**public** **class** Monarch {

**private** String name;

**private** String name2;

//Variables that can be changed

//distance monarch moves in each step

**private** **int** steplength = ContextBuild.*steplength*;

//directionality parameter constrains possible changes in direction

**private** **double** mindir = ContextBuild.*mindir*;

**private** **double** maxdir = ContextBuild.*maxdir*;

//St Co is 0.09

**private** **double** probMoveRange = ContextBuild.*probMoveRange*;

//perception distance in meters

**private** **double** perception = ContextBuild.*perception*;

//number of eggs laid per step

**private** **double** eggsperlay = ContextBuild.*eggsperlay*;

//number of steps to remember in which polygon the monarch was

**private** **int** remembered = ContextBuild.*remembered*;

//turn the code to save monarch agent coordinates on and off - true means they are saved

//private boolean savecoords = false;

**private** **boolean** savecoords = ContextBuild.*savecoords*;

//other variables used in code

//current angle in radians that Monarch is moving in corr rand walk - initialize to random value

**private** **double** currAngle = 2\*Math.***PI***\*Math.*random*();

//angle for corr rand walk if chooses current polygon - initialize to random value

**private** **double** currAngle2 = 2\*Math.***PI***\*Math.*random*();

//magnitude of change in direction

**private** **double** angleChange;

//probEggs of currently occupied polygon

**private** **double** currProbEggs;

//probMove of currently occupied polygon

**private** **double** currProbMove;

//temp variable needed when angle crosses 360/0 degrees

**private** **double** tempAngle1;

**private** **double** tempAngle2;

//angle moved by monarch agent in previous step to use with bounce algorithm

**private** **double** lastAngle;

//number of steps taken each day

**private** **double** cumSteps = 0;

//the number of times a monarch can lay eggs in a day

**private** **int** numtimeslayeggs;

**private** **double** numtimeslayeggs2;

//number of eggs that can be laid on a given day

**private** **double** dailyeggstolay = 0;

//number of eggs laid per day

**private** **double** dailyeggslaid = 0;

//cumulative distance and maximum dist can move

**private** **double** cumDist = 0;

**private** **double** maxdist = 0;

//number of steps taken to lay all eggs

**private** **double** eggslaidsteps = 0;

//current coordinates of Monarch agent used for torus

**private** **double** x;

**private** **double** y;

//coordinates for output - doesn't work

**public** **double** xcoord;

**public** **double** ycoord;

//CLASS\_NAME of current polygon

//if (savecoords == true) {

**public** String CLASS\_NAME;

ArrayList monX = **new** ArrayList(); //xcoords

ArrayList monY = **new** ArrayList(); //ycoords

ArrayList className = **new** ArrayList(); //class names

String xcoords;

String ycoords;

String classnames;

//}

//Story County - slightly outside - used for most purposes

**static** **double** *xmin* = -93.7005;

**static** **double** *xmax* = -93.2300;

**static** **double** *ymin* = 41.8605;

**static** **double** *ymax* = 42.2107;

//get context and geography

**public** **static** GeometryFactory *fac* = **new** GeometryFactory();

CoordinateReferenceSystem equalAreaCRS;

MathTransform transform;

Geometry transformedIntersection;

//constructor

**public** Monarch(String name) {

**this**.name = name;

}

//Each step/tick is a day in the life of an egg-laying monarch

@ScheduledMethod(start = 1, interval = 1, priority = ScheduleParameters.***FIRST\_PRIORITY***)

**public** **void** step(){

name2 = name;

//initialize these variables to 0 each day/tick

cumDist = 0;

cumSteps = 0;

dailyeggslaid = 0;

eggslaidsteps = 0;

currAngle = 2\*Math.***PI***\*Math.*random*();

**double** bouncecounter = 0; //counter for number of bounces in a row

//Array to hold polygons previously visited

**boolean** seq = **false**; //variable to denote whether bounces are sequential

**double**[] memories = **new** **double**[remembered];

//gets current tick

**double** tick = RepastEssentials.*GetTickCount*();

//calculates max distance that can moved during current tick

maxdist = -500\*tick + 10500;

//potential eggs to lay each day - currently starts at 50 and drops to 30

dailyeggstolay = -2\*tick + 52;

//number of times a monarch can lay eggs each day if it lays x eggs each time

**double** numtimeslayeggs1 = dailyeggstolay/eggsperlay;

//number of times a monarch can lay eggs rounded up to nearest integer

numtimeslayeggs2 = Math.*ceil*(numtimeslayeggs1);

numtimeslayeggs = (**int**) numtimeslayeggs2;

//boolean to determine when while loop ends

**boolean** doneMove = **false**;

**while** (! doneMove) {

Context context = ContextUtils.*getContext*(**this**);

Geography<Object> geography = (Geography)context.getProjection("Monarchs");

//System.out.println("++++++++++++++++++++ Start New Step: +++++++++++++++++++++ Last Step:" + cumSteps);

//first coord for network display - Dr. Parry code

Geometry geom = geography.getGeometry(**this**);

Coordinate c1= geom.getCoordinates()[0];

Coordinate c2 = **null**;

//coords to output for utilization distribution analysis

**if** (savecoords == **true**){

xcoord = c1.x;

ycoord = c1.y;

monX.add(c1.x);

monY.add(c1.y);

}

//count of step number when eggs run out

**if** (numtimeslayeggs > 0) {

eggslaidsteps++;

}

ArrayList probs = **new** ArrayList(); //normalized pref/p values

ArrayList dists = **new** ArrayList(); //distances in lat/long units

ArrayList destX = **new** ArrayList(); //array for latitude coord of destinations

ArrayList destY = **new** ArrayList();

**double** runningsum = 0; //sum of fprobs to normalize p's

**double** fprob = 0; //final probMove for polygon after adjustments

//retrieve objects within perception distance

GeographyWithin within = **new** GeographyWithin(geography, perception, **this**);

//loops through objects within perception distance

**for** (Object obj : within.query()) {

//if an object is a ZoneAgent, then cast it as a ZoneAgent

**if** (obj **instanceof** ZoneAgent){

ZoneAgent zoneagent = (ZoneAgent)obj;

//System.out.println("1-ZoneAgentID = " + zoneagent.getID());

CoordinateReferenceSystem crs = geography.getCRS();

**double** probEggs = zoneagent.getprobEggs();

//System.out.println("2-probEggs = " + probEggs);

**double** probMove = zoneagent.getprobMove();

//System.out.println("3-probMove = " + probMove);

Polygon zonegeom = (Polygon)geography.getGeometry(zoneagent);

//gets the lat/long of closest point in polygon, stores in array

DistanceOp Op = **new** DistanceOp(geom, zonegeom);

//Lat/long coordinates of the nearest point of the zoneagent

Coordinate t1 = Op.nearestPoints()[1];

destX.add(t1.x);

destY.add(t1.y);

//distance between the monarch, poly in lat/long units

**double** distl = Op.distance();

dists.add(distl);

//for polygon that the monarch agent is currently within

**if**(distl == 0){

//lay eggs and add to cumulative total of current zoneagent

**double** currcumEggs = zoneagent.cumulativeeggs;

zoneagent.cumulativeeggs = layeggs(probEggs,currcumEggs);

**if** (savecoords == **true**){

CLASS\_NAME = zoneagent.name;

className.add(CLASS\_NAME);

}

currProbEggs = probEggs;

currProbMove = probMove;

**if**(remembered > 0){

//remember this polygon

**for** (**int** i = remembered-1; i > 0; i--){

memories[i] = memories[i-1];

}

memories[0] = zoneagent.ID;

}

}

//CALCULATE SCALED PROBABILITIES FOR POLYGON FOR MOVEMENT CHOICE

//Define an equal area CRS

**try** {

equalAreaCRS = CRS.*decode*("EPSG:2163", **true**);

} **catch** (NoSuchAuthorityCodeException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (FactoryException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

//create a buffer around the point of perception distance

Geometry pointBuffer = GeometryUtils.*generateBuffer*(geography, geom, perception);

//calculate area of intersection of buffer with zoneagent

Geometry intersection = pointBuffer.intersection(zonegeom);

//transform from lat/long units to meters

**try** {

transform = CRS.*findMathTransform*(crs, equalAreaCRS, **true**);

} **catch** (FactoryException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

//Geometry transformedIntersection;

**try** {

transformedIntersection = JTS.*transform*(intersection, transform);

} **catch** (MismatchedDimensionException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (TransformException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**double** meterarea = transformedIntersection.getArea();

//probmove declines linearly by area of buffer

//parea depends only on meterarea

**double** parea = meterarea/(Math.***PI***\*perception\*perception);

//parea depends on area and probEggs

//combined effect of distance and area

**double** pareadist = parea\*probMove;

//check if zoneagent is "remembered" and scale p accordingly

//according to some purists on stackflow, i might be better off

//writing a loop for this instead of coercing to List

**boolean** check = **false**;

**for**(**double** item:memories){

**if**(item == zoneagent.ID)

check = **true**;

}

//if monarch has been there before, adjust probMove

**if** (check == **true**){

//logistic equation to scale memory by area

//I used lat/long area before but now shapefile area is m^2

//now divided by 10000 to get ha

**double** newarea = zoneagent.getArea()/10000;

**double** memareascale = 1/(1+Math.*exp*(-2\*(newarea-0.920479)));

fprob = pareadist\*memareascale;

}**else**{

fprob = pareadist;

}

//System.out.println("4-fprob = " + fprob);

probs.add(fprob);

//total the p's for normalization below

runningsum = runningsum + fprob;

//System.out.println("5-runningsum = " + runningsum);

}

}

// BOUNCE BACK IF MONARCH LEAVES STUDY AREA/SHAPEFILE

// check if array has 0, if doesn't, the monarch is not in a polygon

//\*\*/

**if** (dists.contains(0.0) == **false**) {

//move back 180 degrees and 2 steps

**double** steplengthB = steplength\*2;

**double** bounceAngle = lastAngle-Math.***PI***;

**if** (bounceAngle < 0){

bounceAngle = bounceAngle + 2\*Math.***PI***;

}

geography.moveByVector(**this**, steplengthB, bounceAngle);

seq = **true**;

bouncecounter++;

//if bounces too many times, that means it probably got stuck outside, so move to random loc inside

**if** (bouncecounter > 5){

//Story Co

Coordinate coord = **new** Coordinate(-93.2319 - 0.4661\* Math.*random*(),

41.8634 + 0.3457 \* Math.*random*());

Point bouncegeom = *fac*.createPoint(coord);

geography.move(**this**, bouncegeom);

}

}

**else**{//\*\*/

seq = **false**;

bouncecounter = 0;

// CHOOSE A POLYGON

// if there is more than 1 zone (current zone monarch is in) to choose from

// choose which one to head towards

**if** (probs.size() > 1) {

//logic to determine index of target polygon

**int** whichPoly = -9999;

**double** r = Math.*random*();

//System.out.println("6-r = " + r);

**double** prevProb = 0;

**int** j = 0;

**boolean** done2 = **false**;

**while** (! done2) {

**double** pnorm = (**double**)probs.get(j)/runningsum;

//solution for what happens when hits boundary where both polygons have probEggs = 0

**if** (runningsum == 0){

//set whichpoly so code below doesn't hang but doesn't affect result

whichPoly = 0;

done2 = **true**;

}

//code for normal situations

**if** (prevProb < r && r < prevProb + pnorm) {

whichPoly = j;

//System.out.println("7-whichpoly = " + whichPoly);

//System.out.println("8-j probMove = " + probs.get(j));

done2 = **true**;

} **else** {

j++;

prevProb = prevProb + pnorm;

}

}

//check to see if the target polygon is the polygon currently

//containing the monarch - dist will be 0 if so

**double** polydist = (**double**)dists.get(whichPoly);

**if** (polydist > 0 && runningsum != 0) {

//target poly is NOT the the poly currently containing the monarch

**double** PolyY = (**double**)destY.get(whichPoly)-c1.y;

**double** PolyX = (**double**)destX.get(whichPoly)-c1.x;

//find angle to chosen Polygon

**double** geoRad = Math.*atan2*(PolyY, PolyX); //in radians

**if** (geoRad < 0){

geoRad = geoRad + 2\*Math.***PI***;

}

lastAngle = geoRad;

geography.moveByVector(**this**, steplength, geoRad);

//System.out.println("A-Move1------------ probMove Move");

} **else** {

//moves in corr rand walk if it chooses polygon it is already in

corrrandwalk2(currProbMove);

//System.out.println("B-Move2 ------------ corrwalk, chose current poly");

}

} **else** {

//moves in correlated random walk if it still has eggs or steps left

//and no other polygons in perception distance

corrrandwalk(currProbMove);

//System.out.println("C-Move3 ---------------- corrwalk, no other polys visible");

}

cumDist = cumDist + steplength;

} //end of else statement for when agent is still in a polygon

//check for doneness based on distance moved

**if** (cumDist >= maxdist) {

doneMove = **true**;

}

cumSteps++;

//apparently I have to put SOMETHING into the output file or I get an error

**if** (savecoords == **false**){

xcoords = "9999";

ycoords = "9999";

classnames = "9999";

name2 = "9999";

}

// Concatenate x,y coords and class names

**if** (savecoords == **true**){

xcoords = StringUtils.*join*(monX, ',');

ycoords = StringUtils.*join*(monY, ',');

classnames = StringUtils.*join*(className, ',');

}

///\*\*

// record new coordinate - Dr. Parry

geom = geography.getGeometry(**this**);

c2 = geom.getCoordinates()[0];

//get distance moved

**double** moveDist = 0;

CoordinateReferenceSystem crs = geography.getCRS();

**try** {

moveDist = JTS.*orthodromicDistance*(c1, c2, crs);

} **catch** (TransformException e) {

//Auto-generated catch block

e.printStackTrace();

}

//breaks links when monarch moves around torus

**if** (moveDist < 500) {

// Display path as network

displayNetwork(c1,c2);

}

//\*\*/

}

}

// METHODS

//lay eggs

**private** **double** layeggs(**double** probEggs, **double** currcumEggs){

**double** r = Math.*random*();

**double** newcumEggs = 0;

**if** (probEggs > r && dailyeggstolay > 0){

//need to account for the fact that sometimes don't have 2 eggs left to lay

**if**(dailyeggstolay > eggsperlay){

newcumEggs = currcumEggs + eggsperlay;

dailyeggslaid = dailyeggslaid + eggsperlay;

dailyeggstolay = dailyeggstolay - eggsperlay;

numtimeslayeggs--;

} **else**{

newcumEggs = currcumEggs + dailyeggstolay;

dailyeggslaid = dailyeggslaid + dailyeggstolay;

dailyeggstolay = 0;

numtimeslayeggs--;

}

**return** newcumEggs;

}

**return** currcumEggs;

}

//correlated random walk for when it chooses current polygon

**private** **void** corrrandwalk2(**double** probMove){

//retrieve any Monarchs that go outside the boundaries and return them to opposite side - this should do nothing now that bouncecounter is active

Context context = ContextUtils.*getContext*(**this**);

Geography<Monarch> geography = (Geography)context.getProjection("Monarchs");

Point loc = (Point)geography.getGeometry(**this**);

x = loc.getCoordinate().x;

y = loc.getCoordinate().y;

**if** (x < *xmin*){

Coordinate tempcoord1 = **new** Coordinate(*xmax* - (*xmin* - x), y);

Point temp1 = *fac*.createPoint(tempcoord1);

geography.move(**this**, temp1);

}

**if** (x > *xmax*){

Coordinate tempcoord2 = **new** Coordinate(*xmin* + (x - *xmax*), y);

Point temp2 = *fac*.createPoint(tempcoord2);

geography.move(**this**, temp2);

}

**if** (y < *ymin*){

Coordinate tempcoord3 = **new** Coordinate(x, *ymax* - (*ymin* - y));

Point temp3 = *fac*.createPoint(tempcoord3);

geography.move(**this**, temp3);

}

**if** (y > *ymax*){

Coordinate tempcoord4 = **new** Coordinate(x, *ymin* + (y - *ymax*));

Point temp4 = *fac*.createPoint(tempcoord4);

geography.move(**this**, temp4);

}

**double** localdir = (-(maxdir-mindir)/probMoveRange)\*probMove + maxdir;

angleChange = Math.***PI***\*RandomHelper.*nextDouble*()\*(1-localdir);

**if**(Math.*random*() < 0.5){

tempAngle1 = lastAngle + angleChange;

**if**(tempAngle1 > 2\*Math.***PI***){

currAngle2 = tempAngle1 - 2\*Math.***PI***;

}

**else** {

currAngle2 = tempAngle1;

}

} **else** {

tempAngle2 = lastAngle - angleChange;

**if**(tempAngle2 < 0){

currAngle2 = 2\*Math.***PI*** + tempAngle2;

}

**else**{

currAngle2 = tempAngle2;

}

}

lastAngle = currAngle2;

//Monarch moves

geography.moveByVector(**this**, steplength, currAngle2);

}

//Correlated random walk for when only 1 polygon in sight

**private** **void** corrrandwalk(**double** probMove){

//retrieve any Monarchs that go outside the boundaries and return them to opposite side

Context context = ContextUtils.*getContext*(**this**);

Geography<Monarch> geography = (Geography)context.getProjection("Monarchs");

Point loc = (Point)geography.getGeometry(**this**);

x = loc.getCoordinate().x;

y = loc.getCoordinate().y;

**if** (x < *xmin*){

Coordinate tempcoord1 = **new** Coordinate(*xmax* - (*xmin* - x), y);

Point temp1 = *fac*.createPoint(tempcoord1);

geography.move(**this**, temp1);

}

**if** (x > *xmax*){

Coordinate tempcoord2 = **new** Coordinate(*xmin* + (x - *xmax*), y);

Point temp2 = *fac*.createPoint(tempcoord2);

geography.move(**this**, temp2);

}

**if** (y < *ymin*){

Coordinate tempcoord3 = **new** Coordinate(x, *ymax* - (*ymin* - y));

Point temp3 = *fac*.createPoint(tempcoord3);

geography.move(**this**, temp3);

}

**if** (y > *ymax*){

Coordinate tempcoord4 = **new** Coordinate(x, *ymin* + (y - *ymax*));

Point temp4 = *fac*.createPoint(tempcoord4);

geography.move(**this**, temp4);

}

//get new angle for correlated random walk

//directionality changes according to probEggs

**double** localdir = (-(maxdir-mindir)/probMoveRange)\*probMove + maxdir;

angleChange = Math.***PI***\*RandomHelper.*nextDouble*()\*(1-localdir);

**if**(Math.*random*() < 0.5){

tempAngle1 = lastAngle + angleChange;

**if**(tempAngle1 > 2\*Math.***PI***){

currAngle = tempAngle1 - 2\*Math.***PI***;

}

**else** {

currAngle = tempAngle1;

}

} **else** {

tempAngle2 = lastAngle - angleChange;

**if**(tempAngle2 < 0){

currAngle = 2\*Math.***PI*** + tempAngle2;

}

**else**{

currAngle = tempAngle2;

}

}

lastAngle = currAngle;

//Monarch moves

geography.moveByVector(**this**, steplength, currAngle);

}

///\*\*

//Hazel's network code

**public** **void** displayNetwork(Coordinate c1,Coordinate c2){

Context context = ContextUtils.*getContext*(**this**);

Geography<Object> geography = (Geography)context.getProjection("Monarchs");

Network <Object> net = (Network <Object>) context.getProjection("travel");

net.addEdge(c1, c2);

// System.out.println("network " + net + "edge added from " + c1 + " to " + c2);

MonarchPath mp = **new** MonarchPath(net.getEdge(c1, c2),c1,c2);

context.add(mp);

Coordinate carray[] = **new** Coordinate[2];

carray[0] = c1;

carray[1] = c2;

GeometryFactory fac = **new** GeometryFactory();

LineString ls = fac.createLineString(carray);

geography.move(mp, ls);

}

//\*\*/

**public** String getName() {

**return** name;

}

**public** **double** getEggsToLay(){

**return** dailyeggstolay;

}

**public** **double** getEggsLaid(){

**return** dailyeggslaid;

}

**public** **double** getTimesLaidEggs(){

**double** timeslaideggs = numtimeslayeggs2 - numtimeslayeggs;

**return** timeslaideggs;

}

**public** **double** getcumSteps(){

**return** cumSteps;

}

**public** **double** getMaxDist(){

**return** maxdist;

}

**public** **double** getcumDist(){

**return** cumDist;

}

**public** **double** geteggslaidsteps(){

**return** eggslaidsteps;

}

//this gets the same name as getName() - the name of the monarch agent - unless savecoords == false

**public** String getName2() {

**return** name2;

}

**public** String getMonXs(){

**return** xcoords;

}

**public** String getMonYs(){

**return** ycoords;

}

**public** String getClassNames(){

**return** classnames;

}

**public** String name() {

**return** name;

}

@Override

**public** String toString() {

**return** name;

}

}