoPerson[0] returns agent type

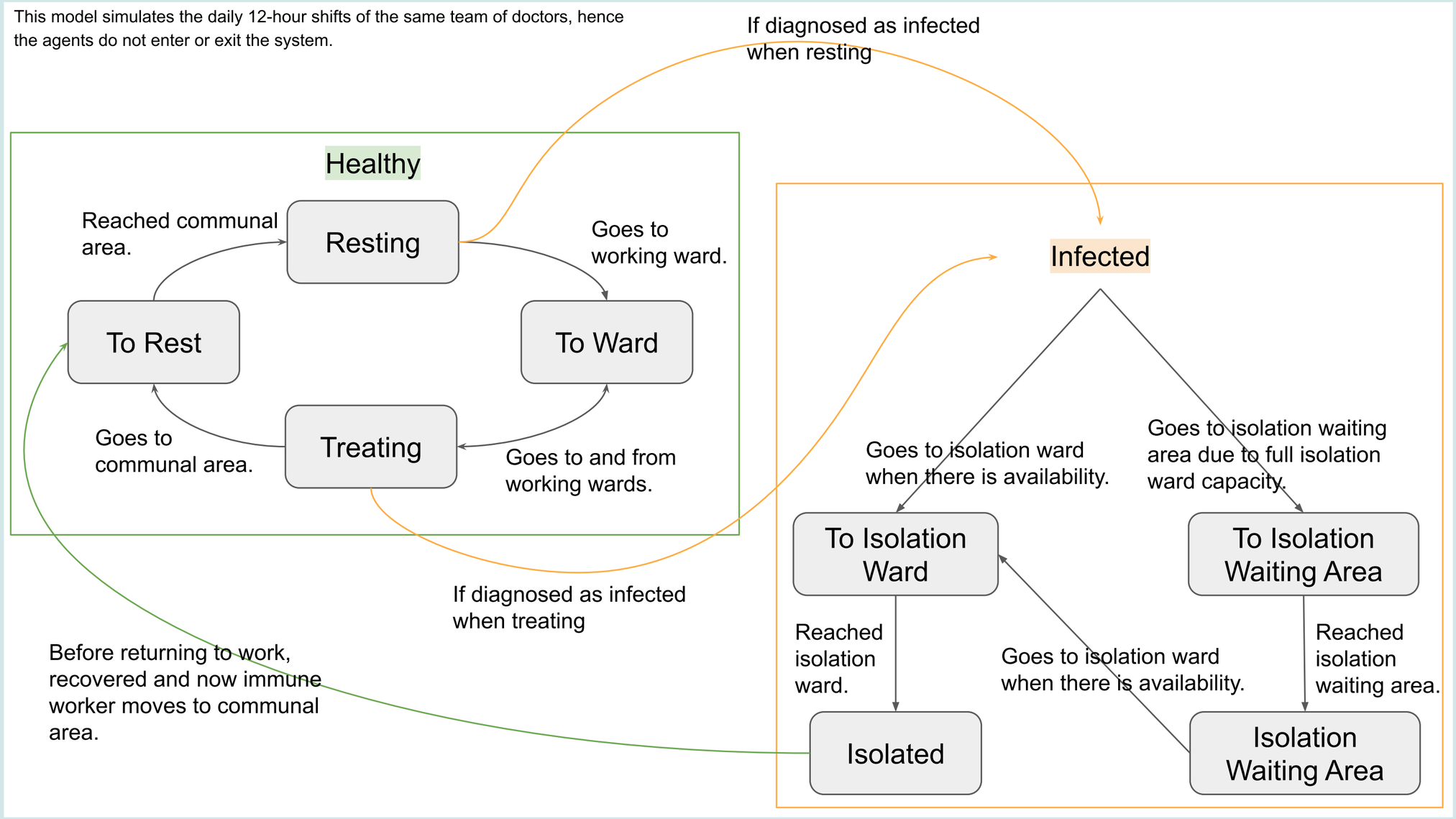
findNextIsoPerson[1] returns agent's ID

AGENTS:
Healthcare Workers & Patients

Agent states

COLOUR STATE	HEALTHCARE WORKER	PATIENT
	Infected and **Diagnosed**	Infected and **Diagnosed**
	Infected but **Undiagnosed**	Infected but **Undiagnosed**
	-	**Uninfected** (admitted for other medical conditions)
	**Healthy**	**Recovered** and Healthy
	Recovered and I**mmune**	-
	-	**Dead** (due to infection)

State diagram for healthcare workers



Initialize healthcare workers*

```
// initialize all doctor entries
for (let i = 0; i < numDoctors; i++) {    // loop through each doctor to add them to the dynamic list
    // random starting points
    var randomCol = getRandomInt(hospitalStartCol, hospitalEndCol);
    var randomRow = getRandomInt(hospitalStartRow, hospitalEndRow);

    // random target wards
    var ward = getRandomWardTarget();

    // create an entry for each healthcare worker as an entry in the doctors list variable
    var newdoctor = {
        "id":1+i,
        "type":"H",     // type: H for healthy, I for infected
        "location":{
            "row":randomRow,
            "col":randomCol
        },        // begin with all doctors being healthy
        "target":{
            "row":ward[0],
            "col":ward[1]
        },
        "state": GOTOWARD,
        "color":"green"
    };
    if (Math.random() < probUndiagnosed) {  // doctor is initially infected if the random number is less than the predefined probability
        newdoctor.type = "I";
        newdoctor.color = "orange";

    } else {
        newdoctor.type = "H";
        newdoctor.color = "green"
    };

    // add each healthcare worker as an entry in the doctors list variable
    doctors.push(newdoctor);      // "push" your new patient into the list of patients (adds a new entry in the patient table)
}
```

Position each doctor at random locations in the hospital

All doctors are initially working, with a target location in one of the wards

probUndiagnosed determines whether the doctor is initially infected

Add each doctor to the doctors list

Note: healthcare workers will be referred to as "doctors" for brevity

DOCTOR

id
1, 2, ..., numDoctors

type
"H": healthy,
"I": infected

location; target
"row"
"col"

state
GOTOWARD

color
"green": healthy,
"orange": infected & undiagnosed

*The same team of healthcare workers will remain in the system throughout, as we continuously simulate their daily shifts

updateDoctors()

To govern state changes of healthcare workers

```
// update individual healthcare worker entries
function updateDoctors(doctorIndex){
    // doctorIndex is an index into the healthcare workers data array
    doctorIndex = Number(doctorIndex);
    var doctor = doctors[doctorIndex];

    // get the current location of the healthcare worker
    var row = doctor.location.row;
    var col = doctor.location.col;
    var state = doctor.state;

    // determine if healthcare worker has arrived at the target
    var hasArrived = (Math.abs(doctor.target.row-row)+Math.abs(doctor.target.col-col))==0;
```

Variables that store information about the doctor of interest: doctor, row, col, state

Function applies to 1 doctor each time, identified by their index in the doctors list.

hasArrived checks if doctor has arrived at the target location

```
    // determine if immune doctor gets loses immunity at each time step
    // doctor.type remains as "H"
    if (doctor.color == "blue") {
        if (Math.random() < probImmunityLoss) {
            doctor.color = "green"        // healthy with no immunity to the infection
        }
    }
```

probImmunityLoss helps determine whether an immune doctor loses immunity at each time step

updateDoctors()

To govern infection of healthcare workers

```
//identify the doctors/patients infected
var infecteddoctors=doctors.filter(function(d){return d.type == "I";});
var infectedpatients=patients.filter(function(d){return d.type == "Infected" || "Undiagnosed";});
// combine them into a single array of all infected people for future use
var infectedpeople = infecteddoctors.concat(infectedpatients)
//console.log(infectedpeople)

//determine if any infected doctor/patient is nearby
// infection by proximity
i=0
if (infectedpeople.length > 0 && doctor.color == "green") {
    while (doctor.color == "green" && i < infectedpeople.length){
        var infected = infectedpeople[i];
        var infectedrow = infected.location.row
        var infectedcol = infected.location.col
        var distance = Math.sqrt((infectedrow-row)*(infectedrow-row)+(infectedcol-col)*(infectedcol-col))

        // determine if the doctor gets infected
        if (distance<DistTransmission){
            if (Math.random() < 0.1*InfectionRate) {
                doctor.type="I"
                doctor.color="orange"    // infected but undiagnosed
                statistics[0].count++
                //totnewinfected++
            }
         }
        i=i+1
    }
}
```

Consolidate an array of all infected agents

- Doctors: type = "I"
- Patients: type = "Infected" or "Undiagnosed"

Only healthy, non-immune doctors can get infected (color = "green")

If the doctor is susceptible to infection, calculate the distance of each infected agent from the doctor.

For each infected agent within the distance of transmission, there is a 0.1*InfectionRate chance of the doctor getting infected.

- 0.1 is multiplied as we assume that doctors have a lower chance of getting infected, due to the many hygiene protocols and PPE requirements in place.

updateDoctors()

To govern behaviour by state

```
// Behavior of doctor depends on his or her state
switch(state){
    case GOTOWARD:
        if (hasArrived){        // doctor has arrived at the ward

            // determine if undiagnosed infected doctor gets diagnosed upon arrival
            // doctor.type remains as "I" during change in color state
            if (doctor.color == "orange") {
                if (Math.random() < DiagnosisRate) {
                    doctor.color = "red"        // infected and diagnosed
                }
            }

            // check that doctor is not diagnosed as infected before treating patient
            if (!(doctor.color == "red")){      // doctor not red
                doctor.state = TREATING;
            }
            // if doctor is infected and diagnosed, send to isolation ward instead
            else if (doctor.color == "red") {   // doctor is red

                // check if isolation ward has available beds
                if (isolationWard.numPatients < isolationWard.capacity){
                    doctor.state = GOTOISOLATION;

                    // update numPatients (ie. infected doctor is assigned a spot in the isolation ward)
                    doctor.target.row = getRandomInt(isolationWard.startRow, (isolationWard.startRow + isolationWard.numRows - 1))
                    doctor.target.col = getRandomInt(isolationWard.startCol, (isolationWard.startCol + isolationWard.numCols - 1))
                    isolationWard.numPatients++
                }
                // else infected doctor will go to isolation waiting area
                else {
                    doctor.state = GOTOWAIT;
                    doctor.target.row = getRandomInt(waitingArea.startRow, (waitingArea.startRow + waitingArea.numRows - 1))
                    doctor.target.col = getRandomInt(waitingArea.startCol, (waitingArea.startCol + waitingArea.numCols - 1))

                    statistics[2].count++    // no. of infected people turned away due to full isolation ward

                    // add infected doctor to infectedwaitinglist
                    var newinfected = {
                        "id":doctor.id,
                        "type":"doctor",
                        "waitingid":nextWaitingID
                    };
                    nextWaitingID++
                    infectedwaitinglist.push(newinfected)
                }

            }

        }
    break;
```

Whenever an infected doctor arrives, we first check if they get diagnosed with the DiagnosisRate

If diagnosed, their colour state changes to red.

Statistics counts, variables and new target locations are updated accordingly

GOTOWARD

Doctor is moving towards a target ward location

If (hasArrived), the doctor is at a ward

TREATING

Doctor proceeds to treat patients as long as they are not diagnosed as infected

GOTOISOLATION

Doctor is sent to the isolation ward, if there are available beds

If diagnosed as infected

GOTOWAIT

Otherwise, they are sent to the isolation waiting area

Doctor is added to the list of people in the waiting area, and assigned a waiting id

updateDoctors()
To govern behaviour by state

```
case TREATING:
    if (hasArrived){                // doctor is treating patients at a ward

        // doctor will stay in the ward for a period of time then leave to next destination

        // if doctor has finished treatment
        if (Math.random() > probTreating) {

            // doctor will go to communal area to rest next
            if (Math.random() < probResting){
                doctor.state = GOTOREST;
                doctor.target.row = getRandomInt(communalArea.startRow, (communalArea.startRow + communalArea.numRows - 1))
                doctor.target.col = getRandomInt(communalArea.startCol, (communalArea.startCol + communalArea.numCols - 1))
            }
            // else doctor will get another target ward location to move to
            else {
                doctor.state = GOTOWARD;
                var ward = getRandomWardTarget()
                doctor.target.row = ward[0]
                doctor.target.col = ward[1]
            }
        }

    }
break;
```

Next target location is defined if doctor transitions to another state

At each time step, there is a probTreating chance that the doctor stays

TREATING

Doctor is at a ward, treating patients.

Will stay in the ward for a period of time before leaving.

If treatment is complete (ie. doctor is not staying)

GOTOREST

There is a probResting chance that the doctor goes to rest at the communal area

GOTOWARD

Otherwise, the doctor goes to another ward location to continue work

updateDoctors()

To govern behaviour by state

```
case GOTOREST:
    if (hasArrived){            // doctor has arrived at communal area

        // determine if undiagnosed infected doctor gets diagnosed upon arrival
        // doctor.type remains as "I" during change in color state
        if (doctor.color == "orange") {
            if (Math.random() < DiagnosisRate) {
                doctor.color = "red"        // infected and diagnosed
            }
        }

        // check that doctor is not diagnosed as infected
        if (!(doctor.color == "red")) {    // doctor not red
            doctor.state = RESTING
        }
        // if doctor is diagnosed as infected, go to isolation ward
        else if (doctor.color == "red") {  // doctor is red

            // check if isolation ward has available beds
            if (isolationWard.numPatients < isolationWard.capacity){
                doctor.state = GOTOISOLATION;
                doctor.target.row = getRandomInt(isolationWard.startRow, (isolationWard.startRow + isolationWard.numRows - 1))
                doctor.target.col = getRandomInt(isolationWard.startCol, (isolationWard.startCol + isolationWard.numCols - 1))
                isolationWard.numPatients++   // infected doctor is assigned a bed in the isolation ward
            }
            // else infected doctor will go to isolation waiting area
            else {
                doctor.state = GOTOWAIT;
                doctor.target.row = getRandomInt(waitingArea.startRow, (waitingArea.startRow + waitingArea.numRows - 1))
                doctor.target.col = getRandomInt(waitingArea.startCol, (waitingArea.startCol + waitingArea.numCols - 1))

                statistics[2].count++     // no. of infected people turned away due to full isolation ward

                // add infected doctor to infectedwaitinglist
                var newinfected = {
                    "id":doctor.id,
                    "type":"doctor",
                    "waitingid":nextWaitingID
                };
                nextWaitingID++
                infectedwaitinglist.push(newinfected)

            }
        }
    }
break;
```

Whenever an infected doctor arrives, we first check if they get diagnosed with the DiagnosisRate

Statistics counts, variables and new target locations are updated accordingly

GOTOREST

Doctor is moving to the communal area

If (hasArrived), the doctor is at the communal area.

RESTING

Doctor proceeds to rest as long as they are not diagnosed as infected

GOTOISOLATION

Doctor is sent to the isolation ward, if there are available beds

If diagnosed as infected

GOTOWAIT

Otherwise, they are sent to the isolation waiting area

Doctor is added to the list of people in the waiting area, and assigned a waiting id

updateDoctors()
To govern behaviour by state

```
case RESTING:
    if (hasArrived){                    // doctor is resting at communal area

        // if doctor will not continue resting
        if (Math.random() > probResting) {
            doctor.state = GOTOWARD;
            var ward = getRandomWardTarget()
            doctor.target.row = ward[0]
            doctor.target.col = ward[1]
        }
    }
break;
```

New target locations are updated accordingly (if any)

At each time step, there is a probResting chance that the doctor continues resting

RESTING

Doctor is at the communal area, resting.

Will stay in the communal area for a period of time before leaving.

GOTOWARD

Otherwise, the doctor goes to another ward location to continue work

updateDoctors()
To govern behaviour by state

```
case GOTOISOLATION:
    if (hasArrived){                // infected doctor has arrived at isolation ward
        doctor.state = ISOLATED

        // isolationWard.numPatients has been updated in the other states, upon assignment of a bed in the isolation ward

    }
break;
case ISOLATED:
    if (hasArrived){

        // if recovered, doctor goes back to work (but let them go to communal rest area first)
        if (Math.random() < probRecoveryYes){
            doctor.state = GOTOREST;
            doctor.target.row = getRandomInt(communalArea.startRow, (communalArea.startRow + communalArea.numRows - 1))
            doctor.target.col = getRandomInt(communalArea.startCol, (communalArea.startCol + communalArea.numCols - 1))

            doctor.type = "H"       // doctor is now healthy
            doctor.color = "blue"   // doctor is immune upon recovery

            // decrement number of infected people in isolation ward
            isolationWard.numPatients--
        }

    }
break;
```

Statistics counts, variables and new target locations are updated accordingly

GOTOISOLATION

Infected doctor is moving towards the isolation ward

If (hasArrived), the infected doctor is at the isolation ward

ISOLATED

Infected doctor is at the isolation ward

Will stay in the isolation ward until recovery

Infected doctor has a probRecoveryYes chance of recovering

GOTOREST

updateDoctors()
To govern behaviour by state

```
    case GOTOWAIT:
        if (hasArrived){                          // infected doctor has arrived at isolation waiting area
            doctor.state = WAITING

            /// update number of patients in isolation waiting area in stats (if we want to include this in our stats display)
            statistics[4].count++;

        }
    break;
```

Statistics count is updated accordingly

GOTOWAIT

Infected doctor is moving to the isolation waiting area

If (hasArrived), the infected doctor is at the isolation waiting area

updateDoctors()
To govern behaviour by state

```
case WAITING:                                   // infected doctor is waiting in the isolation waiting area
    if (hasArrived){

        // check if isolation ward has available beds
        if (isolationWard.numPatients < isolationWard.capacity){

            var nextIsoPerson = findNextIsoPerson()

            if (nextIsoPerson[0] == "doctor" && doctor.id == nextIsoPerson[1]){

                doctor.state = GOTOISOLATION;
                doctor.target.row = getRandomInt(isolationWard.startRow, (isolationWard.startRow + isolationWard.numRows - 1))
                doctor.target.col = getRandomInt(isolationWard.startCol, (isolationWard.startCol + isolationWard.numCols - 1))
                isolationWard.numPatients++   // infected doctor is assigned a bed in the isolation ward

                // update number of patients in isolation waiting area (-1)
                statistics[4].count--;

                // remove this infected doctor from the infectedwaitinglist
                infectedwaitinglist = infectedwaitinglist.filter(function (d) {
                    return d.waitingid != nextIsolationID
                })

                // update nextIsolationID
                nextIsolationID++

            }

        }

    }
break;
```

findNextIsoPerson() obtains the type and id of the next infected agent to get an isolation ward bed

Queuing system is updated

Statistics counts, variables and new target locations are updated accordingly

WAITING

Infected doctor is at the isolation waiting area, waiting for a bed in the isolation ward

Will stay queuing in the waiting area, until there is an availability and they are next in line

GOTOISOLATION

When there are available beds at the isolation ward, and this doctor is the next in line according to the queuing system

updateDoctors()
To govern movement of healthcare workers

```
    // set the destination row and column
    var targetRow = doctor.target.row
    var targetCol = doctor.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 patient
    doctor.location.row = newRow
    doctor.location.col = newCol

}
```

At the end of the updateDoctors() function, and outside the switch(state) section, we determine the next step that the doctor agent takes.

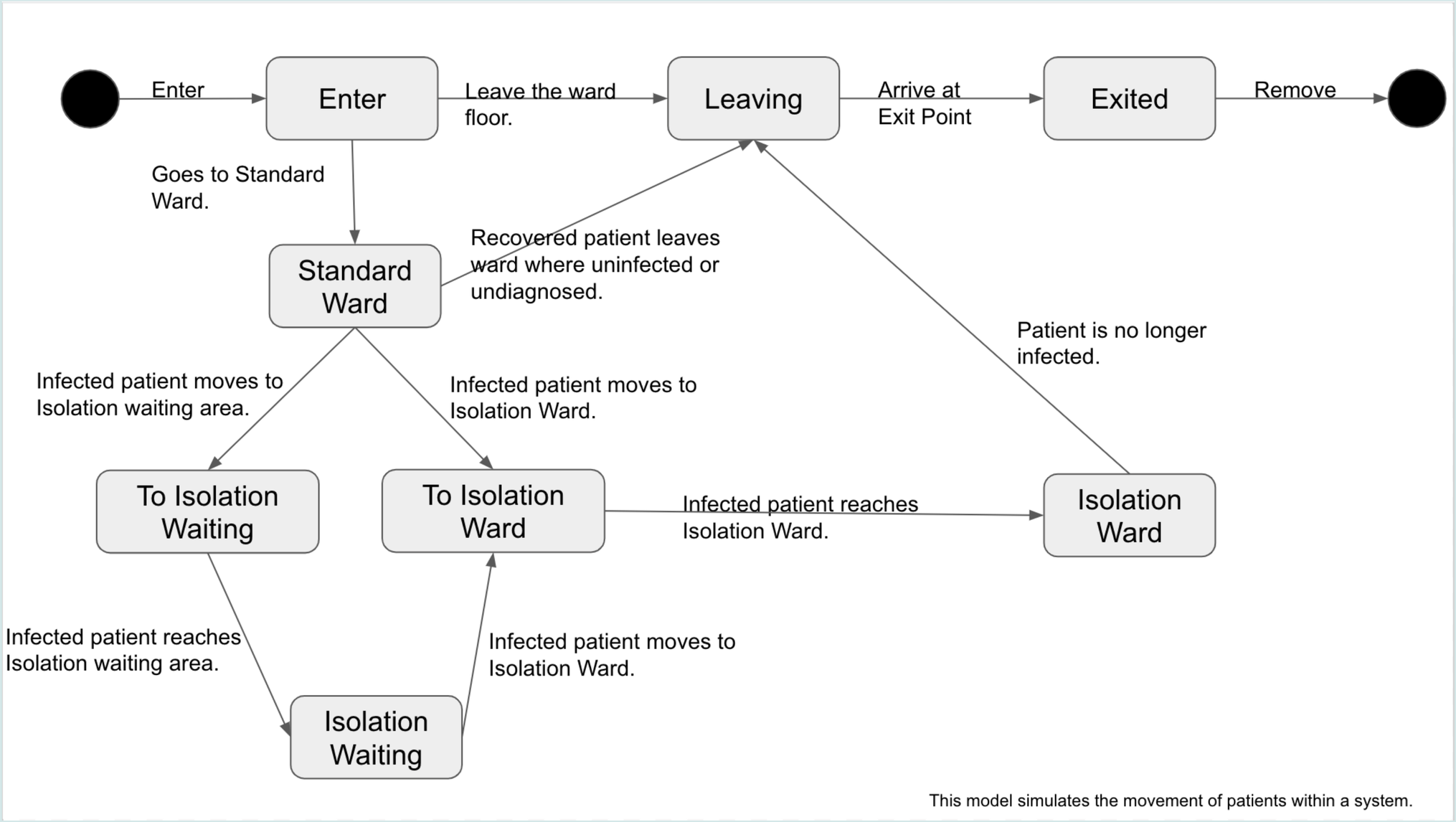
The next step's row and column are assigned to the agent's location row and column variables respectively. This sets their next position.

updateDynamicDoctors()

```
function updateDynamicDoctors() {
    // loop over all the agents and update their states
    for (var doctorIndex in doctors) {
        updateDoctors(doctorIndex)
    }
    updateSurface()
}
```

This function applies the updateDoctors() function to all doctors in the doctor list.

Patient State Diagram



Add Patient Agents

```
function addDynamicPatients() {
    // Patients are dynamic agents: they enter the clinic, wait, get treated, and then leave
    // All patients are green for this.
    if (Math.random() < probArrivalP) { // if patient arrives
        nextPatientID++
        var newpatient = {
            "id" : nextPatientID,
            "type": 'Uninfected',
            "location": {
                "row": entryPoint.startRow,    // appear from entry point
                "col": entryPoint.startCol     // appear from leftmost
            },
            "target": {
                "row": 10,
                "col": 3
            },
            "state": ENTER,
            "ward": 'None',
        }
            if (Math.random() < probUndiagnosed) {newpatient.type = 'Undiagnosed'}
            else {newpatient.type = 'Uninfected'}
        patients.push(newpatient)
    }
}
```

Increment Patient ID so each patient has a unique number

Generate new patient

Assign patient type using undiagnosed probability rate. Presence of patients with undiagnosed infections is commonplace and is required to model the spread of the infection.

PATIENT

id
1, 2, ...

type
"Uninfected": Injured and uninfected,
"Undiagnosed": Infected and undiagnosed,
"Infected": Infected and diagnosed,
"Discharged': Recovered from infection or injury,
"Dead": Dead after getting infected

location; target
"row"
"col"

state
ENTER

ward
"None"

Patient Movement via updatePatient()

Before patient states are considered

```
function updatePatient(patientIndex) {
    //patientIndex is an index into the patients data array
    patientIndex = Number(patientIndex) //it seems patientIndex was coming in as a string
    var patient = patients[patientIndex]
    // get the current location of the patient
    var row = patient.location.row
    var col = patient.location.col
    var type = patient.type    //Uninfected or Undiagnosed or Infected or Discharged or Dead
    var state = patient.state //ENTER or STANDARDWARD or TOISOLATIONWAITING or ISOLATIONWAITING or TOISOLATIONWARD or ISOLATIONWARD or LEAVING or EXITED

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

Store patient variables of location, type and state

Create boolean to tell if patient is at target position

Patient State: ENTER

```
switch (state) {
    case ENTER:
        if (hasArrived) { // check if patient reached random checkpoint
            var standardWards = [standardWard, standardWard2, standardWard3]
            var emptyWards = []
            for (i in standardWards) {
                if (standardWards[i].numPatients < standardWards[i].capacity) {
                    emptyWards.push(standardWards[i])
                }
            }
            if (emptyWards.length == 0) { // no space
                patient.state = LEAVING
                statistics[1].count++ // count total no. of patients turned away due to full standard ward
                patient.target.row = exitRow // exit is a few rows above entry
                patient.target.col = exitCol // first col
            }
            // if space in standard ward
            else {
                patient.state = STANDARDWARD
                // randomize ward choice out of three after checking for capacity across wards and edit the randomization below, as well as numPatients
                var randomWard = Math.floor(Math.random() * emptyWards.length)
                var selectedWard = emptyWards[randomWard]
                if (selectedWard.label == 'Standard Ward 1') {
                    patient.ward = standardWard
                } else if (selectedWard.label == 'Standard Ward 2') {
                    patient.ward = standardWard2
                } else {
                    patient.ward = standardWard3
                }
                // randomize space in ward
                var selectedwardRow = getRandomInt(selectedWard.startRow, (selectedWard.startRow + selectedWard.numRows - 1));
                var selectedwardCol = getRandomInt(selectedWard.startCol, (selectedWard.startCol + selectedWard.numCols - 1));
                patient.target.row = selectedwardRow
                patient.target.col = selectedwardCol
                selectedWard.numPatients++ // checked that increment here
            }
        }
        break
```

If patient reaches checkpoint, check ward availability. Else, no changes to patient.

Increment patient turned away from all standard wards

If all three standard wards are at full capacity, patient state changes and moves to the exit.

If space present in at least one standard ward, patient state changes and is randomly assigned a space in a non-full ward and moves to go there.

Increment count of patient in allocated ward

End of Enter state

Patient State: STANDARDWARD

1

```
case STANDARDWARD:
    if (hasArrived){
        switch (type) {
            case 'Uninfected':
                //identify the doctors/patients infected
                var infecteddoctors=doctors.filter(function(d){return d.type=="I";});
                var infectedpatients=patients.filter(function(d){return d.type=="Infected";});
                var undiagnosedpatients=patients.filter(function(d){return d.type=="Undiagnosed";});
                // combine them into a single array of all infected people for future use
                var undiagnosedpeople = infecteddoctors.concat(undiagnosedpatients)
                var infectiouspeople = infectedpatients.concat(undiagnosedpeople)

                //determine if any infected doctor/patient is nearby
                // infection by proximity
                i=0
                if (infectiouspeople.length > 0 && patient.type == "Uninfected") { // for uninfected patients where infected and undiagnosedpatients exist
                    while (patient.type == "Uninfected" && i < infectiouspeople.length){
                    var infected = infectiouspeople[i];
                    var infectedrow = infected.location.row
                    var infectedcol = infected.location.col
                    var distance = Math.sqrt((infectedrow-row)*(infectedrow-row)+(infectedcol-col)*(infectedcol-col))

                    // determine if the patient gets infected
                    if (distance<DistTransmission){
                        if (Math.random() < InfectionRate) { // if infected
                            patient.type = 'Undiagnosed'
                            statistics[0].count++ // increment infected since uninfected becomes infected
```

For Uninfected patients who are positioned in their allocated standard ward position

Find all infectious people

Check contact and infection by proximity if infectious (diagnosed and undiagnosed) people are present

Check distance

If patient comes into close contact with an infected individual, check if infected. If infected, change their state.

Increment count of infected patient

2

```
    if (Math.random() < DiagnosisRate) { // patient is diagnosed
        patient.type = 'Infected'
        if (isolationWard.numPatients >= isolationWard.capacity) { // iso is completely full
            statistics[2].count++ // turned away due to iso ward fullness
            patient.state = TOISOLATIONWAITING
            var waitingAreaRow = getRandomInt(waitingArea.startRow, (waitingArea.startRow + waitingArea.numRows - 1));
            var waitingAreaCol = getRandomInt(waitingArea.startCol, (waitingArea.startCol + waitingArea.numCols - 1));
            patient.target.row = waitingAreaRow
            patient.target.col = waitingAreaCol

            newinfectedpatient = {
                "type": "patient",
                "id": patient.id, // helps you find that agent in their original list (patients or doctors), possibly for updating their state if needed
                "waitingid": nextWaitingID // tracks who came first
            }
            nextWaitingID++ // increment waiting id
            infectedwaitinglist.push(newinfectedpatient) // add to infected waiting area list
            var patientward = patient.ward
            patientward.numPatients--; // leave standard ward, assuming diagnosed

        } else { // iso has space
            patient.state = TOISOLATIONWARD
            // randomize empty iso bed
            var isolationwardRow = getRandomInt(isolationWard.startRow, (isolationWard.startRow + isolationWard.numRows - 1));
            var isolationwardCol = getRandomInt(isolationWard.startCol, (isolationWard.startCol + isolationWard.numCols - 1));
            patient.target.row = isolationwardRow
            patient.target.col = isolationwardCol
            isolationWard.numPatients++
            var patientward = patient.ward
            patientward.numPatients--; // leave standard ward, assuming diagnosed

        }
    }
    else { // not infected
        if (Math.random() < probRecoveryNo) { // recovered from injury
            patient.state = LEAVING
            patient.type = 'Discharged'
            patient.target.row = exitRow
            patient.target.col = exitCol
            var patientward = patient.ward
            patientward.numPatients--; // leave standard ward
        }
    }
}
i=i+1
}
```

If patient is infected and diagnosed after contact, check isolation ward.

Increment patient turned away from isolation ward

If isolation ward has no space, send patient to waiting area and change state. Store patient info to be accessed at waiting room, with a queue number assigned.

Increment queue number in waiting room

Decrement patient count in patient's standard ward

If isolation ward has space, send patient there and change state.

Increment patient count in the isolation ward

Decrement patient count in patient's standard ward

If patient is not infected, check if they have recovered from their original injury and discharge them if required.

Decrement patient count in patient's standard ward

Iterate through all infected individuals

3

```
} else { // no infected and undiagnosed patients
    if (Math.random() < probRecoveryNo) { // recovered from injury
        patient.state = LEAVING
        patient.type = 'Discharged'
        patient.target.row = exitRow
        patient.target.col = exitCol
        var patientward = patient.ward
        patientward.numPatients--; // leave standard ward
    }
}
break;
```

If no infected individuals are present in the system, check if they have recovered from their original injury and discharge them if required.

Decrement patient count in patient's standard ward

End for case of Uninfected patients in StandardWard

4

For infected but undiagnosed patients in the standard ward

```
case 'Undiagnosed': // if infected but still in standard ward (aka undiagnosed and infected)
    if (Math.random() < DiagnosisRate) {
        patient.type = 'Infected'
        if (isolationWard.numPatients >= isolationWard.capacity) { // iso is completely full
            patient.state = TOISOLATIONWAITING
            var waitingAreaRow = getRandomInt(waitingArea.startRow, (waitingArea.startRow + waitingArea.numRows - 1));
            var waitingAreaCol = getRandomInt(waitingArea.startCol, (waitingArea.startCol + waitingArea.numCols - 1));
            patient.target.row = waitingAreaRow
            patient.target.col = waitingAreaCol


            newinfectedpatient = {
                "type": "patient",
                "id": patient.id, // helps you find that agent in their original list (patients or doctors), possibly for updating their state if needed
                "waitingid": nextWaitingID  // tracks who came first
            }
            nextWaitingID++ // increment waiting id
            infectedwaitinglist.push(newinfectedpatient)  // add to infected waiting area list
            var patientward = patient.ward
            patientward.numPatients--; // leave standard ward
        } else { // iso has space
            patient.state = TOISOLATIONWARD
            var isolationwardRow = getRandomInt(isolationWard.startRow, (isolationWard.startRow + isolationWard.numRows - 1));
            var isolationwardCol = getRandomInt(isolationWard.startCol, (isolationWard.startCol + isolationWard.numCols - 1));
            patient.target.row = isolationwardRow
            patient.target.col = isolationwardCol
            isolationWard.numPatients++
            var patientward = patient.ward
            patientward.numPatients--; // leave standard ward
        }
    } break;
```

If diagnosed and isolation ward has no space, change state and move target to random position in waiting area.

If isolation ward has no space, send patient to waiting area and change state. Store patient info to be accessed at waiting room, with a queue number assigned.

Increment queue number in waiting room

If infected patient is diagnosed and isolation ward has space, change patient state and move patient target to a randomised space in the isolation ward.

Increment patient count in isolation ward

Decrement count in patient's standard ward

End for case of Undiagnosed patients in StandardWard

```
} else {//if have not arrived aka moving to standard ward, nothing→
    var infecteddoctors=doctors.filter(function(d){return d.type=="I";});
    var infectedpatients=patients.filter(function(d){return d.type=="Infected";});
    var undiagnosedpatients=patients.filter(function(d){return d.type=="Undiagnosed";});
    // combine them into a single array of all infected people for future use
    var undiagnosedpeople = infecteddoctors.concat(undiagnosedpatients)
    var infectiouspeople = infectedpatients.concat(undiagnosedpeople)

    //determine if any infected doctor/patient is nearby
    // infection by proximity
    i=0
    if (infectiouspeople.length > 0 && patient.type == "Uninfected") { // for uninfected patients
        while (patient.type == "Uninfected" && i < infectiouspeople.length){
        var infected = infectiouspeople[i];
        var infectedrow = infected.location.row
        var infectedcol = infected.location.col
        var distance = Math.sqrt((infectedrow-row)*(infectedrow-row)+(infectedcol-col)*(infectedcol-col))

        // determine if the patient gets infected
        if (distance<DistTransmission){
            if (Math.random() < InfectionRate) { // if infected after being uninfected
                patient.type = 'Undiagnosed'
                statistics[0].count++ // increment infected
            }
        }
        i=i+1
        }→
    }
}
break;
```

If patient has not arrived at standard ward yet after entering

Iterate through all infected individuals

If infectious (diagnosed and undiagnosed) people are present, check their distance to the uninfected patient

If patient comes into close contact with an infected individual, check if the patient gets infected.

Increment count of infected patient

End of StandardWard state

Patient State: TOISOLATIONWAITING

```
case TOISOLATIONWAITING: // reached isolation waiting
    if (hasArrived) {
        patient.state = ISOLATIONWAITING
        statistics[4].count++ // increment the number of people at waiting area
    }
    break;
```

If infected patient reaches the isolation waiting area, change their state and increment waiting area count.

Else, patient does not undergo any additional changes except for moving to isolation ward.

End of ToIsolationWaiting state

Patient State: ISOLATIONWAITING

```
case ISOLATIONWAITING: // check if isolation ward has space
    if (isolationWard.numPatients < isolationWard.capacity) { // has space
        var nextIsoPerson = findNextIsoPerson() // output is type, id
        if (nextIsoPerson[0] == 'patient' && patient.id == nextIsoPerson[1]) { // if this patient is to go to iso ward
            patient.state = TOISOLATIONWARD // change state
            // randomize empty iso bed
            var isolationwardRow = getRandomInt(isolationWard.startRow, (isolationWard.startRow + isolationWard.numRows - 1));
            var isolationwardCol = getRandomInt(isolationWard.startCol, (isolationWard.startCol + isolationWard.numCols - 1));
            patient.target.row = isolationwardRow
            patient.target.col = isolationwardCol
            isolationWard.numPatients++
            // leaving from iso waiting area
            statistics[4].count--; // decrement since leaving waiting area
            infectedwaitinglist = infectedwaitinglist.filter(function (d) { // remove from isowaitinglist
                return d.waitingid != nextIsolationID
            })
            nextIsolationID++ // increment nextIsolation
        }
        // else nothing happens
    }
    break;
```

Increment patient count in isolation ward

Decrement count of patients in waiting area

Increment next queue number to go to isolation ward

If patient has reached isolation waiting area, check if there is space in the isolation ward.

If there is space in isolation ward, check if this patient is next in the queue to enter the ward.

If next in queue, send patient to isolation ward and change statistics. Else, patient stays in the waiting area.

End of IsolationWaiting state

Patient State: TOISOLATIONWARD

```
case TOISOLATIONWARD:
    if (hasArrived) { // reached isolation ward
        patient.state = ISOLATIONWARD
    }
    break;
```

If patient reaches the isolation ward, change their state. Otherwise, patient continues to move to the isolation ward.

End of TolsolationWard state

Patient State: ISOLATIONWARD

```
case ISOLATIONWARD:
    // Complete treatment randomly according to the probability of recovery
    if (Math.random() < probRecoveryYes) { // chance of recovery from infection
        patient.state = LEAVING
        patient.type = 'Discharged'
        patient.target.row = exitRow
        patient.target.col = exitCol
        isolationWard.numPatients--; // leave isolation ward
    } else { // patient not recovered
        if (Math.random() < probDeath) { // if patient dies
            patient.state = LEAVING
            patient.type = 'Dead'
            statistics[3].count++ //
            patient.target.row = deadRow
            patient.target.col = deadCol
            isolationWard.numPatients--;
        }
    }
    break;
```

Decrement patient count in isolation ward

Increment patient death count

Decrement patient count in isolation ward

Once patient is at the isolation ward, they recover from the infection according to the probability of infection recovery.

If patient recovers, their state changes and they are sent to the discharge position.

If patient does not recover, there is a possibility that they die or are still infected. If patient dies, their state changes and they are sent to the dead position. Else if still recovering, patient stays in isolation ward position.

End of IsolationWard state

Patient State: LEAVING

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

If patient arrives at the dead exit point or discharge exit point, their state changes.

If they are still moving to these positions, no changes occur.

End of Leaving state

Step : Patient Movement

```
// set the destination row and column
var targetRow = patient.target.row
var targetCol = patient.target.col
// compute the distance to the target destination
var rowsToGo = targetRow - row // row is current location
var colsToGo = targetCol - col // col is current location
// set the speed
var cellsPerStep = 1
    // Math.min gives 0 when at target position so dont need to move
    //   // else gives 1 since one step to move
    // Math.sign gives +1 or -1
var newRow = row + Math.min(Math.abs(rowsToGo), cellsPerStep) * Math.sign(rowsToGo)
var newCol = col + Math.min(Math.abs(colsToGo), cellsPerStep) * Math.sign(colsToGo)
// check if the new position is
patient.location.row = newRow
patient.location.col = newCol
```

Set patient movement to be one step at a time

Generate and assign patient target row and col for movement.

End of UpdatePatient()

Step : Remove and Update Patients

```
function removeDynamicPatients() {
    //We need to remove patients who have been discharged.
    //Select all svg elements of class "patient" and map it to the data list called patients
    var allpatients = surface.selectAll('.patient').data(patients)
    //Select all the svg groups of class "patient" whose state is EXITED
    var treatedpatients = allpatients.filter(function (d, i) {
        return d.state == EXITED
    })
    //Remove the svg groups of EXITED patients: they will disappear from the screen at this point
    treatedpatients.remove()

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

function updateDynamicPatients() {
    //loop over all the agents and update their states
    for (var patientIndex in patients) {
        updatePatient(patientIndex)
    }
    updateSurface()
}
```

Remove patients who have Exited from simulation space at the dead or discharge exit doors from the system.

Update all patients within simulation when called at each time step.

Update Simulation Display at each time step

Page Initialization Code

```
// This next function is executed when the script is loaded. It contains the page initialization code.
;
    // do not touch these functions
(function () {
    // Your page initialization code goes here
    // All elements of the DOM will be available here
    window.addEventListener('resize', redrawWindow) //Redraw whenever the window is resized
    simTimer = window.setInterval(simStep, animationDelay) // call the function simStep every animationDelay milliseconds
    redrawWindow()
})()

// We need a function to start and pause the the simulation.
function toggleSimStep() {
    //this function is called by a click event on the html page.
    // Search BasicAgentModel.html to find where it is called.
    isRunning = !isRunning // since a true/false variable where only binary options, change it when clicked via !=
    console.log('isRunning: ' + isRunning) // to track over time
}
```

add a delay for the animation

set a timer to call the function repeatedly

Referred from Prof's code

redrawWindow()

```
function redrawWindow() {
    isRunning = false // used by simStep
    window.clearInterval(simTimer) // clear the Timer
    animationDelay = 550 - document.getElementById('slider1').value // access slider value where max is 550 aka fastest
    probArrivalP = document.getElementById("slider2").value;    // Arrival Rate Parameters are no longer defined in the code but through th
    InfectionRate = document.getElementById("slider3").value;   // Infection Rate Parameters are no longer defined in the code but through
    DistTransmission = document.getElementById("slider4").value;// Distance of Transmission Parameters are no longer defined in the code bu
    DiagnosisRate = document.getElementById("slider5").value;   // Infecton Diagnosis Rate Parameters are no longer defined in the code but
    probRecoveryYes = document.getElementById("slider6").value; // Infection Recovery Rate Parameters are no longer defined in the code but
    numBedsPerWard = document.getElementById("slider7").value;  // No. of beds per ward Parameters are no longer defined in the code but th
    numDoctors = document.getElementById("slider8").value;      // No. of Healthcare Workers Parameters are no longer defined in the code b
    simTimer = window.setInterval(simStep, animationDelay)      // call the function simStep every animationDelay milliseconds

    // Re-initialize simulation variables which are the only variables to change

    currentTime = 0;
    nextPatientID = 0;
    nextWaitingID = 1;
    nextIsolationID = 1;
    statistics[0].count=0; // affected indiv total
    statistics[1].count=0; // patients turned away due to full standard ward
    statistics[2].count=0; // infected people turned away due to full isolation ward
    statistics[3].count=0; // death count
    statistics[4].count=0; // divya added: patient at waiting area

    patients = [];
    doctors = [];
    visitors = [];
    infectedwaitinglist = [];

    // redefine area variables
    areas[0].numPatients = 0
    areas[1].numPatients = 0
    areas[2].numPatients = 0
    areas[4].numPatients = 0
    areas[5].numPatients = 0
    areas[0].capacity = numBedsPerWard
    areas[2].capacity = numBedsPerWard
    areas[4].capacity = numBedsPerWard
    areas[5].capacity = numBedsPerWard
```

add a slider for each single parameter and link parameter value to value chosen by user to be used by the code

re-initialize agent id every time simulation restarts

re-initialize statistics every time simulation restarts

re-initialize empty dynamic lists of agents every time simulation restarts

re-initialize area statistics (number of patients in areas) every time simulation restarts

redrawWindow()

```
//resize the drawing surface; remove all its contents;
var drawsurface = document.getElementById('surface')
var creditselement = document.getElementById('credits')
var w = window.innerWidth
var h = window.innerHeight
var surfaceWidth = w - 3 * WINDOWBORDERSIZE // account for border width
var surfaceHeight = h - creditselement.offsetHeight - 3 * WINDOWBORDERSIZE // account for border height

// assign values for css to access
drawsurface.style.width = surfaceWidth + 'px'
drawsurface.style.height = surfaceHeight + 'px'
drawsurface.style.left = WINDOWBORDERSIZE / 2 + 'px'
drawsurface.style.top = WINDOWBORDERSIZE / 2 + 'px'
//drawsurface.style.border = 'thick solid #0000FF' //The border is mainly for debugging; okay to remove it
drawsurface.innerHTML = '' //This empties the contents of the drawing surface, like jQuery erase().

// Compute the cellWidth and cellHeight, given the size of the drawing surface
numCols = maxCols
cellWidth = surfaceWidth / numCols
numRows = Math.ceil(surfaceHeight / cellWidth)
cellHeight = surfaceHeight / numRows

// In other functions we will access the drawing surface using the d3 library.
//Here we set the global variable, surface, equal to the d3 selection of the drawing surface
surface = d3.select('#surface')
surface.selectAll('*').remove() // we added this because setting the inner html to blank may not remove all svg elements
surface.style('font-size', '100%')
```

Referred from Prof's code

```
    // suzanne added
    for (let i = 0; i < numDoctors; i++) {    // loop through each doctor to add them to the dynamic list
        // random starting points
        var randomCol = getRandomInt(hospitalStartCol, hospitalEndCol);
        var randomRow = getRandomInt(hospitalStartRow, hospitalEndRow);


        // random target wards
        var ward = getRandomWardTarget();

        // create an entry for each healthcare worker as an entry in the doctors list variable
        var newdoctor = {
            "id":1+i,
            "type":"H",     // type: H for healthy, I for infected
            "location":{
                "row":randomRow,
                "col":randomCol
            },        // begin with all doctors being healthy
            "target":{
                "row":ward[0],
                "col":ward[1]
            },
            "state": GOTOWARD,
            "color":"green"
        };

        if (Math.random() < probUndiagnosed) {                // doctor is initially infected if the random number is less than the p
            newdoctor.type = "I";
            newdoctor.color = "orange";
            //statistics[0].count++;
            //totnewinfected++;
        } else {
            newdoctor.type = "H";
            newdoctor.color = "green"
        };

        // add each healthcare worker as an entry in the doctors list variable
        doctors.push(newdoctor);      // "push" your new patient into the list of patients (adds a new entry in the patient table)
    }
    // suzanne end


    // rebuild contents of the drawing surface
    updateSurface()
}
```

redraw Window()

Re-initialize all doctor entries in the doctor list

Since the doctor agents are all present at the beginning of the simulation

end of redrawWindow()

update Surface()

```
function updateSurface(){
    // This function is used to create or update most of the svg elements on the drawing surface.
    // See the function removeDynamicAgents() for how we remove svg elements

    // divya added
    var allpatients = surface.selectAll('.patient').data(patients)

    // If the list of svg elements is longer than the data list, the excess elements are in the .exit() list
    // Excess elements need to be removed:
    allpatients.exit().remove() //remove all svg elements associated with entries that are no longer in the data list
    // (This remove function is needed when we resize the window and re-initialize the patients array)

    // If the list of svg elements is shorter than the data list, the new elements are in the .enter() list.
    // The first time this is called, all the elements of data will be in the .enter() list.
    // Create an svg group ("g") for each new entry in the data list; give it class "patient"
    var newpatients = allpatients.enter().append('g').attr('class', 'patient')
    //Append an image element to each new patient svg group, position it according to the location data, and size it to fill a cell
    // Also note that we can choose a different image to represent the patient based on the patient type
    newpatients.append('svg:image')
        .attr('x', function (d) {
            var cell = getLocationCell(d.location)
            return cell.x + 'px'
        })
        .attr('y', function (d) {
            var cell = getLocationCell(d.location)
            return cell.y + 'px'
        })
        .attr('width', Math.min(cellWidth, cellHeight) + 'px')
        .attr('height', Math.min(cellWidth, cellHeight) + 'px')
        .attr('xlink:href', function (d) {
            if (d.type == 'Uninfected') return urlHealthyPatient
            else if (d.type == 'Undiagnosed') return urlUndiagnosedPatient
            else return urlInfectedPatient // implies diagnosis
        })

    // For the existing patients, we want to update their location on the screen
    // but we would like to do it with a smooth transition from their previous position.
    // D3 provides a very nice transition function allowing us to animate transformations of our svg elements.

    //First, we select the image elements in the allpatients list
    var images_patients = allpatients.selectAll('image')
    // Next we define a transition for each of these image elements.
    // Note that we only need to update the attributes of the image element which change
    images_patients
        .transition()
        .attr('x', function (d) {
            var cell = getLocationCell(d.location)
            return cell.x + 'px'
        })
        .attr('y', function (d) {
            var cell = getLocationCell(d.location)
            return cell.y + 'px'
        })
        .attr('xlink:href', function (d) {
            if (d.type == 'Uninfected') return urlHealthyPatient
            else if (d.type == 'Undiagnosed') return urlUndiagnosedPatient
            else if (d.type == 'Infected') return urlInfectedPatient
            else if (d.type == 'Discharged') return urlDischargedPatient
            else return urlDeadPatient
        })
        .duration(animationDelay)
        .ease('linear') // This specifies the speed and type of transition we want.
```

Adapted for patient agents from Prof's code

updateSurface()

```
// The simulation should serve some purpose
// so we will compute and display the average length of stay of each patient type.
// We created the array "statistics" for this purpose.
// Here we will create a group for each element of the statistics array (five elements)
var allstatistics = surface.selectAll(".statistics").data(statistics);
var newstatistics = allstatistics.enter().append("g").attr("class","statistics");
    // For each new statistic group created we append a text label
    newstatistics.append("text")
    .attr("x", function(d) {
        var cell= getLocationCell(d.location);
        return (cell.x+cellWidth)+"px";
    })
    .attr("y", function(d) {
        var cell= getLocationCell(d.location);
        return (cell.y+cellHeight/2)+"px"; })
    .attr("dy", ".35em")
    .text("");

// The data in the statistics array are always being updated.
// So, here we update the text in the labels with the updated information.
allstatistics.selectAll("text").text(function(d) {
    var nopatients = d.count; // cumulativeValue and count for each statistic are always changing
    return d.name+nopatients.toFixed(1); }); //The toFixed() function sets the number of decimal places to display
```

Adapted from Prof's code

updateSurface()

```
// Finally, we would like to draw boxes around the different areas of our system. We can use d3 to do that too.

//First a box representing the hospital and respective wards.
var allareas = surface.selectAll(".areas").data(areas);
var newareas = allareas.enter().append("g").attr("class", "areas");
// For each new area, append a rectangle to the group
newareas.append("rect")
    .attr("x", function(d){
        return (d.startCol-1)*cellWidth;})
    .attr("y",  function(d){
        return (d.startRow-1)*cellHeight;})
    .attr("width",  function(d){
        return d.numCols*cellWidth;})
    .attr("height",  function(d){
        return d.numRows*cellWidth;})
    .style("fill", function(d) {
        return d.color; })
    .style("stroke","black")
    .style("stroke-width",1);
```

Referred from Prof's code

```
    //Select all svg elements of class "doctors" and map it to the data list called doctors
    var alldoctors = surface.selectAll(".doctor").data(doctors);

    // If the list of svg elements is longer than the data list, the excess elements are in the .exit() list
    // Excess elements need to be removed:
    // suzanne commented out the line below (please check)
    //alldoctors.exit().remove(); //remove all svg elements associated with entries that are no longer in the data list
    // (This remove function is needed when we resize the window and re-initialize the citizens array)

    //suzanne added
    var newdoctors = alldoctors.enter().append('g').attr('class', 'doctor')
    newdoctors
        .append('svg:image')
        .attr('x', function (d) {
            var cell = getLocationCell(d.location)
            return cell.x + 'px'
        })
        .attr('y', function (d) {
            var cell = getLocationCell(d.location)
            return cell.y + 'px'
        })
        .attr('width', Math.min(cellWidth, cellHeight) + 'px')
        .attr('height', Math.min(cellWidth, cellHeight) + 'px')
        .attr('xlink:href', function (d) {
            if (d.color == 'blue') return urlImmuneDoctor
            else if (d.color == 'red') return urlInfectedDoctor
            else if (d.color == 'orange') return urlUndiagnosedDoctor
            else return urlHealthyDoctor   // d.color == 'green'
        })
    var images_doctors = alldoctors.selectAll('image')
    // Next we define a transition for each of these image elements.
    // Note that we only need to update the attributes of the image element which change
    images_doctors
        .transition()
        .attr('x', function (d) {
            var cell = getLocationCell(d.location)
            return cell.x + 'px'
        })
        .attr('y', function (d) {
            var cell = getLocationCell(d.location)
            return cell.y + 'px'
        })
        .attr('xlink:href', function (d) {
            if (d.color == 'blue') return urlImmuneDoctor
            else if (d.color == 'red') return urlInfectedDoctor
            else if (d.color == 'orange') return urlUndiagnosedDoctor
            else return urlHealthyDoctor   // d.color == 'green'
        })
        .duration(animationDelay)
        .ease('linear') // This specifies the speed and type of transition we want.

}
```

update Surface()

Adapted for doctor agents from Prof's code

simStep()

```
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++;
        addDynamicPatients();
        updateDynamicDoctors();
        updateDynamicPatients();
        removeDynamicPatients();
    }
}
```

At each simulation step while simulation is running, add, update and remove patients. Since doctors are not dynamically created and do not exit the system, they are only updated.

User Interface & Visualization Code

HTML Setup

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Hospital Infection Spread Model</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />


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

    <!-- inspectElement is useful for debugging javascript objects
    <script type="text/javascript" src="lib/inspectElement.js"></script>-->

    <!-- plotly is for drawing graphs -->
    <script type="text/javascript" src="lib/plotly.min.js"></script>

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

Referred from Prof's code

HTML Setup

```
</head>
<!-- Set class to official for more formal font; unofficial for informal font -->
<body class="official bgColor0">
    <p style="position:absolute;top:57%;left:79%; font-size:90%;">Simulation Parameters</p>
    <p style="position:absolute;top:57%;left:2%; font-size:90%;">Graphs</p>
    <p style="position:absolute;top:7.7%;left:1.5%; font-size:60%;">Entry</p>
    <p style="position:absolute;top:28%;left:1.7%; font-size:60%;">Exit</p>
    <p style="position:absolute;top:17.9%;left:94.6%; font-size:60%;">Dead</p>
    <p style="position:absolute;top:2.3%;left:11%; font-size:60%;">Standard Ward 1</p>
    <p style="position:absolute;top:28.2%;left:11%; font-size:60%;">Standard Ward 2</p>
    <p style="position:absolute;top:28.2%;left:82%; font-size:60%;">Standard Ward 3</p>
    <p style="position:absolute;top:2.3%;left:58%; font-size:60%;">Waiting Area</p>
    <p style="position:absolute;top:2.3%;left:80%; font-size:60%;">Isolation Ward</p>
    <p style="position:absolute;top:23%;left:46%; font-size:60%;">Communal Area</p>
```

positions and size of labels

```
<div class = "placeholder-box" style="position:absolute;top:63%;left:70%; width: 28%; height: 30%;">
    <p style="position:absolute;top:1%;left:5%;"> 1) Simulation Speed </p>
    <p><input id="slider1" type="range" min="0" value="300" max="5000" step="100"
        oninput="slider1Value.innerText = this.value" onchange="redrawWindow();"
        style="position:absolute;top:17%;left:5%"/></p>
    <p id="slider1Value" style="position:absolute;top:12%;left:37%">300</p>

    <p style="position:absolute;top:25%;left:5%;"> 2) Arrival Rate </p>
  <p><input id="slider2" type="range" min="0" value="0.5" max="1" step="0.05"
    oninput="slider2Value.innerText = this.value" onchange="redrawWindow();"
    style="position:absolute;top:41%;left:5%" /></p>
    <p id="slider2Value" style="position:absolute;top:36%;left:37%">0.5</p>

    <p style="position:absolute;top:49%;left:5%;"> 3) Infection Rate </p>
    <p><input id="slider3" type="range" min="0" value="0.5" max="1" step="0.05"
        oninput="slider3Value.innerText = this.value" onchange="redrawWindow();"
        style="position:absolute;top:65%;left:5%" /></p>
    <p id="slider3Value" style="position:absolute;top:60%;left:37%">0.5</p>

    <p style="position:absolute;top:73%;left:5%;"> 4) Distance of Transmission </p>
    <input id="slider4" type="range" min="0" value="2" max="5" step="0.5"
    oninput="slider4Value.innerText = this.value" onchange="redrawWindow();"
    style="position:absolute;top:89%;left:5%" /></p>
    <p id="slider4Value" style="position:absolute;top:84%;left:37%">2</p>

    <p style="position:absolute;top:1%;left:50%;"> 5) Infection Diagnosis Rate </p>
    <p><input id="slider5" type="range" min="0" value="0.7" max="1" step="0.05"
        oninput="slider5Value.innerText = this.value" onchange="redrawWindow();"
        style="position:absolute;top:17%;left:50%" /></p>
    <p id="slider5Value" style="position:absolute;top:12%;left:82%">0.7</p>

    <p style="position:absolute;top:25%;left:50%;"> 6) Infection Recovery Rate </p>
    <p><input id="slider6" type="range" min="0" value="0.5" max="1" step="0.05"
        oninput="slider6Value.innerText = this.value" onchange="redrawWindow();"
        style="position:absolute;top:41%;left:50%" /></p>
    <p id="slider6Value" style="position:absolute;top:36%;left:82%">0.5</p>

    <p style="position:absolute;top:49%;left:50%;"> 7) No. of Beds per Ward </p>
    <p><input id="slider7" type="range" min="2" value="16" max="30" step="2"
        oninput="slider7Value.innerText = this.value" onchange="redrawWindow();"
        style="position:absolute;top:65%;left:50%" /></p>
    <p id="slider7Value" style="position:absolute;top:60%;left:82%">16</p>

    <p style="position:absolute;top:73%;left:50%;"> 8) No. of Healthcare Workers </p>
    <p><input id="slider8" type="range" min="10" value="30" max="100" step="10"
        oninput="slider8Value.innerText = this.value" onchange="redrawWindow();"
        style="position:absolute;top:89%;left:50%" /></p>
  <p id="slider8Value" style="position:absolute;top:84%;left:82%">30</p>
</div>
```

HTML Setup

slider bars placed in a placeholder box to control:

- simulation speed
- arrival rate of patients
- infection rate
- distance of transmission
- infection diagnosis rate
- infection recovery rate
- number of beds per ward
- number of healthcare workers

specified minimum, maximum and default values of each slider and step of each change

display value of slider next to the slider in the html

HTML Setup

```
    <!-- The surface is the main playing field for the game -->
    <svg  id="surface" style="width:100%; height:55%;" xmlns="http://www.w3.org/2000/svg" version="1.1" onclick="toggleSimStep();">
    </svg>


    <div id="title"  style="position:absolute;bottom:0;left:0;">Hospital Infection Spread Model</div>
    <a id="credits" href="https://www.flaticon.com/" style="position:absolute;bottom:0;right:0;">Icons by https://www.flaticon.com/</a>

    <!-- Anything below this line should be a popup window or dialog or a late-loading library -->

    <script type="text/javascript" src="lib/Hospital_11.js"></script>

    <img src="images/legend.png" alt="Legend" style="position:absolute; top:63%; left:48%; width:21%; height:32%;">

    <!-- Add in Charts -->
    <div id="chart1" style="position:absolute; top:49.5%; left:0.5%; width:28%; height:50%;"></div>
    <div id="chart2" style="position:absolute; top:49.5%; left:24%; width:28%; height:50%;"></div>
    <script type="text/javascript" src="lib/PlotGraph.js"></script>
</body>
</html>
```

edited hyperlink to the source for images

edited position of charts on the screen

Referred from Prof's code

Css Code

```
.colorMain { color: #d4faff; }
.bgColor0 { background-color: #dff8fc; }
.bgColor1 { background-color: #B4FFE5; }
.bgColor2 { background-color: #78D9B7; }
.bgColor3 { background-color: #DAFFF2; }
.bgColor4 { background-color: #E5B4FF; }
.bgColor5 { background-color: #B778D9; }
.bgColor6 { background-color: #66CC00; }
.fontColor0 { color: #FFE5B4; }
.fontColor1 { color: #B4FFE5; }
.fontColor2 { color: #78D9B7; }
.fontColor3 { color: #DAFFF2; }
.fontColor4 { color: #E5B4FF; }
.fontColor5 { color: #B778D9; }
.fontColor6 { color: #F2DAFF; }
```

PlotGraph.js

```
// set functions to retrieve
function getData1() {           // proportion of infected people in the system

    var infecteddoctors=doctors.filter(function(d){return d.type == "I";});
    var infectedpatients=patients.filter(function(d){return d.type == "Infected" || "Undiagnosed";});
    // combine them into a single array of all infected people for future use
    var infectedpeople = infecteddoctors.concat(infectedpatients)
    var numInfected = infectedpeople.length

    var numAgents = patients.length + doctors.length

    return numInfected/numAgents;
    }
function getData2() {     // Cumulative no. of patients turned away / Total no. of patients who entered the
    var turnawaypercent = statistics[1].count/nextPatientID // patients turned away due to full standard wa
    return turnawaypercent;
    }
```

$$\text{\% of infected people in the system} = \frac{\text{No. of infected people}}{\text{Total no. of agents}}$$

END OF MODEL DOCUMENTATION