//===========================================================================

// FLOCKING POLYGONS - Version 29

// AutoRun with Multiple Timers

// 28 October 2013

//===========================================================================

//------------------------ declaration of an abstract class called agentClass

class agentClass {

public:

long double lastX, lastY, lastZ;

long double x, y, z ;

long double dx, dy, dz;

long double dxx, dyy, dzz;

TPoint twoDPoints[3];

TPoint threeDPoints[3];

TPoint boxPoints[5];

TColor color;

float height, width;

int tag;

int seeking;

void erase (void) { // size is dependent upon speed

…

}

void draw (void) {

…

}

void move (void) {

…

}

long double getDirection (void) { // ok

…

}

long double getVelocity (void) { // ok

…

}

// velocities skew towards minimal at +/- pi/2 (N & S)

// all continue to shrink on repeated changes and don't recover

void setDirection (long double dir) {

…

}

// directions skew towards the horizontal as velocities increase

// all continue to skew on repated changes and don't recover

void setVelocity (long double vel) {

…

}

};

agentClass agent[POP]; // creates an array "agent" of type "agentClass"

agentClass circles[4];

agentClass cursor;

// ==========================================================================

// ================================================================ FUNCTIONS

// ==========================================================================

// ========================================================= NEAREST NEIGHBOR

int nneighbor (int me) { // suspect

nn;

…

return nn;

}

// ======================================================= DISTANCE TO CURSOR

double distanceToCursor(int i) {

int dist;

…

return dist;

}

// =================== SUM THE NEIGHBORS VECTORS WITHIN A RADIUS FROM SOMEONE

void sumNeighborsVectorsFrom (int who) { // square, not Pythagorean, radius

…

}

// ======================================== ADJUST INFLUENCE BY INDEPENDENCE

void adjustNeighborsVectorsByIndependence (void) {

…

}

// ================================================================ ADD NOISE

void addNoise (void) {

…

}

// ===================================================================== STEP

void step (void) {

iterations ++;

lastMaxNeighbors = maxNeighbors;

maxNeighbors = 0;

for (me = 0; me < pop; me++) {

if (numberOfNeighbors > maxNeighbors) maxNeighbors = numberOfNeighbors;

// draw trails or not

if (trail == 0) {

agent[me].erase();

}

//######################### beginning of rules ##################

//######################### beginning of rules ##################

// ####################################################### RULE 1

if (rule == 1) { // adopt nearest neighbors direction and speed

agent[me].dx = agent[nneighbor(me)].dx;

agent[me].dy = agent[nneighbor(me)].dy;

}

// ####################################################### RULE 2

// adopt weighted average direction and speed of those

// within a square radius

if (rule == 2) {

// calculates numberOfNeighbors and...

sumNeighborsVectorsFrom(me);

// also calculates numberOfNeighbors

// adjusts vector sums by independence

adjustNeighborsVectorsByIndependence();

// adds noise

addNoise();

}

// ####################################################### RULE 3

if (rule == 3) { // flock to cursor

sumXdistancesFromWho = cursorX - agent[me].x;

sumYdistancesFromWho = cursorY - agent[me].y;

sumZdistancesFromWho = cursorZ - agent[me].z;

numberOfNeighbors = 1;

// adjusts vector sums by independence

adjustNeighborsVectorsByIndependence();

// adds noise

addNoise();

}

// ####################################################### RULE 4

if (rule == 4) { // flee from cursor

sumXdistancesFromWho = agent[me].x - cursorX;

sumYdistancesFromWho = agent[me].y - cursorY;

sumYdistancesFromWho = agent[me].z - cursorZ;

numberOfNeighbors = 1;

// adjusts vector sums by independence

adjustNeighborsVectorsByIndependence();

// adds noise

addNoise();

}

// ####################################################### RULE 5

if (rule == 5) { // assimilate to nearest neighbor's color

agent[me].color = agent[nneighbor(me)].color;

}

// ####################################################### RULE 6

if (rule == 6) { // follow assigned individual

if (agent0FollowsCursor && me == 0) { // if 0 follows cursor

sumXdistancesFromWho = cursorX - agent[me].x;

sumYdistancesFromWho = cursorY - agent[me].y;

sumZdistancesFromWho = cursorZ - agent[me].z;

}

else {

sumXdistancesFromWho =

agent[agent[me].seeking].x - agent[me].x;

sumYdistancesFromWho =

agent[agent[me].seeking].y - agent[me].y;

sumZdistancesFromWho =

agent[agent[me].seeking].z - agent[me].z;

}

numberOfNeighbors = 1;

// adjusts vector sums by independence

adjustNeighborsVectorsByIndependence();

// adds noise

addNoise();

}

// ####################################################### RULE 7

if (rule == 7) {

// empty

}

// ####################################################### RULE 8

if(rule == 8) { // Seek nearest neighbor

nneighbor(me); // calculates nn and dist

sumXdistancesFromWho = agent[nn].x - agent[me].x;

sumYdistancesFromWho = agent[nn].y - agent[me].y;

sumZdistancesFromWho = agent[nn].z - agent[me].z;

numberOfNeighbors = 1;

// adjusts vector sums by independence

adjustNeighborsVectorsByIndependence();

// adds noise

addNoise();

}

// ####################################################### RULE 9

if (rule == 9) { //flee nearest neighbor

nneighbor(me); // calculates nn and dist

sumXdistancesFromWho = agent[me].x - agent[nn].x;

sumYdistancesFromWho = agent[me].y - agent[nn].y;

sumZdistancesFromWho = agent[me].z - agent[nn].z;

numberOfNeighbors = 1;

// adjusts vector sums by independence

adjustNeighborsVectorsByIndependence();

// adds noise

addNoise();

}

// ###################################################### RULE 10

if (rule == 10) { // Odd (.y and .x switched from seek cursor)

sumXdistancesFromWho = cursorX - agent[me].y;

sumYdistancesFromWho = cursorY - agent[me].x;

sumZdistancesFromWho = cursorZ - agent[me].z;

numberOfNeighbors = 1;

// adjusts vector sums by independence

adjustNeighborsVectorsByIndependence();

// adds noise

addNoise();

}

// ###################################################### RULE 11

if (rule == 11) { // Sarah Van Name's Four Corners

if (agent[me].x < 840 && agent[me].y < 525) {

sumXdistancesFromWho = 420 - agent[me].x;

sumYdistancesFromWho = 262.5 - agent[me].y;

sumZdistancesFromWho = 100 - agent[me].z;

}

if (agent[me].x < 840 && agent[me].y > 525) {

sumXdistancesFromWho = 420 - agent[me].x;

sumYdistancesFromWho = 787.5 - agent[me].y;

sumZdistancesFromWho = 100 - agent[me].z;

}

if (agent[me].x > 840 && agent[me].y < 525) {

sumXdistancesFromWho = 1260 - agent[me].x;

sumYdistancesFromWho = 262.5 - agent[me].y;

sumZdistancesFromWho = 100 - agent[me].z;

}

if (agent[me].x > 840 && agent[me].y > 525) {

sumXdistancesFromWho = 1260 - agent[me].x;

sumYdistancesFromWho = 787.5 - agent[me].y;

sumZdistancesFromWho = 100 - agent[me].z;

}

dynaColor = colorRamp(125, 255, spectrum);

numberOfNeighbors = 1;

// adjusts vector sums by independence

adjustNeighborsVectorsByIndependence();

// adds noise

addNoise();

}

// ###################################################### RULE 12

if(rule == 12) { // Flee near, seek far

nneighbor(me); // calculates nn and dist

if (dist > radius) { // seek far

sumXdistancesFromWho = agent[nn].x - agent[me].x;

sumYdistancesFromWho = agent[nn].y - agent[me].y;

sumZdistancesFromWho = agent[nn].z - agent[me].z;

}

if (dist < radius) { // flee near

sumXdistancesFromWho = agent[me].x - agent[nn].x;

sumYdistancesFromWho = agent[me].y - agent[nn].y;

sumZdistancesFromWho = agent[me].z - agent[nn].z;

}

numberOfNeighbors = 1;

// adjusts vector sums by independence

adjustNeighborsVectorsByIndependence();

// adds noise

addNoise();

}

// ###################################################### RULE 13

if (rule == 13) { // Sarah Van Name's Lissajou Figure

agent[mv(me + 1)].x = 840 + .45 \* tan(me \* 20) \* 512;

agent[mv(me + 1)].y = 525 + 0.3 \* sin(me \* 20) \* 512;

sumXdistancesFromWho = agent[mv(me + 1)].x - agent[me].x;

sumYdistancesFromWho = agent[mv(me + 1)].y - agent[me].y;

sumZdistancesFromWho = agent[mv(me + 1)].z - agent[me].z;

numberOfNeighbors = 1;

// weight vectors by independence

adjustNeighborsVectorsByIndependence();

// add noise

addNoise();

}

// ###################################################### RULE 14

if(rule == 14) { // Flee Larger, Seek Smaller

if(agent[nneighbor(me)].getVelocity() > agent[me].getVelocity())

{

sumXdistancesFromWho = -1 \* (agent[nneighbor(me)].x

- agent[me].x);

sumYdistancesFromWho = -1 \* (agent[nneighbor(me)].y

- agent[me].y);

sumZdistancesFromWho = -1 \* (agent[nneighbor(me)].z

- agent[me].z);

}

else {

sumXdistancesFromWho =

agent[nneighbor(me)].x - agent[me].x;

sumYdistancesFromWho =

agent[nneighbor(me)].y - agent[me].y;

sumZdistancesFromWho =

agent[nneighbor(me)].z - agent[me].z;

}

numberOfNeighbors = 1;

// weight vectors by independence

adjustNeighborsVectorsByIndependence();

// add noise

addNoise();

}

// ###################################################### RULE 15

if (rule == 15) { // circle: seek radius from cursor

if (distanceToCursor(me) < radius) {

// flee

sumXdistancesFromWho = agent[me].x - cursorX;

sumYdistancesFromWho = agent[me].y - cursorY;

}

else {

// flock

sumXdistancesFromWho = cursorX - agent[me].x;

sumYdistancesFromWho = cursorY - agent[me].y;

}

numberOfNeighbors = 1;

// weight vectors by independence

adjustNeighborsVectorsByIndependence();

// add noise

addNoise();

}

// ###################################################### RULE 16

if (rule == 16) { // Gravitational attraction to cursor

sumSidesSquared = pow(cursorX - agent[me].x, 2) +

pow(cursorY -agent[me].y, 2);

gravForce = (radius \* 10000) / sumSidesSquared;

gravHypot = sqrt(sumSidesSquared);

sumXdistancesFromWho = (gravForce / gravHypot) \*

(cursorX - agent[me].x);

sumYdistancesFromWho = (gravForce / gravHypot) \*

(cursorY - agent[me].y);

numberOfNeighbors = 1;

// adjusts vector sums by independence

adjustNeighborsVectorsByIndependence();

// adds noise

addNoise();

}

// ###################################################### RULE 17

if (rule == 17) { // Orbits 1 with Sun at Center

// gravityScaler is symonymous with independence

// z is ignored

orbits1sunDistance = sqrt(pow(orbits1sun.x - agent[me].x, 2)

+ pow(orbits1sun.y - agent[me].y, 2));

agent[me].dx -= (gravityScaler) / pow(orbits1sunDistance, 3)

\* (agent[me].x - orbits1sun.x);

agent[me].dy -= (gravityScaler) / pow(orbits1sunDistance, 3)

\* (agent[me].y - orbits1sun.y);

// color Sun

worldBMP->Canvas->Pen->Color = clYellow;

worldBMP->Canvas->Brush->Color = clYellow;

worldBMP->Canvas->Ellipse

(orbits1sun.x - 10, orbits1sun.y -10,

orbits1sun.x + 10, orbits1sun.y + 10);

}

// ###################################################### RULE 18

if (rule == 18) { // Orbits 2 with two Suns

// gravityScaler is symonymous with independence

// z is ignored

orbits2sun1Distance = sqrt(pow(orbits2sun1.x - agent[me].x, 2)

+ pow(orbits2sun1.y - agent[me].y, 2));

orbits2sun2Distance = sqrt(pow(orbits2sun2.x - agent[me].x, 2)

+ pow(orbits2sun2.y - agent[me].y, 2));

agent[me].dx -= ((gravityScaler) / pow(orbits2sun1Distance, 3)

\* (agent[me].x - orbits2sun1.x))

+ ((gravityScaler) / pow(orbits2sun2Distance, 3)

\* (agent[me].x - orbits2sun2.x));

agent[me].dy -= ((gravityScaler) / pow(orbits2sun1Distance, 3)

\* (agent[me].y - orbits2sun1.y))

+ ((gravityScaler) / pow(orbits2sun2Distance, 3)

\* (agent[me].y - orbits2sun2.y));

// color two Suns

worldBMP->Canvas->Pen->Color = clYellow;

worldBMP->Canvas->Brush->Color = clYellow;

worldBMP->Canvas->Ellipse

(orbits2sun1.x -10, orbits2sun1.y -10,

orbits2sun1.x +10, orbits2sun1.y +10);

worldBMP->Canvas->Ellipse

(orbits2sun2.x -10, orbits2sun2.y -10,

orbits2sun2.x +10, orbits2sun2.y +10);

}

// ###################################################### RULE 18

if (rule == 19) { // Orbits 3 with one Sun, Mars and Jupiter

// gravityScaler is symonymous with independence

// z is ignored

// clockwork Mars orbit

orbits3mars.x = 0.2279 \* .5 \* monitorHeight

\* cos(iterations / 227.9) + orbits3sun.x;

orbits3mars.y = 0.2279 \* .5 \* monitorHeight

\* sin(iterations / 227.9) + orbits3sun.y;

// clockwork Jupiter orbit

orbits3jupiter.x = 0.7786 \* .5 \* monitorHeight

\* cos(iterations / 778.6) + orbits3sun.x;

orbits3jupiter.y = 0.7786 \* .5 \* monitorHeight

\* sin(iterations / 778.6) + orbits3sun.y;

// asteroid orbits

orbits3sunDistance = sqrt(pow(orbits3sun.x - agent[me].x, 2)

+ pow(orbits3sun.y - agent[me].y, 2));

orbits3marsDistance = sqrt(pow(orbits3mars.x - agent[me].x, 2)

+ pow(orbits3mars.y - agent[me].y, 2));

orbits3jupiterDistance = sqrt(pow(orbits3jupiter.x

- agent[me].x, 2)

+ pow(orbits3jupiter.y - agent[me].y, 2));

agent[me].dx -=

((gravityScaler) / pow(orbits3sunDistance, 3)

\* (agent[me].x - orbits3sun.x))

+ ((0.1 \* gravityScaler) / pow(orbits3marsDistance, 3)

\* (agent[me].x - orbits3mars.x))

+ ((0.3 \* gravityScaler) / pow(orbits3jupiterDistance, 3)

\* (agent[me].x - orbits3jupiter.x));

agent[me].dy -=

((gravityScaler) / pow(orbits3sunDistance, 3)

\* (agent[me].y - orbits3sun.y))

+ ((0.1 \* gravityScaler) / pow(orbits3marsDistance, 3)

\* (agent[me].y - orbits3mars.y))

+ ((0.3 \* gravityScaler) / pow(orbits3jupiterDistance, 3)

\* (agent[me].y - orbits3jupiter.y));

// color sun and planets

worldBMP->Canvas->Pen->Color = clYellow;

worldBMP->Canvas->Brush->Color = clYellow;

worldBMP->Canvas->Ellipse

(orbits3sun.x -10, orbits3sun.y -10,

orbits3sun.x +10, orbits3sun.y +10);

worldBMP->Canvas->Pen->Color =

colorRamp(iterations, colorRange, 5);

worldBMP->Canvas->Brush->Color =

colorRamp(iterations, colorRange, 5);

worldBMP->Canvas->Ellipse

(orbits3mars.x -2, orbits3mars.y -2,

orbits3mars.x +2, orbits3mars.y +2);

worldBMP->Canvas->Pen->Color =

colorRamp(iterations, 2.7 \* colorRange, 2);

worldBMP->Canvas->Brush->Color =

colorRamp(iterations, 2.7 \* colorRange, 2);

worldBMP->Canvas->Ellipse

(orbits3jupiter.x -2, orbits3jupiter.y -2,

orbits3jupiter.x +2, orbits3jupiter.y +2);

}

//############################### end of rules ##################

//############################### end of rules ##################

if (obstacles) {

if (Form1->Shape1->Visible &&

(((agent[me].x - Form1->Shape1->Left < 10)

&& (agent[me].x - Form1->Shape1->Left > 0))

|| ((agent[me].x - (Form1->Shape1->Left

+ Form1->Shape1->Width) > -10)

&& (agent[me].x - (Form1->Shape1->Left

+ Form1->Shape1->Width) < 0)))

&& ((agent[me].y > Form1->Shape1->Top)

&& (agent[me].y < Form1->Shape1->Top

+ Form1->Shape1->Height)))

{

agent[me].dx = -agent[me].dx;

}

if (Form1->Shape1->Visible &&

(((agent[me].y - Form1->Shape1->Top < 10)

&& (agent[me].y - Form1->Shape1->Top > 0))

|| ((agent[me].y - (Form1->Shape1->Top

+ Form1->Shape1->Height) > -10)

&& (agent[me].y - (Form1->Shape1->Top

+ Form1->Shape1->Height) < 0)))

&& ((agent[me].x > Form1->Shape1->Left)

&& (agent[me].x < Form1->Shape1->Left

+ Form1->Shape1->Width)))

{

agent[me].dy = -agent[me].dy;

}

if (Form1->Shape2->Visible &&

(((agent[me].x - Form1->Shape2->Left < 10)

&& (agent[me].x - Form1->Shape2->Left > 0))

|| ((agent[me].x - (Form1->Shape2->Left

+ Form1->Shape2->Width) > -10)

&& (agent[me].x - (Form1->Shape2->Left

+ Form1->Shape2->Width) < 0)))

&& ((agent[me].y > Form1->Shape2->Top)

&& (agent[me].y < Form1->Shape2->Top

+ Form1->Shape2->Height)))

{

agent[me].dx = -agent[me].dx;

}

if (Form1->Shape2->Visible &&

(((agent[me].y - Form1->Shape2->Top < 10)

&& (agent[me].y - Form1->Shape2->Top > 0))

|| ((agent[me].y - (Form1->Shape2->Top

+ Form1->Shape2->Height) > -10)

&& (agent[me].y - (Form1->Shape2->Top

+ Form1->Shape2->Height) < 0)))

&& ((agent[me].x > Form1->Shape2->Left)

&& (agent[me].x < Form1->Shape2->Left

+ Form1->Shape2->Width)))

{

agent[me].dy = -agent[me].dy;

}

if (Form1->Shape3->Visible &&

(((agent[me].x - Form1->Shape3->Left < 10)

&& (agent[me].x - Form1->Shape3->Left > 0))

|| ((agent[me].x - (Form1->Shape3->Left

+ Form1->Shape3->Width) > -10)

&& (agent[me].x - (Form1->Shape3->Left

+ Form1->Shape3->Width) < 0)))

&& ((agent[me].y > Form1->Shape3->Top)

&& (agent[me].y < Form1->Shape3->Top

+ Form1->Shape3->Height)))

{

agent[me].dx = -agent[me].dx;

}

if (Form1->Shape3->Visible &&

(((agent[me].y - Form1->Shape3->Top < 10)

&& (agent[me].y - Form1->Shape3->Top > 0))

|| ((agent[me].y - (Form1->Shape3->Top

+ Form1->Shape3->Height) > -10)

&& (agent[me].y - (Form1->Shape3->Top

+ Form1->Shape3->Height) < 0)))

&& ((agent[me].x > Form1->Shape3->Left)

&& (agent[me].x < Form1->Shape3->Left

+ Form1->Shape3->Width)))

{

agent[me].dy = -agent[me].dy;

}

}

// end of all rules

agent[me].move();

} // end of poll everyone in current population

}