游戏编程中的寻路算法研究

付朝晖[1]，丁　梦[2]，喻　昕[1]

（1.长沙民政职业技术学院，湖南 长沙 410004；2.湖南工业大学，湖南 株洲 412008）

摘　要：探讨了游戏开发中的一些寻路算法，提出了如何在游戏中使用遗传算法实现路径探索的基本思路，分析了遗传算法实现寻路的优点及其存在的问题。

关键词：电子游戏；人工智能；遗传算法

中图分类号：TP311　　**文献标识码**：A　　**文章编号**：1673-9833(2007)04-0084-04

Algorithm of Seeking the Path in Game Programming

Fu Zhaohui[1]，Ding Meng[2]，Yu Xin[1]

（1.Changsha Social Work College，Changsha 410004，China；
2.Hunan University of Technology，Zhuzhou Hunan 412008，China）

Abstract：A few algorithm of seeking the path have been discussed in game developing，and it proposes how to realize the fundamental thought of the exploration by using genetic algorithm in game programming ,then it analyzes how to realize the merit and finds out the existing problem in genetic algorithm.

Key words：lectronic games；artificial intelligence；genetic algorithm

0　引言

近年来，游戏产业的快速发展带动了游戏中人工智能（Artificial Intelligence，简称AI）的发展，越来越多的游戏采用人工智能技术提高游戏的可玩性。在电子游戏中，玩家操控主要角色，而其他角色的行为逻辑由人工智能操纵，这些角色我们称之为NPC（Non-Player Character，非玩家控制角色）。大部分游戏在开发过程中都会遇到路径探索问题，快速、准确地计算出游戏角色从地图中的A点到达B点的一条路径，一直是游戏开发者追求的目标，同时也是游戏人工智能研究的一个重要方面。

1　游戏编程中的简单寻路算法

在游戏关卡中常常会放置一些怪物（即NPC），这些怪物通常在一个区域内走来走去，这个区域被称为“巡逻区域”；一旦玩家的角色进入怪物的“视野”，怪物就会发现玩家角色，并主动向其所在的位置移动，这个区域称为“警戒区域”；当玩家角色和怪物更加靠近时，会进入到怪物的“攻击区域”，这时怪物会对玩家角色进行伤害。在某些RPG（Real-Time Strategy Game，即时战略游戏）中，NPC在不利的情况下还会选择主动逃跑。如何模拟这些行为逻辑，目前游戏业已经有一些比较成熟的方法。

1.1　随机寻路算法

随机寻路算法适合模拟游戏中那些没有什么头脑的生物，它们总是在场景中漫无目的地走来走去。可以用以下的代码进行模拟：

```
npc_x_velocity = -5 + rand( ) %10；
npc_y_velocity = -5 + rand( ) %10；
int npc_move_count=0；
while(++npc_move_count<num){
```

收稿日期：2007-06-21

作者简介：付朝晖（1972-），男，湖南常德人，长沙民政职业技术学院讲师，硕士生，主要研究方向为计算机技术；
丁　梦（1981-），女，湖南湘潭人，湖南工业大学助教，硕士生，主要研究方向为软件技术.

```
    npc_x+ = npc_x_velocity；
    npc_y+ = npc_y_velocity;
  } //end while
```

在上例中，NPC 会选取一个随机方向和速率运动一会儿，然后再选取另一个。当然，还可以加上更多的随机性，如，改变运动方向的时间不是固定的 num 个周期，或者更倾向于朝某个方向等。实际编程中还必须考虑到碰撞检测，当 NPC 遇到障碍物后，会随机选取一个前进的方向，继续行走。

1.2　跟踪算法

当游戏中的主角进入到 NPC 的“警戒区域”后，游戏的 AI 可轻易获得目标的位置，然后控制 NPC 对象移向被跟踪的对象。跟踪算法可以模拟这一行为：

```
void Bat_AI(void)
{
if (ghost.x > bat.x)
  bat.x++;
else
if (ghost.x < bat.x)
  bat.x--;

if (ghost.y > bat.y)
  bat.y++;
else
if (ghost.y < bat.y)
  bat.y--;
……
} // end Bat_AI
```

这段代码放到程序中实际运行时不难发现，NPC 会迅速地追踪到目标。这种跟踪非常精确，但是在游戏中过于精确却不一定是一件好事，因为这会使 NPC 的行为看上去显得有点假。一种更自然的跟踪方式是使跟踪者的方向矢量与从跟踪目标的中心到跟踪者的中心所定义的方向矢量靠拢。

这个算法可以这样设计：假设 AI 控制的对象称作跟踪者（tracker）并有以下属性：

Position: (tracker.*x*，tracker.*y*)

Velocity: (tracker.*xv*，tracker.*yv*)

被跟踪对象称作跟踪目标（target），有如下属性：

Position: (target.*x*，target.*y*)

Velocity: (target.*xv*，target.*yv*)

基于上面的定义，下面是调整跟踪者的速度向量的常用逻辑循环：

1）计算从跟踪者到跟踪目标的向量：

TV = (target.*x* -tracker.*x*，target.*y*-tracker.*y*) = (tv*x*，tv*y*)，规格化 TV——也就是说 (tv*x*，tv*y*)/ Vector_Length (tv*x*，tv*y*) 使得最大长度为 1.0，记其为 TV*。记住 Vector_Length( ) 只是计算从原点(0，0)开始的矢量长度。

2）调整跟踪者当前的速度向量，加上一个按 rate 比例缩放过的 TV*：

```
tracker.x += rate*tvx;
tracker.y += rate*tvy;
```

注意：当 *rate* >1.0 时，跟踪向量会合得更快，跟踪算法对目标跟踪得更紧密，并更快地修正目标的运动。

3）跟踪者的速度向量修改过之后，有可能向量的速度会溢出最大值，就是说，跟踪者一旦锁定了目标的方向，就会继续沿着该方向加速。所以，需要设置一个上界，让跟踪者的速度从某处慢下来。可做如下改进：

```
tspeed = Vector_Length(tracker.xv，tracker.yv);
if(tspeed>max_SPEED){
tracker.xv*=0.7;
tracker.yv*=0.7;
}
```

也可以选择其它的边界值 0.5 或 0.9 等均可。如果追求完美，甚至可以计算出确切的溢出，并从向量中缩去相应的数量。追踪过程中同样也会遇到障碍物，因此，碰撞检测是必不可少的。程序员可以根据不同的游戏类型设计碰撞后的行为逻辑。

1.3　闪避算法

这个技术是让游戏的 NPC 能避开玩家角色的追击，跟前面的跟踪代码很相似，跟踪算法的对立面就是闪避算法，只要把上例中的等式翻转，闪避算法就成了，下面是转换后的代码：

```
if (ghost.x > bat.x)
  bat.x--;
else
if (ghost.x < bat.x)
  bat.x++;
if (ghost.y > bat.y)
  bat.y--;
else
if (ghost.y < bat.y)
  bat.y++;
……
```

以上介绍的 3 个算法可以模拟 NPC 的一些简单的寻路、跟踪和闪避行为，在小游戏中会经常用到。但是，在较大型的游戏中使用这样简单的算法就会大大影响游戏效果了。因此，大型游戏的人工智能算法都较复杂。

2　遗传算法在路径探索中的应用

改进优化后的 A* 算法可以很好地胜任游戏中的路径搜索[1]，一直以来被游戏界认为是最好、最成熟的寻路算法之一，因而被广泛应用。由于 A* 算法是按照寻找最低耗费的路径来设计，A* 寻路会找到最短，最直接

的路径，当算法具体实现后，得到的这条路径也是唯一的路径[2]。于是，当游戏重来时，玩家会发现NPC总是只有一条路可走，这样就显得不够真实。玩家希望NPC有足够的智力能找到一条适合的路径，也要有不同的选择。如果能够为游戏中的角色设置一个智能系统来进行控制，这个系统能通过自身不断地学习，逐渐适应复杂的环境，自己找到一条“较好”的路径，并且有较高的效率，就会使游戏角色的行为逻辑看上去更真实一些。这样就需要借助人工智能的一些技术，如遗传算法等。

2.1 遗传算法

遗传算法是模拟生物进化的步骤，将繁殖、杂交、变异、竞争和选择等概念引入到算法中，通过维持一组可行解，并通过对可行解的重新组合，改进可行解在多维空间内的移动轨迹或趋向，最终走向最优解。它克服了传统优化方法容易陷入局部极值的缺点，是一种全局优化算法[3，4]。

遗传算法的步骤如下：

1）对待解决问题进行编码——生物的性状是由生物遗传基因的编码所决定的，使用遗传算法时，需要把问题的每一解编码成一个基因编码，一个基因编码就代表问题的一个解，每个基因编码有时被称作是一个个体，有时也把基因编码称作染色体[5]；

2）随机初始化群体$X(0)=(x1，x2，\cdots，xn)$；

3）对当前群体$X(t)$中每个个体xi计算其适应度$F(xi)$，适应度表示了该个体的性能好坏；

4）从当前群体选出2个成员，选出的概率正比于染色体的适应性，适应分愈高，被选中概率也愈大；

5）按照预先设定的杂交率，从每个选中染色体的一个随机确定的点上进行杂交；

6）按照预定的变异率，通过对被选染色体的位的循环，把相应的位实行翻转；

7）如果不满足终止条件继续3）。

2.2 路径探索的遗传算法实现

在这个路径探索例子中，首先创建1个地图，它有1个入口，1个出口。地图中放置一些障碍物，在入口处放置1个NPC。地图可以用1个二维整数数组Map[ ][ ]来表示，其中用0来表示可以通行的空间，1代表墙壁或障碍物，8为入口，9为出口。游戏开发者通常是借助地图编辑器之类的工具来生成这个数组。

接下来要使它能找到出口，并避免与所有障碍物相碰撞。这种地图设计方法被封装在一个称为CNpcMap的类中，只需要以常量的形式来保存地图数组以及起点和终点就行了。除了存储地图，这个CNpcMap类中需要1个数组NpcPath[ ][ ]，用来记录NPC在地图中行走的路径。可以用由1（UP）、2（DOWN）、3（LEFT）、4（RIGHT）所组成的动作方向序列来检测NPC走了多远，并计算出NPC能到达的最远位置，然后返回1个适应性分数，它正比于NPC最终位置离出口的距离。NPC所到达的位置与出口越近，给NPC的适应性分数就越高。如果NPC实际已到达了出口，将得到满分1，这时，循环就会自动结束，此时得到问题的一个解。

根据遗传算法的步骤，第1步是为染色体编码，染色体把NPC的每一个动作方向编入代码中。NPC有上、下、左、右4个动作方向，编码后的染色体是代表这4个动作方向的一个数组DirAction[ ]。首先通过程序生成由1234组成的整型随机数数组，就能根据它得到NPC行动时的方向。例如染色体{3，2，1，2，4，3，2，1，…}。

第2步要做的是将NPC置于地图的入口，然后指示NPC根据DirAction[ ]数组中所列的方向指令序列一步步地走。如果有一个方向使NPC碰到了墙壁或障碍物，则忽略该指令序列并继续按下一条指令序列去走就行了。这样不断走下去，直到用完所有方向或NPC到达出口为止。遗传算法以整型随机数数组作为初始群体，一般要求产生几百个这样的随机染色体，测试它们每一个能让NPC走到离出口有多么近，然后让其中最好的那些作为种子产生后代，期望它们的“子孙”中能有走得离出口更近一点。这样继续下去，直到找出一个解。因此，需要定义一种结构，其中包含一个染色体，以及一个与该染色体相联系的适应性分数。这个结构的定义如下：

```
struct Chromosome
{
  int  DirAction[ ];
  double   FitnessScore;
  Chromosome ( )：FitnessScore(0){}
  Chromosome( int num)：FitnessScore(0)
  {
       for (int i=0；i<num；++i)
    {
     // 创造随机字符数组
    }
  }
}
```

在创建Chromosome对象时，把一个整型数作为参数传递给构造函数，则它就会自动创建一个以此整数为长度的随机数字数组，并将其适应性分数初始化为零，完成对基因组的设置。

第3步，也是遗传算法类中最为关键的一步，就是测试染色体群中每一个体的适应性分数，数。函数CalFitSco( )根据代表上、下、左、右4个方向的数组DirAction[ ]，计算出NPC离开出口的最终距离，返回一个适应性分数。计算适应性分数程序如下：

```
int Dist_X = abs(Npc_X - End_X)；
int Dist_Y = abs(Npc_Y - End_Y)；
```

```
return 1/(Dist_X+Dist_Y+1)；//加1避免分子为0
```

这里的Dist_X和Dist_Y就是NPC所在的位置相对于地图出口的水平和垂直偏离值。如果NPC到达出口，则Dist_X + Dist_Y = 0。CalFitSco( )保持对每一代中适应性分数最高的基因组以及与所有基因组相关的适应性分数的跟踪。每当一个新的基因组群被创建出来时，需要将它们保存下来。

第4步，从当前群体选出2个个体以备杂交。赌轮选择是常用的一种方法，被选中的几率和它们的适应性分数成比例，适应性分数愈高的染色体，被选中的概率也愈大。但这不是说适应性分数最高的成员一定能选入下一代，它只是有最大的概率被选中。

```
while( Parents< V_Num)
 {
 //用赌轮法选择
 Chromosome Parent1 = RSele( )；
 Chromosome Parent2 = RSele( )；
 ……
 }
```

在每次迭代过程中，需选择2个染色体作为“后代”染色体的“父辈”，一个染色体的适应性分数越高，其被赌轮方法选择作为“父辈”的概率就越大。

第5步杂交操作

```
Chromosome Cbaby1，Cbaby2；
Hybridize (Parent1.DirAction，
Parent2.DirAction ， Cbaby1.DirAction ，
Cbaby2.DirAction)；
```

创建2个新的染色体，它们与所选的父辈一起传递给杂交函数Hybridize( )。这一函数执行了杂交，并把新的染色体的二进制位串存放到Cbaby1和Cbaby2中。

第6步变异操作

```
Differentiation(Cbaby1.DirAction)；
Differentiation(Cbaby2.DirAction)；
```

以上这2步实现对染色体后代杂交变异，完成后把2个后代染色体加入新的群体，这样就完成了一次迭代过程。该过程不断重复，原有的群体由新生一代所组成的群体代替，直到染色体收敛到了一个解。

程序运行中有可能不是总能找到一条通往出口的路径，这时有可能会导致NPC会在一个角落不确定地走来走去，原因是群体太快地收敛到一个特殊类型的染色体，由于群体中的成员变得相似，Hybridize算子的优势这时实际上已经不能发挥作用，只有很少的变异操作在起作用。但变异率设置得很低，当染色体类型的差异消失后，仅仅依靠变异本身已不能去发现一个解[6]。选择适当的参数可以改善这个问题，但是目前还没有一个有效的规则，不同的问题需要不同的值，用户只能通过自己的实践选取合适的值，在具体实现中把杂交率选为0.6，变异率取为0.05，而基因组数目取为染色体长度的2倍，可得到较理想的解。

2.3　遗传算法寻路的优缺点

与其他寻路算法相比，遗传算法具有如下优点：

1）在搜索中用到的是随机的变换规则，而不是确定的规则。它在搜索时采用启发式的搜索，而不是盲目的穷举，因而具有很高的搜索效率。

2）遗传算法给出的是一组优化解，而不是一个优化解，这样，在游戏中可表现出更多的行为空间。

3）遗传算法具有很强的可并行性，可通过并行计算来提高计算速度，因而更适用于大规模复杂问题的优化。

但是，遗传算法毕竟是一种较新的算法，实际运用中存在一些问题，主要集中在以下几个方面：

1）遗传算法的理论研究较滞后。由于遗传算法本身是一种仿生的思想，尽管实践效果较好，但理论证明比较困难，而且，这种算法提出来的时间还不是太长，因此，其理论和实践的研究几乎是平行进行的。

2）算法本身的参数还缺乏定量的标准，目前采用的都是经验数值，而且，不同编码、不同遗传技术都会影响到遗传参数的选取，因而会影响到算法的通用性。

3）遗传寻路算法还算不上真正的实时算法，在游戏中的应用会受到一些限制，在实时性很强的游戏中不宜使用。

3　结语

近年来，游戏的AI发展很快，各种AI技术被引入到游戏中，如遗传算法、人工神经网络计算、地形分析技术、团队寻径算法、A*算法等，这些技术出色地解决了游戏中一些基本问题。有理由相信，未来游戏的AI将会给玩家带来更多的惊喜。

参考文献：

[1] 何国辉,陈家琪.游戏开发中智能路径搜索的算法研究[J].计算机工程与设计，2006，27（13）：2334-2337.

[2] 陈和平,张前哨.A*算法在游戏地图寻径中的应用与实现[J].计算机应用与软件，2005，22（10）：118-120.

[3] 王小平，曹立明.遗传算法－理论、应用与软件实现[M].西安：西安交通大学出版社，2002.

[4] 阎平凡，张长水.人工神经网络与模拟进化计算[M].北京：清华大学出版社，2000.

[5] 王淑琴.神经网络和遗传算法在游戏设计中的应用研究[D].长春：东北师范大学，2004.

[6] 金朝红，吴汉松，李腊梅，等.一种基于自适应遗传算法的神经网络学习算法[J].微型机信息，2005，21(10-11)：49-51.