FLOCKING POLYGONS AND ORBITS  
Some important variables and functions

class agentClass {

public:

long double lastX, lastY, lastZ;

long double x, y, z ;

long double dx, dy, dz;

TColor color;

bool dontPaintMe;

float height, width;

float scale; // for visualized shape size

int tag, seeking;

int nn, nnr;

int age, sex, hunger, wealth, live;

void erase (void) {

void draw (void) {

void move (void) {

long double getDirection (void) { // ok

long double getVelocity (void) { // ok

void setDirection (long double dir) { // problematic

void setVelocity (long double vel) { // problematic

}

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

int nneighbor (int me) { // suspect?

int nneighborR (int me) { // suspect?

double distanceToCursor(int i) {

double distanceFromTo (int from, int to) {

// =================== SUM THE NEIGHBORS VECTORS WITHIN A RADIUS FROM “WHO”

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

numberOfNeighbors = 0;

sumXdistancesFromWho = 0;

sumYdistancesFromWho = 0;

sumZdistancesFromWho = 0;

sum\_dH = 0;

xDistanceFromWho = 0; // are they close enough to count?

yDistanceFromWho = 0; // are they close enough to count?

zDistanceFromWho = 0; // are they close enough to count?

for (int neigh = 0; neigh < pop; neigh++) {

// look at everyone except “who” who is less than “radius” away

if (neigh == who) continue; // but skip who

xDistanceFromWho = fabs(agent[neigh].x - agent[who].x);

yDistanceFromWho = fabs(agent[neigh].y - agent[who].y);

zDistanceFromWho = fabs(agent[neigh].z - agent[who].z);

if ((xDistanceFromWho < radius) && (yDistanceFromWho < radius)

&& (zDistanceFromWho < radius)) {

numberOfNeighbors++;

sumXdistancesFromWho += agent[neigh].dx;

sumYdistancesFromWho += agent[neigh].dy;

sumZdistancesFromWho += agent[neigh].dz;

}

}

}

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

void adjustNeighborsVectorsByIndependence (void) {

adjustedAveSumXdistances =

(independence \* agent[me].dx + sumXdistancesFromWho)

/ (numberOfNeighbors + independence);

adjustedAveSumYdistances =

(independence \* agent[me].dy + sumYdistancesFromWho)

/ (numberOfNeighbors + independence);

adjustedAveSumZdistances =

(independence \* agent[me].dz + sumZdistancesFromWho)

/ (numberOfNeighbors + independence);

if (adjustedAveSumXdistances == 0)

adjustedAveSumXdistances = .00001;

if (adjustedAveSumYdistances == 0)

adjustedAveSumYdistances = .00001;

if (adjustedAveSumZdistances == 0)

adjustedAveSumZdistances = .00001;

}

void addNoise (void) {

void step (void) {