Robot World: A Evolution Simulation

Blake Bryce Bredehoft

January 25, 2005

Abstract: My project is an agent based simulation, posing robots in a “game of life”, with each new generation of robot comes new genes using a random number selection process creating the mutations and evolutions that in real life we experience for DNA cross over and such. There are two versions of my program a Simulation and a Game.

The Simulation: As stated in my abstract. The Process: The base of my program was not the genetics, but the graphics itself. First one robot, then a random Artificial Intelligence for it. I then modified the world to sustain several robots, with dying a breeding. After introducing two more Artificial Intelligences a “group” Artificial Intelligence and a “battery” Artificial Intelligence promoting grouping and collecting batteries selectively. I then tweaked the environment and code until it could sustain life. I then programed in several possible places for environmental interaction, like viruses and the batteries. I added the graphical output for easier analysis. Finally I created the random number selection process to splice the genes of the parents and create a child. The heart of the program.

The Game: There is the above stated “simulation” version of my program, and a later created “game” version. The game version includes all the same components as the simulation but also has a user controlled robot with “laser eyes” and “grenade launchers” use to kill the other robots. It also includes “Bosses”.The Process: Taking the simulation version and modifying it was easy, first removing the natural deaths and graph. Then I introduced and tweaked the user controls and the user controlled robot. Then adding lasers and grenades and all the necessary coding, i finally added a status display embedded window in the top left corner. I continually add new pieces of flare to the program, such as bosses.

Robot World: A Evolution Simulation

Blake Bryce Bredehoft

January 25, 2005

1 Introduction

My project has several basic components: 3D modeling, Artificial Intelligence, The selection process, and then basic game theory. All these components form the amalgam that makes up my project. Both the simulation and the game use all these components except the simulation doesn't have any game theory.

2 Background

2.1 Monte Carlo Simulation

Since Monte Carlo is a Simulation technique, let's first define exactly what we mean by Simulation. A true Simulation will merely describe a system, not optimize it! (However, it should be noted that a true simulation may be modified in a manner such that it can be used to significantly enhance the efficiency of a system.) Therefore, our primary goal in Simulation is to build an experimental model that will accurately and precisely describe the real system. However, the breadth and extent of Simulation models is extensive! This can be illustrated by considering the three general "classifications" of Simulation Models, below. And in each of these "classifications", I have defined two possible "characteristics".

1. Functional Classification
   Deterministic Characteristic - These are "exact" models that will produce the same outcome each time they are run.
   Stochastic Characteristic - These models include some "randomness" that may produce a different outcome each time it is run. This randomness forces us to make a large number of runs to develop a "trend" in our "collection" of outcomes. Further, the exact number of how many "runs" you must make to obtain the "right trend" is simply a matter of statistics.
2. Time Dependence
   Static Characteristic - These models are not time-dependent. This even includes the calculation of a specific variable after a fixed period of time.
   Dynamic Characteristic - These models depict the change in a system over many time intervals during the calculation process.
3. Input Data
   Discrete Characteristic - The input data form a discrete frequency distribution.

Discrete frequency distributions are characterized by the random variable X taking on an enumerable number of values xi that each have a corresponding frequency, or count, pi.

Continuous Characteristic - The input data can be described by a continuous frequency distribution. Continuous frequency distributions are characterized by a continuous analytical function of the form y = *f* (x) where y is defined as the frequency of x. This definition is valid for all possible values of x (over the domain of the function).

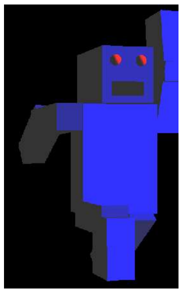
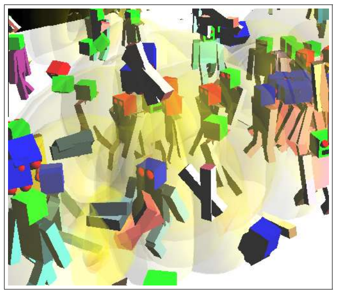
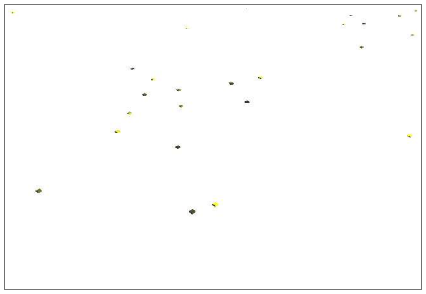
We can now say that Monte Carlo Simulations are "True Stochastic Simulations" in that they describe the "final state" of a model by just knowing the frequency distributions of the parameters describing the "beginning state" and the appropriate metric that maps, or transforms, the beginning state to the final state. They can also be either static (easy) or dynamic (more difficult). If a prediction were required, then "every possible" option would have to be considered and this is where the well-known "Variance Reduction Methods" (antithetic variables, correlated sampling, geometry splitting, source biasing, etc.) would be used to reduce the number of iterations required in the simulation. Definition courtesy of JAMES F. WRIGHT, Ph.D Ltd. Co. at http://www.drjfwright.com/.

3 Theory

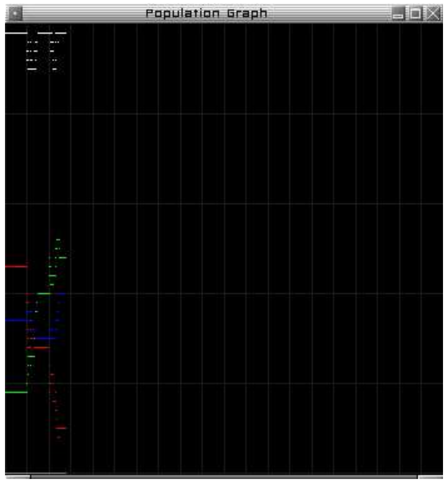
3.1 3D Modeling

My graphics are done using OpenGL. In the simulation version there are three different aspects to the graphical output: the agents (the robots), the environment (the floor and batteries), and the events (explosions) and the population graph. The game version has all these same component except the agents include a user controlled robot and bosses, the events include grenades, lasers and grenade explosions, there is no graph, and there is a stat indicator, and a mini map. There is also the interface of my program from where you launch the simulations and games.

The environment consist of a floor and background and small cubes representing batteries. Simple enough (below right). The robots consist of prisms and spheres to form arms, legs, torso, head and facial features (below left). The explosions are tori spinning on the y-axis that get more transparent as the grow in size (below center). The is also a program that is able to output numbers for the counters in the game version. See appendix A.1 for example code.

There is also a graph out put that is fairly simple. Every iteration it plots a new point on the grid, and never erases, therefore creating a line graph for the populations of each artificial intelligence type (below left). The game mode utilizes a mini map and a status bar, the bar includes life, number of grenades, and number of kills (bottom right).



Robot Civilization

3.2 Artificial Intelligence

There are three different main artificial intelligences in the program and one that uses a combination of the others. The first is a random artificial intelligence that is the most basic, second is the group artificial intelligence that condones forming groups for reproduction, and the last is the battery or food artificial intelligence that promotes eating. The fourth is one that is advanced, and has the agent use the battery artificial intelligence when it requires energy, and uses the group artificial intelligence when he doesn't require energy.

The random artificial intelligence first randomly decides whether to turn left, right, or go forward. It has a preference to turn if it turned the iteration before. This done over time produces interesting behavior. The fact that offspring spawn close to parents and that parents have to be close to produce offspring means that after a while these will group, and any robot that randomly strays from the group will die off, where as those in the group re spawn as fast as they die. Code is located in Appendix A.2.

The group artificial intelligence goes through the list of robots and first recognizes all the robots that are of a color suitable for reproduction for the given robot. Then from this list the closest robot is chosen, the robot then turns towards this robot and walks. This obviously forms groups that are more efficient than the groups produced by the random artificial intelligence, because robots will not stray off. Code is located in Appendix A.2.

The battery artificial intelligence goes through the list of batteries and finds the one closest to the robot and then turns the robot towards it and walks. Robots will end up heading after the same battery and form groups, these groups may or may not be able to reproduce though, but when a pair find each other these groups produce to be fairly strong. Code is located in Appendix A.2.

The group that is the strongest is an amalgamation of both group artificial intelligent robots and battery artificial intelligent robots. These groups stay together due to the number of group robots and will search for food due to the battery robots. Proving to be extremely effective in keeping alive. Sooner or later however one of the artificial intelligences ends up getting bred out.

There is a fourth artificial intelligence that is an amalgamation of the group artificial intelligence and the battery artificial intelligence. When the agent is low on energy it uses the battery artificial intelligence, until it has a decent amount then it uses the group artificial intelligence.

3.3 Selection Process

The selection process doesn't start with selecting but instead starts at the beginning of every iteration. The process is began by inventorying the population, tallying the number of robots and their characteristics, this then produces a few tables stored as a "genome" (code for class in Appendix A.3). This tables are formed to make graphs of the frequency of the specific gene settings.

When the selection is called, it goes thorough and finds the optimal gene, in the parents gene pool, using a random number process, and the fore mentioned graphs. This is done for each gene. There is also a level of randomness calculated in the allows for mutations. These mutations give a new status to the gene, not in the gene pool. Code can be seen in Appendix A.3.

While the theory behind this selection process may seem somewhat simple the code on the other hand is not.

3.4 Game Theory

There are lasers and grenades. Your tools for destroying the surrounding robot population. Combine their power by shooting the grenade as it falls with your laser and produce a powerful explosion. After every 50 kills boss robots appear. One will appear the first 50, two the second 50 and so on. The bosses use a boss artificial intelligence of their own.

The boss artificial intelligence will find the user controlled robot a turn it toward it and walk. When lasers are charged the boss will shoot at the user after waiting a short period. In this short period the user, if clever, can move out of the path of the lasers. One hit will kill the boss similar to the other robots.

The game ends when the user's controlled robot dies. It gets damaged when a robot comes in contacted with it or if it is hit by a boss's laser.

The score is the number of kills, as well as a ratio of kills per unit time. In the ratio is the real information, the efficiency of the user is much more impressive than the total number of kills.

A Appendix

A.1 Code

```
void body(double red, double green, double blue, Robot & numrobot)
{
 glColor3f(red,green,blue);
 glPushMatrix();
   glScalef(1.5,2.0,0.5);
   glutSolidCube(1.0);
 glPopMatrix();
}
void head(double red, double green, double blue, Robot & numrobot)
{
 glColor3f(red,green,blue);
 glPushMatrix();
   glTranslatef(0.0, 1.5, 0.0);
   glutSolidCube(1);
          glPushMatrix();
            glColor3f(0.0,0.0,0.0);
            glTranslatef(0.0, -0.25, 0.5);
            glScalef(2.5,1.0,0.1);
            glutSolidCube(.25);
          glPopMatrix();
   if(numrobot.hunger!=0)glColor3f(1/numrobot.hunger, 0.0, 0.0);
   else glColor3f(1.0, 0.0, 0.0);
   glTranslatef(0.0, 0.25,0.5);
          glPushMatrix();
            glTranslatef(0.25, 0.0,0.0);
            glutSolidSphere( 0.2*numrobot.sight,  20,  20);
          glPopMatrix();
          glPushMatrix();
            glTranslatef(-0.25, 0.0,0.0);
            glutSolidSphere( 0.2*numrobot.sight,  20,  20);
          glPopMatrix();
 glPopMatrix();
}
void rightarm(double red, double green, double blue, Robot & numrobot)
{
glColor3f(red,green,blue);
 glPushMatrix();
   glTranslatef(0.0, 0.75, 0.0);
            glPushMatrix();
              glRotatef(numrobot.Rarmspin1, 1.0, 0.0, 0.0);
                   glPushMatrix();
                     glTranslatef(-1.0,-1*numrobot.armlength/3, 0.0);
                     glScalef(0.5,numrobot.armlength,0.5);
                     glutSolidCube(1.0);
                   glPopMatrix();
                   glPushMatrix();
                     glTranslatef(0.0,-2*numrobot.armlength/3, 0.0);
                     glRotatef(numrobot.Rarmspin2, 1.0, 0.0, 0.0);
                     glTranslatef(-1.0,-1*numrobot.armlength/3, 0.0);
                     glScalef(0.5,numrobot.armlength,0.5);
                     glutSolidCube(1.0);
                   glPopMatrix();
```

```
        glPopMatrix();
  glPopMatrix();
}
void leftarm(double red, double green, double blue, Robot & numrobot)
{
glColor3f(red,green,blue);
  glPushMatrix();
    glTranslatef(0.0, 0.75, 0.0);
        glPushMatrix();
          glRotatef(numrobot.Larmspin1, 1.0, 0.0, 0.0);
                glPushMatrix();
                  glTranslatef(1.0,-1*numrobot.armlength/3, 0.0);
                  glScalef(0.5,numrobot.armlength,0.5);
                  glutSolidCube(1.0);
                glPopMatrix();
                glPushMatrix();
                  glTranslatef(0.0,-2*numrobot.armlength/3, 0.0);
                  glRotatef(numrobot.Larmspin2, 1.0, 0.0, 0.0);
                  glTranslatef(1.0,-1*numrobot.armlength/3, 0.0);
                  glScalef(0.5,numrobot.armlength,0.5);
                  glutSolidCube(1.0);
                glPopMatrix();;
        glPopMatrix();
  glPopMatrix();
}

void Rleg(Robot & numrobot)
{
glPushMatrix();
glRotatef(numrobot.Rlegspin1, 1.0, 0.0, 0.0);
        glPushMatrix();
          glTranslatef(-0.5, -.1*numrobot.leglength, 0.0);
          glScalef(0.5,numrobot.leglength*.23,0.5);
          glutSolidCube(1.0);
        glPopMatrix();
        glPushMatrix();
          glTranslatef(-0.5, -.2*numrobot.leglength, 0.0);
          glRotatef(numrobot.Rlegspin2, 1.0, 0.0, 0.0);
          glTranslatef(0.0, -.1*numrobot.leglength, 0.0);
          glScalef(0.5,numrobot.leglength*.23,0.5);
          glutSolidCube(1.0);
        glPopMatrix();
glPopMatrix();
}
void Lleg(Robot & numrobot)
{
glPushMatrix();
       glRotatef(numrobot.Llegspin1, 1.0, 0.0, 0.0);
        glPushMatrix();
          glTranslatef(0.5, -.1*numrobot.leglength, 0.0);
          glScalef(0.5,numrobot.leglength*.23,0.5);
          glutSolidCube(1.0);
        glPopMatrix();
        glPushMatrix();
          glTranslatef(0.5, -.2*numrobot.leglength, 0.0);
          glRotatef(numrobot.Llegspin2, 1.0, 0.0, 0.0);
          glTranslatef(0.0, -.1*numrobot.leglength, 0.0);
          glScalef(0.5,numrobot.leglength*.23,0.5);
```

```
              glutSolidCube(1.0);
            glPopMatrix();
          glPopMatrix();
}
void legs(double red, double green, double blue, Robot & numrobot)
{
glColor3f(red,green,blue);
  glPushMatrix();
    glTranslatef(0.0, -1.0, 0.0);
          Rleg(numrobot);
          Lleg(numrobot);
  glPopMatrix();
}
void drawExplosion(double red, double green, double blue, int numrobot)
{
if(DeadRoboto[numrobot].death>10||deadrobotnum>maxrobot*.8)destroyRobot2(numrobot);
DeadRoboto[numrobot].death+=.25;
DeadRoboto[numrobot].isdead.continexplode();
DeadRoboto[numrobot].bodyspin1+=((float)(rand()%100))/10;
DeadRoboto[numrobot].bodyspin2+=((float)(rand()%100))/10;
glPushMatrix();
            glTranslatef(DeadRoboto[numrobot].xmove, -2+(.4*DeadRoboto[numrobot].leglength),
DeadRoboto[numrobot].zmove);
            glPushMatrix();
                  glRotatef(DeadRoboto[numrobot].bodyspin1, 0.0, 1.0, 0.0);
                  glColor4f(1,1,0,1/DeadRoboto[numrobot].death-.01);
                  glutSolidTorus(2,(DeadRoboto[numrobot].death*DeadRoboto[numrobot].death)/
20,20,20);
            glPopMatrix();
            glPushMatrix();
                  glRotatef(-1*DeadRoboto[numrobot].bodyspin1, 0.0, 1.0, 0.0);
                  glColor4f(1,.5,0,1/DeadRoboto[numrobot].death-.01);
                  glutSolidTorus(2,(DeadRoboto[numrobot].death*DeadRoboto[numrobot].death)/20-
1,20,20);
            glPopMatrix();
glPushMatrix();
      if(DeadRoboto[numrobot].isdead.hpy<-4)DeadRoboto[numrobot].isdead.hvy*=-.8;
                  glPushMatrix();
                        glTranslatef(DeadRoboto[numrobot].isdead.hpx,DeadRoboto[numrobot].
isdead.hpy,DeadRoboto[numrobot].isdead.hpz);
                        glRotatef(DeadRoboto[numrobot].bodyspin1, 0.0, 1.0, 0.0);
                        glRotatef(DeadRoboto[numrobot].bodyspin2, 1.0, 0.0, 0.0);
                        if(DeadRoboto[numrobot].AITYPE==1)head(1,0,0,DeadRoboto[numrobot]);
                        if(DeadRoboto[numrobot].AITYPE==2)head(0,0,1,DeadRoboto[numrobot]);
                        if(DeadRoboto[numrobot].AITYPE==3)head(0,1,0,DeadRoboto[numrobot]);
                        if(DeadRoboto[numrobot].AITYPE==4)head(0,0,0,DeadRoboto[numrobot]);
          glPopMatrix();

            if(DeadRoboto[numrobot].isdead.bpy<-3)DeadRoboto[numrobot].isdead.bvy*=-.8;
                  glPushMatrix();
                        glTranslatef(DeadRoboto[numrobot].isdead.bpx,DeadRoboto[numrobot].
isdead.bpy,DeadRoboto[numrobot].isdead.bpz);
                        glRotatef(DeadRoboto[numrobot].bodyspin1, 0.0, 1.0, 0.0);
                        glRotatef(DeadRoboto[numrobot].bodyspin2, 1.0, 0.0, 0.0);
                        body(red,green,blue,DeadRoboto[numrobot]);
          glPopMatrix();
          if(DeadRoboto[numrobot].isdead.apy<-3)DeadRoboto[numrobot].isdead.avy*=-.8;
                  glPushMatrix();
```

```
                                glTranslatef(DeadRoboto[numrobot].isdead.apx,DeadRoboto[numrobot].
isdead.apy,DeadRoboto[numrobot].isdead.apz);
                                glRotatef(DeadRoboto[numrobot].bodyspin1, 0.0, 1.0, 0.0);
                                glRotatef(DeadRoboto[numrobot].bodyspin2, 1.0, 0.0, 0.0);
                                rightarm(red,green,blue,DeadRoboto[numrobot]);
                glPopMatrix();
                        glPushMatrix();
                                glTranslatef(-1*DeadRoboto[numrobot].isdead.apx,DeadRoboto[numrobot].
isdead.apy,-1*DeadRoboto[numrobot].isdead.apz);
                                glRotatef(DeadRoboto[numrobot].bodyspin1, 0.0, 1.0, 0.0);
                                glRotatef(DeadRoboto[numrobot].bodyspin2, 1.0, 0.0, 0.0);
                                leftarm(red,green,blue,DeadRoboto[numrobot]);
                glPopMatrix();
                glRotatef(DeadRoboto[numrobot].bodyspin1, 0.0, 1.0, 0.0);
                glRotatef(DeadRoboto[numrobot].bodyspin2, 1.0, 0.0, 0.0);
                legs(red,green,blue,DeadRoboto[numrobot]);
glPopMatrix();
glPopMatrix();
}
void drawRobot(double red, double green, double blue, Robot & numrobot) // Display routine for Window
{
  glPushMatrix();
    glTranslatef(numrobot.xmove, -2+(.4*numrobot.leglength), numrobot.zmove);
    glRotatef(numrobot.bodyspin1, 0.0, 1.0, 0.0);
    body(red,green,blue,numrobot);
    if(AITYPE==4)
                {
                if(numrobot.AITYPE==1)head(1,0,0,numrobot);
                if(numrobot.AITYPE==2)head(0,0,1,numrobot);
                if(numrobot.AITYPE==3)head(0,1,0,numrobot);
                if(numrobot.AITYPE==4)head(0,0,0,numrobot);
                }
    else head(red,green,blue,numrobot);
    rightarm(red,green,blue,numrobot);
    leftarm(red,green,blue,numrobot);
    legs(red,green,blue,numrobot);
  glPopMatrix();
}

void drawBattery(int numbatt)
{
glPushMatrix();
    glTranslatef(Batts[numbatt].xmove, -3, Batts[numbatt].zmove);
    glColor3f((float)Batts[numbatt].Power()/100,(float)Batts[numbatt].Power()/100,0);
    glutSolidCube(.25);
glPopMatrix();
}

void drawfloor(void) // Display routine for Window
{
   glPushMatrix();  // Flat Blue Box
       glTranslatef( 0.0, -3, 0.0);
        glScalef(maxx*2.1, .1, maxz*2.1);
        glColor3f(0.35, 0.5, 1.0);
        glutSolidCube(1.0);
      glPopMatrix();
}
void graphit()
```

```
{
glColor3f(.1, 0.1, 0.1);
if(timer==1)
            {
            for(int i=0;i<=maxrobot;i+=10)
                        {
                        glBegin(GL_LINES);
                        glVertex3f(0,i, 0.0);
                        glVertex3f(maxtime,i, 0.0);
                        glFlush();
                        glEnd();
                        }
            for(int i=0;i<=maxtime;i+=100)
                        {
                        glBegin(GL_LINES);
                        glVertex3f(i,0, 0.0);
                        glVertex3f(i,maxrobot, 0.0);
                        glFlush();
                        glEnd();
                        }
            }

                        // Set drawing color to yellow
            glBegin( GL_POINTS );                // Start drawing POINTS
glColor3f(1.0, 0.0, 0.0);
   glVertex3f(timer,population[0][timer], 0.0);
glColor3f(0.0, 0.0, 1.0);
   glVertex3f(timer,population[1][timer], 0.0);
glColor3f(0.0, 1.0, 0.0);
   glVertex3f(timer,population[2][timer], 0.0);
glColor3f(0.5, 0.5, 0.5);
   glVertex3f(timer,population[3][timer], 0.0);
glColor3f(1.0, 1.0, 1.0);
   glVertex3f(timer,population[4][timer], 0.0);


    // Plot point at (x,y) z=0 in 2D
             glutSwapBuffers();
            glFlush();  // Force writing all pending OpenGL actions to the screen.
  glEnd();
}
void display()  // Initial Window 1 display calls the other routine
{
  glClearColor(0.0, 0.0, 0.0, 1.0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
   glTranslatef(0.0, 0.0, 0.0);
   drawfloor();
   for(int i=0;i<robotnum;i++)
   {
            glPushMatrix();
            drawRobot(Roboto[i].RED,Roboto[i].GREEN,Roboto[i].BLUE,Roboto[i]);
            glPopMatrix();
   }
   for(int i=0;i<deadrobotnum;i++)
   {
            glPushMatrix();
             drawExplosion(DeadRoboto[i].RED,DeadRoboto[i].GREEN,DeadRoboto[i].BLUE,i);
            glPopMatrix();
    }
    for(int i=0;i<battnum;i++)
    {
            glPushMatrix();
            drawBattery(i);
            glPopMatrix();
    }
    glPopMatrix();
  glFlush();
  glutSwapBuffers();
}
```

A.2 Code

```
void RANDOMAI(int numrobot)
{
Roboto[numrobot].mating++;
int action= rand()%100;
if(Roboto[numrobot].meml<.5)Roboto[numrobot].meml=.9;
if(Roboto[numrobot].memr>.5)Roboto[numrobot].memr=.1;
if(action>(100*Roboto[numrobot].meml))
            {
            Roboto[numrobot].meml-=.01;
            Roboto[numrobot].bodyspin1+=3.0;
            Roboto[numrobot].memr=.90;
            }
else if(action<(100*Roboto[numrobot].memr))
            {
            Roboto[numrobot].memr+=.01;
            Roboto[numrobot].bodyspin1-=3.0;
            Roboto[numrobot].meml=.90;
            }
else
            {
            walk(1,Roboto[numrobot]);
            }

}


void BATTRYAI(int numrobot)
{
Roboto[numrobot].mating++;
int closest=-1;
float zcdist,xcdist,zdist,xdist;
if(battnum>0)
            {
for(int i=0;i<battnum;i++)
            {
            if(closest==-1)
                                {
                                closest=i;
                                zcdist=(Batts[closest].zmove-Roboto[numrobot].zmove)*(Batts[closest].zmove-
Roboto[numrobot].zmove);
                                xcdist=(Batts[closest].xmove-Roboto[numrobot].xmove)*(Batts[closest].xmove-
Roboto[numrobot].xmove);
```

```
				if(zcdist+xcdist==0||zcdist+xcdist>((maxx+maxz)/3)*Roboto[numrobot].sight)
closest=-1;
				}
			else	{
				zdist=(Batts[i].zmove-Roboto[numrobot].zmove)*(Batts[i].zmove-Roboto
[numrobot].zmove);
				xdist=(Batts[i].xmove-Roboto[numrobot].xmove)*(Batts[i].xmove-Roboto
[numrobot].xmove);
				if((sqrt(zdist+xdist)<sqrt(zcdist+xcdist))&&(sqrt(zdist+xdist)!=1))
					{
					closest=i;
					zcdist=zdist;
					xcdist=xdist;
					}
				}
		}
if(closest!=-1)
		{
		if(Batts[closest].zmove-Roboto[numrobot].zmove<0)
			Roboto[numrobot].bodyspin1=((180*atan2((Batts[closest].xmove-Roboto[numrobot].
xmove),
				(Batts[closest].zmove-Roboto[numrobot].zmove)))/(M_PI));
		else	Roboto[numrobot].bodyspin1=((180*atan2((Batts[closest].xmove-Roboto[numrobot].
xmove),
				(Batts[closest].zmove-Roboto[numrobot].zmove)))/(M_PI));
		}
if(closest==-1)	{RANDOMAI(numrobot);return;}
		}
else			{RANDOMAI(numrobot);return;}

walk(1,Roboto[numrobot]);
}

void GROUPYAI(int numrobot)
{
Roboto[numrobot].mating++;
int closest=-1;
float zcdist,xcdist,zdist,xdist;
if(robotnum>1)
		{
for(int i=0;i<robotnum;i++)
		{
		if(i!=numrobot&&Roboto[numrobot].arefriend(Roboto[i])&&
		     fabs((Roboto[i].zmove-Roboto[numrobot].zmove)*(Roboto[i].zmove-Roboto[numrobot].
zmove))>Roboto[numrobot].armlength*2&&fabs((Roboto[i].xmove-Roboto[numrobot].xmove)*(Roboto
[i].xmove-Roboto[numrobot].xmove))>Roboto[numrobot].armlength*2)
			{
			if(closest==-1)
				{
				closest=i;
				zcdist=(Roboto[closest].zmove-Roboto[numrobot].zmove)*(Roboto[closest].
zmove-Roboto[numrobot].zmove);
				xcdist=(Roboto[closest].xmove-Roboto[numrobot].xmove)*(Roboto[closest].
xmove-Roboto[numrobot].xmove);
				if(zcdist+xcdist==0||zcdist+xcdist>((maxx+maxz)/3)*Roboto[numrobot].sight)
closest=-1;
				}
			else	{
```

```
                              zdist=(Roboto[i].zmove-Roboto[numrobot].zmove)*(Roboto[i].zmove-Roboto
[numrobot].zmove);
                              xdist=(Roboto[i].xmove-Roboto[numrobot].xmove)*(Roboto[i].xmove-Roboto
[numrobot].xmove);
                              if((sqrt(zdist+xdist)<sqrt(zcdist+xcdist))&&(sqrt(zdist+xdist)!=1))
                                        {
                                        closest=i;
                                        zcdist=zdist;
                                        xcdist=xdist;
                                        }
                              }
                    }
          }
if(closest!=-1)
          {
          if(Roboto[closest].zmove-Roboto[numrobot].zmove<0)
                    Roboto[numrobot].bodyspin1=((180*atan2((Roboto[closest].xmove-Roboto[numrobot].
xmove),
                              (Roboto[closest].zmove-Roboto[numrobot].zmove)))/(M_PI));
          else    Roboto[numrobot].bodyspin1=((180*atan2((Roboto[closest].xmove-Roboto[numrobot].
xmove),
                              (Roboto[closest].zmove-Roboto[numrobot].zmove)))/(M_PI));
          }
if(closest==-1){RANDOMAI(numrobot);return;}
          }

walk(1,Roboto[numrobot]);
}

void ADVANCEDAI(int numrobot, Robot Robots[][numcolonies],int colony)
{
if(Robots[numrobot][colony].hunger>2)BATTRYAI(numrobot,Robots,colony);
else GROUPYAI(numrobot,Robots,colony);
}
```

A.3 Code

```
class Genomebit
          {
          public:
                    int numpoints;
                    float domain[100];
                    int numbots[100];
                    int getdomain(int indicator,float number);
          };

int Genomebit::getdomain(int indicator,float number)
          {
          switch (indicator)
                              {
                              case 0:
                                        for(int i=0;i<this->numpoints;i++)
                                                  {
                                                  if (number==this->domain[i])
                                                            return(i);
                                                  }
                                        break;
```

```
                        case 1:
                                for(int i=0;i<this->numpoints;i++)
                                        {
                                        if (fabs(number-this->domain[i])<=(1.0/40.0))
                                                return(i);
                                        }
                                break;
                        case 2:
                                for(int i=0;i<this->numpoints;i++)
                                        {
                                        if (fabs(number-this->domain[i])<=(1.0/50.0))
                                                return(i);
                                        }
                                break;
                        case 3:
                                for(int i=0;i<this->numpoints;i++)
                                        {
                                        if (fabs(number-this->domain[i])<=(1.0/50.0))
                                                return(i);
                                        }
                                break;
                        }
        return(0);
        }

class Genome
        {
        public:
                Genomebit AI;
                Genomebit leglength;
                Genomebit armlength;
                Genomebit sight;
                void createGenome();
        };

void Genome::createGenome()
        {
        this->AI.numpoints=5;
        for(int i=0;i<this->AI.numpoints;i++)
                {
                this->AI.domain[i]=i+1;
                }
        this->leglength.numpoints=40;
        for(int i=0;i<this->leglength.numpoints;i++)
                {
                this->leglength.domain[i]=((float)i)/20+4.5;
                }
        this->armlength.numpoints=20;
        for(int i=0;i<this->armlength.numpoints;i++)
                this->armlength.domain[i]=((float)i)/25+1.2;
        this->sight.numpoints=20;
        for(int i=0;i<this->sight.numpoints;i++)
                this->sight.domain[i]=((float)i)/25+.3;
        }

Genome Thegenome;

void GeneticsGetData(int colony,Robot Robots[][numcolonies])
```

```
{
for(int i=0;i<100;i++)
        {
        Thegenome.AI.numbots[i]=0;
        Thegenome.leglength.numbots[i]=0;
        Thegenome.armlength.numbots[i]=0;
        Thegenome.sight.numbots[i]=0;
        }
for(int i=0;i<robotnum[colony];i++)
        {
        Thegenome.AI.numbots[Thegenome.AI.getdomain(0,Robots[i][colony].AITYPE)]++;
        Thegenome.leglength.numbots[Thegenome.leglength.getdomain(1,Robots[i][colony].leglength)]
++;
        Thegenome.armlength.numbots[Thegenome.armlength.getdomain(2,Robots[i][colony].armlength)]
++;
        Thegenome.sight.numbots[Thegenome.sight.getdomain(3,Robots[i][colony].sight)]++;
        }
}


void Genetics (Robot M, Robot D,int colony)
{
float sight,leglength,armlength;
//determine sight gene
int largest=Thegenome.sight.getdomain(3,M.sight);
if(Thegenome.sight.getdomain(3,M.sight)<Thegenome.sight.getdomain(3,D.sight))
        for(int i=Thegenome.sight.getdomain(3,M.sight);i<=Thegenome.sight.getdomain(3,D.sight);i++)
                if(Thegenome.sight.numbots[i]>=Thegenome.sight.numbots[largest])
                        largest=i;
if(Thegenome.sight.getdomain(3,M.sight)>=Thegenome.sight.getdomain(3,D.sight))
        for(int i=Thegenome.sight.getdomain(3,M.sight);i<=Thegenome.sight.getdomain(3,D.sight);i++)
                if(Thegenome.sight.numbots[i]>=Thegenome.sight.numbots[largest])
                        largest=i;
sight=Thegenome.sight.domain[largest]+((float)((rand()%100-50)/10000));
//determine leglength gene
largest=Thegenome.leglength.getdomain(1,M.leglength);
if(Thegenome.leglength.getdomain(1,M.leglength)<Thegenome.leglength.getdomain(1,D.leglength))
        for(int i=Thegenome.leglength.getdomain(1,M.leglength);i<=Thegenome.leglength.getdomain
(1,D.leglength);i++)
                if(Thegenome.leglength.numbots[i]>=Thegenome.leglength.numbots[largest])
                        largest=i;
if(Thegenome.leglength.getdomain(1,M.leglength)>=Thegenome.leglength.getdomain(1,D.leglength))
        for(int i=Thegenome.leglength.getdomain(1,M.leglength);i<=Thegenome.leglength.getdomain
(1,D.leglength);i++)
                if(Thegenome.leglength.numbots[i]>=Thegenome.leglength.numbots[largest])
                        largest=i;
leglength=Thegenome.leglength.domain[largest]+((float)((rand()%100-50)/10000));
//determine armlength gene
largest=Thegenome.armlength.getdomain(2,M.armlength);
if(Thegenome.armlength.getdomain(2,M.armlength)<Thegenome.armlength.getdomain(2,D.armlength))
        for(int i=Thegenome.armlength.getdomain(2,M.armlength);i<=Thegenome.armlength.getdomain
(2,D.armlength);i++)
                if(Thegenome.armlength.numbots[i]>=Thegenome.armlength.numbots[largest])
                        largest=i;
if(Thegenome.armlength.getdomain(2,M.armlength)>=Thegenome.armlength.getdomain(2,D.armlength))
        for(int i=Thegenome.armlength.getdomain(2,M.armlength);i<=Thegenome.armlength.getdomain
(2,D.armlength);i++)
                if(Thegenome.armlength.numbots[i]>=Thegenome.armlength.numbots[largest])
```

```
                                largest=i;
armlength=Thegenome.armlength.domain[largest]+((float)((rand()%100-50)/10000));

//mutations
//mutate sight
float sightcc=0,legcc=0,armcc=0;
int possibility=1;
if((M.RED==D.RED)&&(M.GREEN==D.GREEN)&&(M.BLUE==D.BLUE))
          possibility+=10;
if(Thegenome.sight.getdomain(3,M.sight)==Thegenome.sight.getdomain(3,D.sight))
          possibility+=5;
if(rand()%100<possibility)
          {
          sight=((float)(rand()%75)/100)+.4;
          sightcc=.1;
          }
//mutate leglength
possibility=1;
if((M.RED==D.RED)&&(M.GREEN==D.GREEN)&&(M.BLUE==D.BLUE))
          possibility+=10;
if(Thegenome.leglength.getdomain(1,M.leglength)==Thegenome.leglength.getdomain(1,D.leglength))
          possibility+=5;
if(rand()%100<possibility)
          {
          leglength=((float)(rand()%200)/100)+4.5;
          legcc=.1;
          }
//mutate armlength
possibility=1;
if((M.RED==D.RED)&&(M.GREEN==D.GREEN)&&(M.BLUE==D.BLUE))
          possibility+=10;
if(Thegenome.armlength.getdomain(2,M.armlength)==Thegenome.armlength.getdomain(2,D.armlength))
          possibility+=5;
if(rand()%100<possibility)
          {
          armlength=((float)(rand()%75)/100)+1.2;
          armcc=.1;
          }

//creates new robot;
Roboto[robotnum[colony]][colony].create(sight,
                                        leglength,
                                        armlength,
                                        M.AITYPE,
                                        M.zmove,
                                        M.xmove,
                                        (M.RED+D.RED)/2+sightcc,
                                        (M.GREEN+D.GREEN)/2+legcc,
                                        (M.BLUE+D.BLUE)/2+armcc,
                                        1.01*fabs(M.RED-D.RED)+sightcc,
                                        1.01*fabs(M.GREEN-D.GREEN)+legcc,
                                        1.01*fabs(M.BLUE-D.BLUE)+armcc,
                                        M.dettractionR+D.dettractionR/2,
                                        M.dettractionG+D.dettractionG/2,
                                        M.dettractionB+D.dettractionB/2);
}
```