学校代号：10532

学　　号：G08240052

密　　级：

湖南大学工程硕士学位论文

基于神经网络和遗传算法的人工智能游戏研究与应用

学位申请人姓名：余颖

导师姓名及职称：林亚平教授 沈震冰高级工程师

培 养 单 位：软件学院

专 业 名 称：软件工程

论文提交日期：2011年4月27日

论文答辩日期：2011年5月17日

答辩委员会主席：王如龙 教授

Based on Neural Network and Genetic Algorithm Research and Application of Artificial Intelligence Game

by

YU Ying

B.E. (Changsha RailWay University) 1997

A thesis submitted in partial satisfaction of the

Requirements for the degree of

Master of Engineering

in

Software Engineering

in the

Graduate School

of

Hunan University

Supervisor

Professor LIN Yaping

Sentior Engineer SHEN Zhenbing

April, 2011

湖 南 大 学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：余毅　　日期：2011年 5月 26日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖南大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密□。

（请在以上相应方框内打“√”）

作者签名：余毅　　日期：2011年 5月 26日

导师签名：[illegible] 沈震冰　　日期：2011年 5月 26日

摘 要

人工智能是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术以及应用系统的一门新的技术科学。人工智能并非为游戏制作而产生，但是随着越来越多的游戏进入市场，游戏中引入人工智能逐渐成为了一款游戏吸引玩家的有效手段之一。人工智能在游戏中能增加玩家的挑战性、创造更真实的虚拟世界、增加游戏的可玩性、辅助其他功能。

“学习”显然属于一种非定性 AI 的范围，难度有些大，这也是目前各大游戏公司努力的方向之一。几个主流的游戏都用到了非定性 AI 方法，如“Creatures”，“Black &White”，这些游戏的成功，成功激起了人们对“学习”AI 技术的兴趣，诸如决策树（decision tree）、神经网络、遗传算法(GA)以及概率方法。

本文通过描述了遗传算法与人工神经网络的概念和现状，对遗传算法与人工神经网络在游戏中的应用进行了说明，基于遗传算法和人工神经网络，结合自主性 NPC 的设计，构建了新算法 NPG 算法。

论文通过设计自动玩俄罗斯方块的程序，引入了局面判断和最优决策的评判方法。该方法枚举出当前方块以及即将产生的方块在下落时期可能出现的各种变形以及可能产生的各种局面。该方法才有了三大最权威的估值函数，从攻击能力和防守能力两个方面对方块的高度、可能消除的行数以及可能形成的 holes 数目进行综合评估，得到一个加权平均值，依据该权值挑选出最大值作为最优决策的决策参数。决策过程贯穿游戏始终。实验表明，该方法有效的平衡了攻击性和防守性。

关键词：人工智能；遗传算法；人工神经网络；决策规划；最优评判

Abstract

Artificial Intelligence is the research and development for simulation, extension and expansion of human intelligence, which consists of theory, methods, techniques and applications to new technological sciences. AI is not supposed to develop games at early stage. However, facing the pressure and competition that increasing games spring up in market, introducing artificial intelligence into the game has gradually become an effective and essential means to attract players, for it improves the plays' sense of achievement, enables the virtual world to appear more realistic and assists other features.

"Learning", with some difficulty to perform, obviously belongs to the scope of a non-qualitative AI. It is also the goal that major game companies attempt to achieve. To be specific, the methods of non-qualitative AL have been adopted by several mainstream games such as "Creatures" and "Black & White". Undoubtedly, the success of these games extremely stirs people's interest of "learning", which can be exemplified by the numerous studies on the decision tree, neural networks, genetic algorithms (GA) and probabilistic methods.

This paper describes the genetic algorithm and artificial neural network concepts and status, genetic algorithms and artificial neural network application in the game are described, based on genetic algorithms and artificial neural networks, independent of the NPC with the design, construction of the new algorithm NPG algorithm.

The paper brings in the situation and determines the optimal decision of the evaluation method by designing process which can play Tetris automatically. The method we employ has three most authoritative valuation functions at present. First, we use the method to enumerates the overall possible variant situations of current and upcoming boxes in their falling period as well as all probable results they generates. Moreover, we count the box height, the number of rows which may be eliminated and the possible number of holes in terms of the attacking and defending capabilities thereby making a comprehensive assessment .Eventually, we obtain a weighted average to select the maximum weight as the decisive parameters of optimal strategy. What is worth noticing is that the decision-making process will go throughout the game. By experiments we find that the approach is very effective and efficient.

Keywords: Artificial intelligence; genetic algorithms; artificial neural networks; decision-making plan; the best judge

目 录

插图索引

附表索引

第1章 绪 论

1.1 本文研究背景及意义

自计算机出现以来，几十年的时间，计算机学科发展出了数十门新的学科，其中，计算机图形学和硬件技术无疑是其中的佼佼者，两者的发展带动了整个产业技术不断更新，更造就了一大黄金产业，计算机游戏。

十几年来，游戏软件成为软件产业中最重要的内容之一，特别是网络游戏被开发出来后，甚至改变了人类的生活方式。PC 游戏出现至今，从单纯文字类网络游戏到 3D 大型场景类网络游戏，PC 游戏的类型已经没有早年那么容易定义了，PC 游戏的形式也是趋于多样化。但总体说来，游戏的类型分为：RPG(角色扮演类型)、FPS(第一人称射击)、RTS(即时战略游戏)等。

无论是大型网游还是网页游戏，异或是棋牌类游戏，这些游戏都有很重要的一个共同特征，玩家控制的角色(PLAYER)和游戏控制的角色(NPC)必须有行为交互。交互设计得是否合理，将直接影响玩家对游戏的兴趣度，进而影响到整个游戏的生存。

因此，对 PLAYER 和 NPC 之间的角色行为交互，已成为游戏软件业一个必须要面对的问题，也成为摆在众多学科前沿的研究者面前非常重要的研究课题之一。

目前市面上大部分游戏中，角色行为的交互方法均采用一种预设模型，其特征为，角色的行为都是预先确定的，比如腾讯的棋牌类游戏，在玩家赢得比赛后，系统会给予一定的奖励或消息激励。但是，这种奖励或消息均是系统预定义好的，没有将玩家自身的特色或游戏当前的情景结合起来，难免让人觉得千篇一律，重复几次，甚至让人觉得厌烦。

这种确定型的交互行为，实现起来非常简单，只要用枚举型变量将这些交互变量定义好就可以。但是，正如之前所说，这种确定型的交互完全无法体现不同角色之间的自主性，角色的行动方式单调乏味，玩家只要玩过几局，往往非常容易预测出角色的行为，使得游戏的可玩度大大降低。

因此，游戏设计者希望能够在游戏软件中，设计出一种智能型的 NPC 或自动型的 NPC，这些角色能够根据以往的经验和知识，结合当前环境变化，动态影响 PLAYER，使得 PLAYER 无时不刻都能体会到新鲜感。

这种自主性和自适应性的 NPC 角色，具备很强的自适应能力和自我推理能力，应用到游戏中，往往能使玩家更受欢迎。一款游戏如果拥有这种智能型 NPC，往往能够极大吸引玩家注意力，倾注玩家的游戏热情，从而延长游戏的生存周期，

使游戏厂商获得更多的回报和关注度。

因此，目前较为成功的游戏厂商，纷纷将注意力集中在游戏中智能型 NPC 的实现。

人工智能的出现，让这种智能型 NPC 的设计成为可能，合理有效的采用人工智能手段，实现人机交互、人机对战，成为研究的最前沿。

一些国外的游戏公司，首先尝试着从人工智能领域发展和提炼出更加适合游戏开发的高级技术，应用得较多的如：利用强化学习或者采用决策树的方法来实现自主性 NPC。而一度风靡世界的游戏 Colin McRae Rally2 则采用了神经网络和学习系统相结合实现角色的自主性。

一些专家和学者已经在自主性角色行为方面做出了不少开创性的贡献。如 Galyean、Blumberg[1]等人在模型中引入更多的行为控制机制，并将行为学习这种机器学习方面的问题考虑了进去。Reynolds[2]则对自主性角色的群体行为进行了仔细分析。John David Fungc 在文献[3]中指出，对于自主角色的认知能力及更高层次上，虚拟世界模型金字塔顶层是用认知模型进行创建，底层模型则由行为模型进行创建，认知模型和物理模型之间创建一个缓冲区，用行为模型进行填充。而在动态虚拟世界中，情景演算(situation calculus)也被首次考虑进去。

众所周知 Tetris 是永恒的娱乐经典，但它实际上又和那些传统的经典娱乐方式不同，因为它的本质是电子化的。比如，Tetris 与象棋完全不一样，玩家无法以任何实体形式去玩俄罗斯方块。自其诞生以来，Tetris 以无数形式经过了大量变形和更改，但游戏核心依然保持不变。这些游戏的核心能够提供吸引人亲自动手实践的挑战[4]。

这个具有定义性和广泛影响的益智游戏激发了整整一种风格的作品，特别是后来数年间发源于日本的某些优秀的益智游戏，它们全都在 Tetris 的基础上作了自己独特的改动。然而，虽然这些游戏看上去各有千秋，但它们与 Tetris 的内在相似性却无可否认。

棋牌类商业 AI 已经形成了一定的市场，但以俄罗斯方块(Tetris)为主的经典小游戏至今仍然没有能够投入商业化的优秀 AI 程序。

休闲类游戏的 AI 算法，最广为人知的便是博弈树算法（极大极小搜索树）。我们知道，深蓝曾经战胜过世界冠军，然而深蓝的下一代小深（比深蓝更厉害，博弈树层数更多）却只和世界冠军战成平手，这不禁让人怀疑，是否博弈树出了差错。

但是，博弈树经过多年，多人的论证，几乎成为公认的棋牌类休闲游戏搜索树算法。经过仔细论证最后发现，问题出在评估函数(evaluation function)[5]身上。

评估函数严格意义上来说还算不上人工智能，因为其不具备普适性，一个游戏的评估函数往往不能被其他游戏采用[6-7]，但这里不是我们讨论的重点。

局面评估函数，不同的游戏不同的算法。在计算机程序中，进行局面评估函数算法设计时，可以输入一对特定的棋型来求评估值，将计算出的评估值与预想进行比较。然后，分析结果并改良算法。

无论是局面评估还是神经网络、异或是机器学习，各种方法侧重点不同，各有优缺点，而且，这些方法相互之间没有必然的联系。本文在考量这些方法的基础上，对认知角色建模，采用神经网络和遗传算法相结合的方法，对自主角色进行设计，并以自动俄罗斯方块游戏为载体，验证新算法的可行性。

1.2 国内外研究现状

游戏中最广泛使用的 AI 技术就是作弊[8]。例如，在战争模拟游戏中，由计算机控制的玩家可以得知对手（人类）的所有信息（基地位置，军队种类、数量和所在地等），不需要像人类玩家那样派出侦察兵去收集这类情报。这种作弊手法很常见，让计算机可以比人类玩家取得某种优势，这样使得机器能够更胜人类玩家一筹，激发起玩家征服的兴趣。当然，这种作弊需要一定的技巧，既不能让计算机表现得无懈可击，这样无疑会打击人类玩家的积极性，也不能表现得过于简单，这会让玩家失去继续下去的兴趣。作弊必须采取中庸之道。

作弊不是唯一常用的现有 AI 技术，有限状态机（finite state machine, FSM）也是一种随处可见的游戏 AI 技术[9]。其基本概念是列举出计算机控制的角色，一连串动作或状态，再利用 if-then 条件语句检查各类情况和满足条件，据此执行动作或状态，或者在动作或状态之间做转换。

目前最常用的是模糊状态机（fuzzy state machine）中用到模糊逻辑[10]，让最后执行的动作难以预测，减少必须以 if-then 语句大量列举条件的重担。

其他还有诸如以规则为主的描述式系统，以及某些人工生命技术等等，这些技术的灵活运用，特别是在一些视频类游戏中，获得了极大的成功[11,12]。

游戏 AI 的下一件大事就是“学习”了，游戏必须能够更多地演化和学习，更具适应性，这样游戏会跟玩家一起成长，玩家也难以预测游戏的行为，因此就能拓展游戏的生命周期。

“学习”显然属于一种非定性 AI 的范围，难度有些大，这也是目前各大游戏公司努力的方向之一。几个主流的游戏都用到了非定性 AI 方法，如“Creatures”，“Black &White”，“Battlecruiser 3000AD”，“Dirt Track Racing”，“Fields of Battle”以及“Heavy Gear”，这些游戏的成功，成功激起了人们对“学习”AI 技术的兴趣，诸如决策树（decision tree）、神经网络、遗传算法(GA)以及概率方法(probabilistic method)[13,14]。

遗传算法(Genetic Algorithm,GA) 是基于达尔文生物进化论的自然选择学说和群体遗传学原理而建立的一种随机全局优化算法[15,16]。GA 属于自适应的全局

优化概率搜索算法。GA 由三大算子组成，分别是：选择、交叉和变异。GA 具备一定的智能性，在搜索过程中，算法能够自动获取、积累相关知识，而且其搜索过程为自适应控制过程，搜索完成后，根据相应计算，能达到最优解。

遗传算法对各种复杂结构进行简单编码，通过编码，对简单的遗传操作及优胜劣汰的自然学则进行学习指导，调整搜索方向。搜索过程中，种群中的个体逐步优化并逼近全局最优解。

透过遗传算法的工作机制，很容易看出，该算法具备很好的普适性和可规划性，而且，形式上也非常简明，能够迅速的与其他算法相结合。

Peng Huo,Simon C.K.Shiu 等人在文献[17]中，将遗传算法应用到策略对抗性游戏中，并提出了一种新的变异算子。Jong Yih Kuo 和 Yuan Cheng Ou 在文献[18]中，将遗传算法应用到桌面足球游戏中，并在交叉部分引入了模糊理论。T.E.Revello 和 R.McCartney[19]成功将遗传算法应用到战争策略型游戏中，并取得了良好的效果，地图刷新率提升了 20%。Anurag Bhatt、Pratul Varshney、Kalyanmoy Deb 等人在文献[20]中，成功将遗传算法应用到零和博弈游戏中，创造了游戏连续运行 100 个小时无平局的成果。

人工神经网络(BP)通过对人脑神经元结构进行模拟，通过一系列复杂的算法对数据进行处理，从而达到预测、判别模式、联想记忆等功能。网络分布式矩阵结构的连接权值上携带有大量的知识，而学习过程则通过调整计算权值来实现。游戏角色的建模往往需要建立高维矩阵，传统方法如回归分析、多元统计分析等方法，在处理高维数据方面往往表现得不如人意，于是，引入神经网络便成为不少专家的选择之一。

Hinton、 Scjnowski 等人在文献中提出一种 Boltzman 机模型，该模型引入了统计物理学的相关知识，首次提出了多层次网络，并将模拟退火技术引入到学习过程[21]。

Rumelhart 和 McCelland 等人将粒子群算法与人工神经网络算法结合起来，提出一种新算法，PSO-BP 算法，将之运用到学习的训练过程，并成功运用到一款 RPG 游戏中[22]。

Kaixiang Peng,Xuli Zhang 等人[23]成功将人工神经网络算法运用到地图的边缘检测之中，并将之应用于战争策略游戏中，对生成新地图提供了一种新的机制。Hongmei Shao,Gaofeng Zhen 等人在三大神经元激励函数 linear、sigmoid、tansig 基础之上，提出了一种新的激励函数，并将之运用到游戏的回路搜索中，带来了良好的激励方式，增加了游戏的可玩性。

针对俄罗斯方块这款游戏中的 AI，国外也有不少研究，像一些爱好者建立的网站 www.tetris.com 等等，Colin Fahey 也建立了专门的 Tetris 项目，在他的个人主页 http://www.colinfahey.com/tetris/tetris_en.html 上有相关的研究成果[24]。

目前流行于 Tetris 界的典型性评估函数有 Pierre Dellacherie 算法、Roger Llima 算法和 Colin Fahey 算法，这三个算法分别由 Pierre Dellacherie、Roger Espel Llima 和 Colin P. Fahey 三位专家提出[25]。

这三个算法都是基于贪婪策略的，Pierre Dellacherie 算法和 Roger Llima 算法只考虑到下一步的综合评估，而 Colin Fahey 算法考虑了接下来两步以达到局部最优。

1.3 研究主要内容

本文首先对遗传算法与人工神经网络的基本概念，研究现状进行了说明，提出了遗传算法与人工神经网络在游戏开发中的应用方式，在此基础上，结合自主性 NPC 设计，提出了一种基于遗传算法的神经网络算法 NPG。

结合具体的 Tetris 游戏，本文对 Pierre Dellacherie 算法、Roger Llima 算法和 Colin Fahey 算法三种评估算法进行分析，在此基础上，提出了一种新算法 PD-Colin 算法。

设计了一款自主性 Tetris 游戏，应用了几种算法，并对此作出了评估。

1.4 本文结构

全文又分为五个部分：

第一章 绪论 包括国内游戏产业的现状，存在的问题，国内外游戏人工智能的应用现状。

第二章 遗传算法和人工神经网络的基本说明，对遗传算法与人工神经网络在游戏开发中的应用进行了论述，并提出了一种基于遗传算法的神经网络算法 NPG。

第三章 对 Tetris 游戏中三种估值算法进行了描述，提出了新算法 PD-Colin 算法。

第四章 俄罗斯方块游戏的结构分析与设计与实现，将两个个算法成功运用到俄罗斯方块游戏中。

第五章，对本文的工作进行总结，并对本研究课题未来的发展趋势和可能的发展方向做进一步展望。

第 2 章 遗传算法与人工神经网络

2.1 遗传算法理论

1962年，受达尔文“物竞天择、适者生存”的启发，随着孟德尔遗传变异理论的蓬勃发展，美国密执安大学打Holland教授提出了通过对生物在自然环境中遗传和进化过程进行模拟的全局优化概率搜索算法——遗传算法(Genetic Algorithm,GA)。这是一种多参数、多个体同时优化的方法。

遗传算法第一次被提出后，并未引起多大的关注。70年代，De Jong在计算机上，对遗传算法进行了大量的纯数值函数优化计算实验。80年代，Goldberg对前辈的研究工作进行了归纳总结，提出了遗传算法的基本框架。至此，遗传算法才真正引起关注。在第一届遗传算法应用研究会议上，共收到255篇论文，初步显示出其在解决复杂系统优化问题的良好能力。而且，由于其能对多参数、多个体同时优化，遗传算法求解组合优化问题更是显示出巨大的优越性。

遗传算法是一种新的计算方法，是生物遗传学方法与计算机科学相互结合的新方法。在进化中，对复杂的结构进行简单编码。指导学习和确定搜索方向则通过对简单编码进行遗传操作与自然选择来完成。

2.1.1 遗传算法的定义

遗传算法工作前无需知道求解问题本身，算法对每个染色体进行评价，将问题的解映射为染色体，通过适应值来对染色体进行选择。优胜劣汰则通过适应性好的染色体繁殖机会更多来表现。在算法执行前，需先给出一组假设解，即随机给予一群染色体。然后将这组假设解嵌入到问题中，给出一个评价函数评价其适应度。适应性高的染色体拥有更多的繁殖机会，对染色体进行复制，淘汰适应性低的染色体。然后交叉、变异，产生下一代染色体，再对新群体进行优化，直到产生最优解[26]。

下面给出遗传算法中几个重要概念[15]：

1.）个体与个体空间：个体x即一个字符串，个体空间则是由全体个体组成，记为C。字符串用不同的字符个数表示进制编码。如十进制编码，采用0，1，……，9这10个符号表示；二进制编码则采用0,1两个符号表示；其他诸如八进制、十六进制等同表示。在十进制编码中，待优化问题的可行域用个体空间表示，前提是将个体与决策向量对应。

2.遗传算子：遗传算子有很多种，最重要的三个分别是：选择算子、交叉算子、变异算子[27]。

选择算子：根据优胜劣汰法则，评估个体的优劣程度，决定该个体是繁殖还是淘汰。通过选择算子计算后，适应度低的个体基本被淘汰，适应度高的个体则被保留。常见的选择算子包括：期望值方法、最佳个体保存法、比例选择、随机联赛选择以及排序法。

交叉算子：该算子决定是生物的多样性，是指在相互配对的染色体按照定义好的基因交换规则进行交换，进而形成新个体的计算过程。交叉算子主要应用于二进制编码和十进制编码。十进制编码中，主要的交叉方式为：简单交叉、算术交叉；二进制编码中，主要的交叉方式包括：单点交叉、多点交叉、均匀交叉。

变异算子：该算子决定是新物种的诞生，是指对染色体编码过程中，对某些基因位上的基因用其他等位基因进行替换，从而产生新个体甚至新物种的运算过程。变异算子主要应用于二进制编码和十进制编码。十进制编码中，主要的变异方式为：均匀变异、非均匀变异；二进制编码中，主要的交叉方式包括：基本位变异、高斯变异、均匀变异。

3.适应度函数 $F(t)$：生物个体对环境有个适应度，适应度函数是对个体空间C进行正实数空间映射的过程，其形式化表述为：$F(t):C\to R^{+}$。

遗传算法整个流程基本是封闭的，进化搜索过程不需要额外的外部信息，适应度函数是其唯一的评估标准，个体适应度值则是搜索的唯一参考。因此，适应度函数的构建是整个算法中最重要的一步，直接影响算法的质量、收敛速度甚至决定了算法能否找到最优解[28]。

适应度函数与目标函数 $f(x)$ 属于变换关系，一般有三种适应度函数：

A. 直接将目标函数视为适应度函数。

B. 如果待求的目标函数为最小化问题时，采用界限构造法：

$$F(f(x))=\begin{cases}V_{\max}-f(x) & |V_{\max}>f(x)|\\ 0 & |V_{\max}\le f(x)|\end{cases}$$

如果待求的目标函数为最大化问题，则对应选择即可。其中，$V_{\max}$ 为 $f(x)$ 的最大值估计。$V_{\max}$ 一般采用迭代法求解，先选取一个初始值，虽然一直进化更新。

C. 如果待求目标函数是最小化问题，可以采用尺度变换法：

$$F(f(x))=\frac{1}{1+v+f(x)}\quad |v\ge 0, v+f(v)\ge 0|$$

v为目标函数的保守估计值。

4.数学模型：遗传算法是一种迭代计算，逐步逼近最优解的搜索寻优技术，其数学模型可以表示为：

$$Model(GA)=(C,F,P_0,N,\Phi,\Gamma,\Psi,T)$$

其中：C：个体编码方法；

F：个体适应度函数；

P_0：初始种群；

N：种群大小；

Φ：选择算子；

Γ：交叉算子；

Ψ：变异算子；

T：算法终止条件；

2.1.2 遗传算法的步骤

根据遗传算法的原理，其步骤共可以分解为如下9步[28]：

A. 对目标问题编码；

B. 随机初始化种群；

C. 计算个体空间所有个体的适应度值；

D. 评估适应度，为选择做准备；

E. 选择：将选择算子作用于种群；

F. 交叉：将交叉算子作用于后代；

G. 变异：将变异算子作用于后代；

H. 判断是否达到条件T，不满足则进入C，否则进入I；

I. 输出所有适应度值最优的个体。

可以看出，整个遗传算法流程中，最重要的三个操作：选择、交叉和变异。

2.2 BP 神经网络

人工神经网络(BP)通过对人脑神经元结构进行模拟，通过一系列复杂的算法对数据进行处理，从而达到预测、判别模式、联想记忆等功能[29]。网络分布式矩阵结构的连接权值上携带有大量的知识，而学习过程则通过调整计算权值来实现。游戏角色的建模往往需要建立高维矩阵，传统方法如回归分析、多元统计分析等方法，在处理高维数据方面往往表现得不如人意，于是，引入神经网络便成为不少专家的选择之一。

2.2.1 BP 神经网络基本原理

BP 神经网络是一种对人类大脑神经系统的模拟，属于多层前馈型网络[30]。该网络中，神经元之间的传递函数为 S 型函数，输出为连续值，其值域范围为 0 到 1。BP 神经网络的映射方式非常自由，输入输出可以以任意形式进行非线性映

射。下面对基本流程作简单的说明[31]：

假设：该网络为三层网络；输入节点：x_i；输出节点：z_l；中间节点：y_j；期望输出值：e_l；输入节点与中间节点的连接权值：w_{ji}；中间节点与输出节点的连接权值v_{lj}；训练集$S\{x_i,e_l\}$。

目标：调整w_{ji}与v_{lj}，使$|z_l-e_l|\le\varepsilon$；

方式：给定初始训练集，反复学习训练，使w_{ji}、v_{lj}、ε构成梯度，误差z_l-e_l沿着梯度下降，直到能确定与最小误差对应的网络参数，停止训练。

数学模型：

1..中间节点的输出：$y_j=f(\sum_i w_{ji}x_i-\varepsilon_j)=f(n_j)$其含义为：中间层上，输入第i个节点时，第j个节点的输出。

2..输出节点的输出：$z_l=f(\sum_j v_{lj}y_j-\varepsilon_l)=f(n_l)$其含义为：输出层上，输入第j个节点时，第l个节点的输出。

3.输出节点的误差：$E_l=\frac{1}{2}\sum_l(e_l-z_l)^2$，总误差：$E=\frac{1}{2N}\sum_{l=1}^{N}E_l$。

4.传递函数：$f(x)=\frac{1}{1+e^{-x}}$。

5.权值的修正：$w_{ji}=w_{ji}+\mu\frac{\partial E}{\partial w_{ji}}$。

2.2.2 BP 算法的实现步骤

A.初始化样本，设定权值和阈值；

B．对样本进行训练，输入初始节点值，计算期望输出值与网络输出值的误差；

C. 反复学习训练，使误差沿着梯度下降，直到能确定与最小误差对应的网络参数，停止训练。

2.3 遗传算法在游戏开发中的应用

游戏中，玩家模型是建模不可预测性的最大来源。要让玩家体会到独一无二、与众不同，就必须针对玩家的特点，建立对应的有挑战性的敌人，而不是呆板的NPC。但是，游戏玩家的行为具备很强的自主性，难以预测。

但遗传算法能够解决一定的问题，在给出一定的初始建模方法后，通过遗传算法，搜索到表现最好的解法，然后将之与玩家进行匹配，每位玩家分配到的解法都有可能不同，这正是我们的目标所在[32]。下面我们就遗传算法的几个步骤与

游戏开发结合起来进行说明。

2.3.1 数据编码

遗传算法最重要的特点之一即搜索事先无法预测，这种无法预测性，可以增加NPC的独特属性，增加玩家的挑战性和趣味性。应用遗传算法，可以增加NPC的响应数量，增加或减少种群的响应度[33]。

例如，在魔兽世界这款游戏中，某一法师(玩家类型之一)攻击NPC，我们可以为NPC事先建立多种响应机制，如反击玩家、逃跑、隐藏，甚至呼叫同伴夹击玩家等等。我们将情景与响应分配给染色体，每一条染色体对应一种响应方式，当我们进行选择时，种群的成员就会具有不同的行为方式。

这种情况下，实现起来也比较简单，创建一个工作行为类，染色体数组是类的一个成员变量，数组中的元素代表NPC的行为，只要将行为与染色体进行一一映射，编码的过程将非常简单[34]。

2.3.2 初始化族群

种群的多样性，直接影响找出最佳方案的可能性。而建立种群多样性的最佳方法，就是在初始赋值过程，采用随机函数对每个染色体赋予一个随机行为。 而且，这种初始化还必须能够随着时间及种群的数目随时改变。

2.3.3 适应度函数

游戏中的最佳响应，是个仁者见仁智者见智的问题，很难进行量化。但被广泛接受的一种说法是：NPC能根据不同玩家的不同行为做出较为合理的反应，既然玩家觉得趣味性，又不觉得失真（如：在魔兽世界游戏中，高等级法师玩家遇到低等级的游戏怪物时，低等级游戏怪物依然向着高等级发起冲击，这与正常的逻辑思维严重不符，会降低玩家的挑战性和趣味性）

因此，如何建立一个好的适应度函数，以一定的方式量化和评定行为的最佳响应,便成为找出种群成员中最让玩家觉得有挑战性的非玩家角色最可能的方式。在RPG游戏中，每个角色会被分配一定的血量和体力，当角色被击中后，血量会下降，不同等级的非玩家给予角色的伤害值是不相同的，血量和体力的下降速度也不一样，当血量低到一定程度时，角色会死亡。所以，如果我们对之建模，就可以以血量和体力为参数，作为其适应度[35]。同样的，NPC也必须赋予一定的血量和体力，这样，我们构建出来的适应度函数，收敛速度快，适应度高。

2.3.4 交叉和变异

构建了适应度函数后，我们就可以利用它，计算出个体空间中，所有个体成员的适应度，并将之按照优劣进行排序，找出适应度最差的个体进行淘汰，找出

适应度最好的个体进行复制和繁殖，利用个体特征和属性通过交叉产生下一代。

为了增加种群的多样性，我们引入一个随机突变函数，挑选一定的染色体，随机赋予一个新值。控制这种随机突变的概率，然后利用突变后的染色体与其他染色体进行交叉，产生新的后代。如此往复，数代之后，就容易产生对玩家极具挑战性的非玩家角色。

2.4 NPC 感知系统

在游戏设计术语中，游戏角色所在环境的内部模型叫认知模型(Cognitive Model)[36]。对认知模型，我们能采用精确的数学方式进行定义，这就解决了我们对游戏角色行为和学习模式难以量化的困难，因此，利用认知模型对游戏角色进行自主性建模成为游戏设计人员的方法。

游戏中，一般将非玩家角色(俗称 NPC)的行为定义为：固定的和动态的两种。基于两种行为建立的认知模型也有所区别。固定行为的认知模型建立方式相对简单，设计一些枚举变量，将 NPC 所在环境的领域知识填装到枚举变量中，NPC 系统根据玩家的动作， 自动的分配 NPC 行为，做出对应的反应。

建立动态行为的认知模型则相对困难，目前一般的方法是对 NPC 的行为进行规划，分为目标引导、行为协调以及约束满足。

NPC的感知系统是NPC角色建模的核心之一，结合BP网络的特点，我们采用它作为NPC的感知系统。

对于基于BP网络的NPC的感知系统，有两个重要问题：

1.输入参数的确定：BP网络的第一阶段为计算前向输出，其需要接受节点输入，我们将虚拟游戏的环境特征参数作为其输入节点，以此为基础，进行学习。在游戏设计中，环境变量往往以三种形式存在：枚举变量、布尔变量、集合变量。但是，神经网络无法对这三种数据类型进行直接处理，必须对之进行转化。

2.权重的确定：BP网络的第二阶段即反复调整连接权矩阵，权重的确定直接影响神经网络的收敛性，输出变量的值也直接受权重的影响。必须为每一个输入参数赋予一个权重，确保其能对应一个输出变量。实际上，权重要想确定成功，还必须引入一个激励函数。具体到NPC自主角色的神经网络系统中，我们需要引入一个S型激励函数。

在其中间节点，也就是隐层节点上，我们采用反tansig函数作为激励函数，其公式为[37]：

$$f(x)=\frac{1-e^{-x}}{1+e^{-x}} \tag{3.1}$$

在输出节点上，我们也引入一个激励函数，采用反函数，其公式为：

$$f(x)=\frac{1}{x} \tag{3.2}$$

当我们确定了每个神经元的权值w_i时，如果我们为每个神经元再赋予一个阈值t，则，我们可以确定每个神经元的输出：

$$y=f(\sum_{i=1}^{n} wx_i-t) \tag{3.3}$$

2.5 基于遗传算法的神经网络

神经网络的优化问题，一直是研究人员面临的重点和难点，其直接影响到了神经网络的应用，原始的神经网络在不同领域进行应用时，往往会出现收敛性问题、训练时间过长、泛化误差大等问题。到目前为止，国际上还没有完全统一的理论公式来对神经网络进行优化。

通过自身的经验和反复的实验，今年来，不少研究人员提出了自己的理论和方法，比较有代表性的是：S.Y.Kung 等人[38]提出的代数映射法，其主要思想是将神经网络的权值等同为方程组中的变量，然后利用反向传播算法、代数映射法进行求解，得到最优值。第二种方法是 C.Lebiere 等人[39]提出的节点增长算法，其主要思想是在最小网络的基础上，建立一种多层结构，结构中的单元全部采用自动训练，隐层上的单元则采用逐步添加法。每添加一个新的隐层单元，就冻结该单元的输入端权值。在此基础上，Fahlman 等人[40]提出了改进算法最差层增长算法 WLG，其主要思想是对泛化误差进行处理。再一个影响力较大的方法是剪枝算法，其主要思想是首先构造大网络，然后计算节点对网络的误差，删除影响小的节点，保留影响大的节点，其代表性算法是 A.Maccato 等人提出的结构化神经网络。

90 年代中期，遗传算法被首次引入到神经网络优化领域。神经网络擅长局部寻优，属于一种贪婪策略。遗传算法则属于整体寻优，而且能多点同时处理，两者的结合，取得了非常良好的化学反应。

但是，遗传算法与神经网络的结合也带来了新的问题，编码方式过于单一，未能考虑神经网络的激励函数、操作算子太固定，无法保证算法收敛到最优解的概率为 1，等等。

经过考虑，我们采取两大策略解决以上问题，第一，采用混合编码方式，使其更加适应游戏的设计。第二，采用特殊的操作算子，使其收敛性加快。

2.5.1 混合编码方式

我们引入网络数据包的组成思想，将编码串分成三个部分：节点部分、权重分配函数部分和激励函数部分。利用二进制编码，对节点部分进行编码，用来表

示神经网络隐层数和节点数，节点部分是编码串的固定长部分，长度由神经网络的结构决定。权重分配函数部分根据节点部分，由节点分段后进行异或计算决定。激励函数部分则采用变长设计，其长度由节点部分决定[41]。

节点二进制编码中，0 表示该隐层不存在，即该神经网络只有一个输入节点和输出节点。如串 1101 0011 表示，第一个隐层的节点数为 13，第二个隐层的节点数为 3。串 1100 0000 则表示，第一个隐层的节点数为 12，第二个隐层的节点数为 0，该隐层不存在。

节点串、权重分配函数串、激励函数串组成的新串中，节点串决定了权重分配函数和激励函数的长度，如上面提到的节点串 1101 0011，其长度为 16，权重分配函数的长度应为 14，激励函数长度为 16，前 8 位与节点串相同。激励函数串中，0 表示 tansig 函数，1 表示采用其他激励函数。

2.5.2 混合遗传算子

普通遗传操作算子中，其基本选择分别为：轮盘选择算子，双点交叉算子和多点变异算子。在算法操作过程中，为了保证该算法的收敛到最优解的概率为 1，可以采用保留最佳个体的策略。

结合混合编码方式，我们调整遗传操作算子的选择，对交叉算子和变异算子进行混合[42]。

交叉算子的基本策略为：

1. 首先对节点串进行双点交叉。
2. 根据交叉后的节点串，调整权重分配函数串和激励函数串(交叉过程，节点串会发生改变)。
3. 对权重分配函数串采用双点交叉。
4. 对激励函数串采用双点交叉。形成的新算子作为交叉算子。

交叉策略图如下：

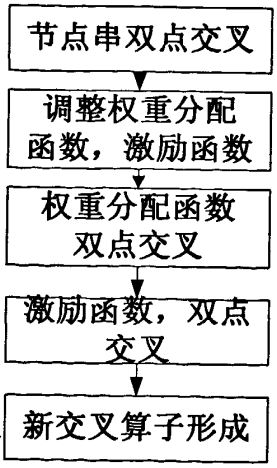

图 2.1 交叉策略图

变异算子的基本策略为：

1. 对节点串进行多点变异。
2. 根据变异后的节点串，调整权重分配函数串和激励函数串。
3. 对权重分配函数串采用多点变异。
4. 对激励函数串采用多点变异。

变异策略图如下：

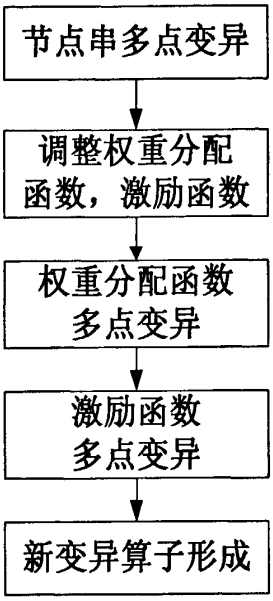

图 2.2 变异策略图

2.5.3 基于遗传算法的神经网络算法 NPG

结合基本的引入遗传算法的神经网络，我们对其进行调整和改进，得到新的 NPG 算法的步骤为：

1. 从 NPC 神经网络中提取权重向量。
2. 对求解权重最优解问题编码（混合编码）；
3. 随机初始化 NPC 种群；
4. 根据权重分配函数，计算单个 NPC 针对虚拟环境的权重值，并对权重进行排序；
5. 选择、交叉、变异(混合遗传操作算子)；
6. 判断是否满足终止条件；
7. 输出新的网络权重群体；
8. 将新权重插入到 NPC 神经网络中；
9. 计算性能，满足，算法终止，否则，转到第一步。

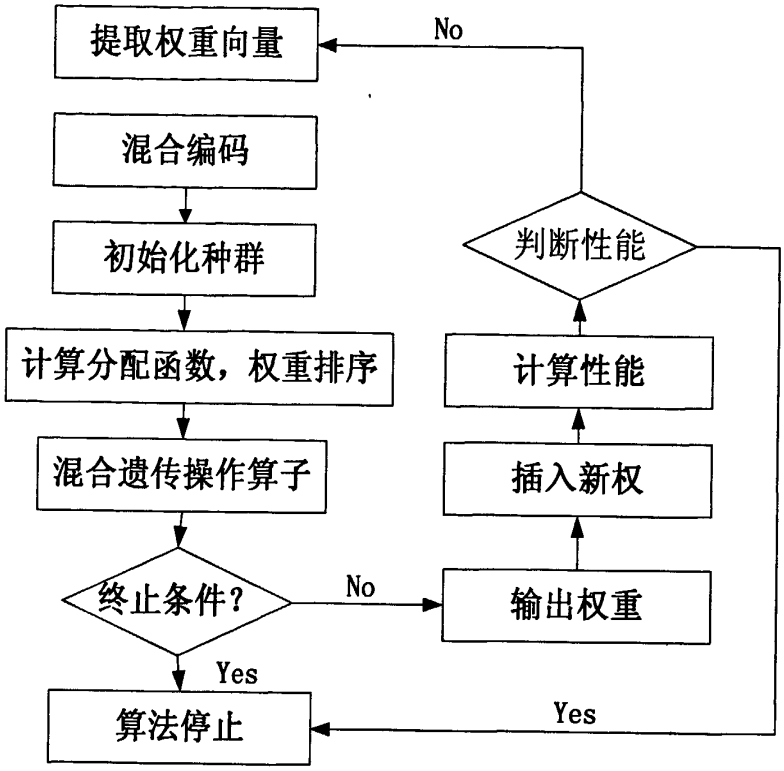

图 2.3 算法流程图

第一步的权重向量确立过程，采用公式 3.1-3.3。第五步，遗传操作算子我们采用特殊的遗传操作算子。

确立了权重后，要建立智能型 NPC，还必须建立各种虚拟环境下的 NPC 行为模型，同时，这种虚拟环境是动态变换的。让玩家觉得有挑战性的 NPC，就必须具备较强的适应度，这种适应度是其竞争力大小的直接反应。

这种竞争，与遗传算法中的选择机制对应，所以，必须针对 NPC 建立一个适应度函数，这个函数必须具备两个特点：第一，学习能力不同的 NPC 其适应度值应该不相同；第二，当环境发生变化时，NPC 的适应度也必须随之改变(实际上，这是很多虚拟网游中智能成长型 NPC 最重要的特点之一，即，NPC 也能升级)。

综合以上两点，结合遗传算法中经常采用的适应度函数，我们提出如下公式：

$$F(t)=\lambda\times\frac{1-e^{-t(\frac{k}{a})^2}}{1+e^{-t}}$$

式中：λ表示修正值，k表示NPC类型，a表示NPC的生命值。

利用NPG算法，我们很容易建立一个智能NPC模型。

第 3 章 智能 Tetris 游戏的估值算法

Tetris 的起源实际上要追溯到八十年代中期，尽管它的人气一直到几年后才开始显露出来。游戏的概念十分简单，如今已经成为一种公认的规则：屏幕顶部以随机顺序落下形状各异的碎块，根据一定的策略，将之拼接成没有空隙的行列。

Tetris 无法赢得最终胜利，根据游戏的设计，坚持的时间越长，游戏速度将越来越快，坚持下去的难度将越来越大。Tetris 之所以吸引玩家在于，某些与坠落的玩具碎片和它们的形状有关的东西，使得哪怕新手也会很自然地企图把它们排列起来，并加以适当组合。

Tetris 中的人工智能，最重要部分便是估值算法，目前最主流的包括三大算法，分别为：Pierre Dellacherie 算法、Roger Llima 算法和 Colin Fahey 算法，三个算法在进攻性或防守性上各有所长，本章在三个算法的基础上，提出了攻守更加平衡的 PD-Colin 算法。

3.1 Tetris 常用估值算法

3.1.1Pierre Dellacherie 算法

Pierre Dellacherie 算法是一种只考虑当前落子的算法，算法过程简述如下：

I. 尝试着对当前落子的每一种旋转变换、从左到右地落子，产生所有落法。

II. 对每一种落法进行评价。

评价函数为：

```
rating = (-1.0) * landingHeight
       + ( 1.0) * erodedPieceCellsMetric
       + (-1.0) * boardRowTransitions
       + (-1.0) * boardColTransitions
       + (-4.0) * boardBuriedHoles
       + (-1.0) * boardWells;
```

landingHeight：指当前落子落下去之后，落子中点距底部的方格数。

erodedPieceCellsMetric：消去行 * 当前落子被消去的格子数。

boardRowTransitions：指各行的“变换次数”之和，一行中从有方块到无方块、无方块到有方块被视为一次“变换”，游戏区域左右边界也视作有方块。

boardColTransitions：指各列的“变换次数”之和。

boardBuriedHoles：指各列中间的“空洞”方格个数之和。

boardWells：指各“井”的深度的连加到 1 的和之和，“井”指两边皆有方块的空列。

评价还包括优先度。优先度在两个局面的评分相同时发挥作用，取评分相同但优先度高者。

优先度的计算方法为：

若落子落于左侧:priority = 100 * 落子水平平移格子数 + 10 + 落子旋转次数。

若落子落于右侧:priority = 100 * 落子水平平移格子数 + 落子旋转次数。

III. 比较每一种落法的评分与优先度。在同为最高评分的落法中，取优先度最高者。

3.1.2 Roger Llima 算法

Roger Llima 算法是一种只考虑当前落子的算法，与 Pierre Dellacherie 算法在过程上相似，算法过程简述如下：

I. 尝试着对当前落子的每一种旋转变换、从左到右地落子，产生所有落法。

II. 对每一种落法进行评价。

评价函数为：

```
int coeff_f     = 260;
int coeff_height    = 110;
int coeff_hole  = 450;
int coeff_y     = 290;
int coeff_pit       = 190;
int coeff_ehole    = 80;
float rating = 0.0f;
rating    = (float)(v);
rating -= (float)(coeff_f          * frontier);
rating -= (float)(coeff_height  * maxHeight);
rating -= (float)(coeff_hole      * holes);
rating -= (float)(coeff_ehole    * eHoles);
rating += (float)(coeff_y          * pieceDropDistance);
```

frontier：所有和左右边界相邻的单元格，如果为空，则此值加 1，它对应得权值为 coeff_f。

maxHeight：整个局面最高的单元格的高度，对应的权值为 coeff_height。

holes：整个局面中“洞”的个数，包括“桥”和“双层桥”，对应的权值为

coeff_hole。

eHoles：试图计算出局面中的“洞”的填补难度，对应的权值为 coeff_ehole。

pieceDropDistance：描述“井”的深度，对应的权值为 coeff_y。

评价还包括优先度。优先度在两个局面的评分相同时发挥作用，取评分相同但优先度高者。

优先度的计算方法为：

若落子落于左侧:priority = 100 * 落子水平平移格子数 + 10 + 落子旋转次数。

若落子落于右侧:priority = 100 * 落子水平平移格子数 + 落子旋转次数。

III. 比较每一种落法的评分与优先度。在同为最高评分的落法中，取优先度最高者。

3.1.3Colin Fachey 算法

Colin Fahey 算法是一种考虑当前落子和下一个落子的算法，算法过程简述如下：

I. 尝试着对当前落子的每一种旋转变换、从左到右地落子，产生所有落法。

II. 对下一个落子的每一种旋转变换、从左到右地落子，产生所有落法。

III. 对下一落子进行评价。

评价函数如下：

```
double weightRowElimination = ( 0.30);
double weightTotalOccupiedCells = (-0.00);
double weightTotalShadowedHoles = (-0.65);
double weightPileHeightWeightedCells = (-0.10);
double weightSumOfWellHeights = (-0.20);
int rowsEliminated = 0;
rowsEliminated = tempBoard.CollapseAnyCompletedRows();
trialMerit   = (weightRowElimination)*(double)(rowsEliminated);
trialMerit += (weightTotalOccupiedCells)
    *(double)(tempBoard.TotalOccupiedCells());
trialMerit += (weightTotalShadowedHoles)
    *(double)(tempBoard.TotalShadowedHoles());
trialMerit += (weightPileHeightWeightedCells)
    *(double)(tempBoard.PileHeightWeightedCells());
trialMerit += (weightSumOfWellHeights)
     *(double)(tempBoard.SumOfWellHeights());
```

weightRowElimination：指落子落下后被消去的行数。

weightTotalOccupiedCells：指落子落下后，整个局面被占用的格子数量，数量越大，越不利。

weightTotalShadowedHoles：指落子落下后，局面上“洞”的数量，也包括“桥”和“双层桥”。

weightPileHeightWeightedCells：指落子落下后，局面最高点的高度，在 AI 操作游戏时，此参数权值相对较小。

weightSumOfWellHeights：指所有“井”的深度之和。

评价还包括优先度。优先度在两个局面的评分相同时发挥作用，取评分相同但优先度高者。

优先度的计算方法为：

若落子落于左侧:priority = 100 * 落子水平平移格子数 + 10 + 落子旋转次数。

若落子落于右侧:priority = 100 * 落子水平平移格子数 + 落子旋转次数。

VI. 比较每一种落法的评分与优先度。在同为最高评分的落法中，取优先度最高者。

3.2 PD-Colin 算法

综合前面三种算法，我们不难发现，防守性较强的算法，着重于对当前局面和候选图形进行了评估。攻击性较强的算法，则着重于对当前落子的变换方式和当前局面最可消除方式进行了计算。由此，我们可以提出：既评估当前局面又参考候选图形，以便达到攻守平衡的效果。在此基础上，我们提出了 PD-Colin 算法。

PD-Colin 算法参考了 Pierre Dellacherie 算法和 Colin Fahey 算法，是一种考虑当前落子、下一个落子以及当前局面的算法，算法过程简述如下：

I. 尝试着对当前落子的每一种旋转变换、从左到右地落子，产生所有落法。

II. 对每种落法产生的局面，对下一落子的每一种旋转变换、从左到右地落子，产生所有下一个落子的落法。

III. 对当前落子和下一个落子所产生的落法进行评价，得到两个评价 P_1, P_2。

IV. 利用线性公式 $L=\alpha P_1+\beta P_2$（其中，α,β 为经验系数，且 $\beta \le \alpha^2 < 0.5$）对两个评价进行综合运算，得到一个新的评价。

评价函数如下：

```
double rowElimination = ( 0.44);
double totalOccupiedCells = (-0.20);
double totalShadowedHoles = (-0.48);
double pileHeightedCells = (0.12);
```

```
double sumOfWellHeights = (-0.23);
double wells=(-0.15);
double rating = (-1.0) * rowElimination
        + ( 1.0) * totalOccupiedCells
        + (-1.0) * totalShadowedHoles
        + (-1.0) * pileHeightedCells
        + (-4.0) * sumOfWellHeights
        + (-1.0) * wells;
int rowsEliminated = 0;
rowsEliminated = tempBoard.CollapseAnyCompletedRows();
trialMerit   = (rowElimination)*(double)(rowsEliminated);
trialMerit += (totalOccupiedCells)
    *(double)(tempBoard.TotalOccupiedCells());
trialMerit += (totalShadowedHoles)
    *(double)(tempBoard.TotalShadowedHoles());
trialMerit += (pileHeightedCells)
    *(double)(tempBoard.PileHeightedCells());
trialMerit += (wells)
     *(double)(tempBoard.TotalWells());
trialMerit += (sumOfWellHeights)
     *(double)(tempBoard.SumOfWellHeights());
trialMerit +=rating;
```

rowElimination：指落子落下后被消去的行数。

totalOccupiedCells：指落子落下后，整个局面被占用的格子数量，数量越大，越不利。

totalShadowedHoles：指落子落下后，局面上“洞”的数量，也包括“桥”和“双层桥”。

pileHeightedCells：指落子落下后，局面最高点的高度，在 AI 操作游戏时，此参数权值相对较小。

wells：指各“井”的深度的连加到 1 的和之和。

sumOfWellHeights：指所有“井”的深度之和。

评价还包括优先度。优先度在两个局面的评分相同时发挥作用，评分相同优先度高者算法最优。

优先度的计算方法为：

若落子落于左侧:priority = 120 * 落子水平平移格子数 + 12 + 落子旋转次

数。

若落子落于右侧:priority = 120 * 落子水平平移格子数 + 落子旋转次数。

评价以评分优先，评分高者算法得分高，若评分相同则比较优先度，优先度高者得分高。

3.3 四种算法的比较分析

前提条件：七种图形的选择算法为随机算法：

pieceIndex = 1 + (randomInteger % 7);

由于 p 为常数(1/7),经过 n 次试验发生 k 次的概率遵循二项式分布规律：

$$P(k) = (1-p)^{n-k} * p^k * \frac{n!}{(n-k)!k!}$$

其中：P=0.0,……,1.0; k=0,1,2……,n;

平均值(mean)=n*p;方差(variance)=n*p*(1-p);

标准差(standard deviation)=sqrt(variance);

实验时我们取n=70次，得出的二次项分布图如图3.1所示：

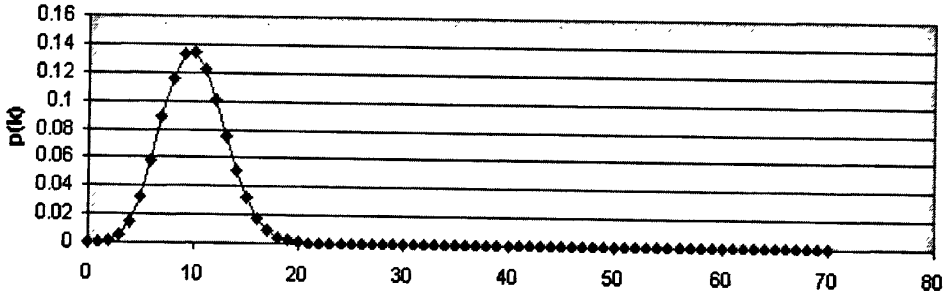

图 3.1 Binomial Distribution(n=70,p=1/7)图

俄罗斯方块游戏进行得好坏的评价标准是得分和安全性。在消去相同行数的情况下，不同的消行组合得分越高，即一次消去多行的情况越多，游戏进行得越好。同样，在整个游戏过程中，最高平面平均高度越低，游戏进行得越好。

了分析三种算法的优劣，分别让三种算法消去总计1000行，分别记录消行的种类和次数，最后计算出得分。每种算法重复三遍测试，标准俄罗斯方块游戏计分方式如表3.1所示。

表 3.1 标准俄罗斯方块计分方式表

消除行数（行）	得分（分）
1	1
2	3
3	5
4	8

Pierre Dellacherie 算法测试结果如表 3.2 所示：

表 3.2 Pierre Dellacherie 算法下落测试表

消除行数	第一次	第二次	第三次	平均值
1	168	144	164	159
2	327	354	340	340
3	53	38	43	45
4	2	4	3	3

Pierre Dellacherie 算法最终平均得分为 1428 分。

Roger Llima 算法测试结果如表 3.3 所示：

表 3.3 Roger Llima 算法下落测试表

消除行数	第一次	第二次	第三次	平均值
1	705	671	701	692
2	144	166	140	150
3	5	4	7	5
4	0	0	0	0

Roger Llima 算法最终平均得分为 1167 分。

Colin Fahey 算法测试结果如表 3.4 所示：

表 3.4 Colin Fahey 算法下落测试表

消除行数	第一次	第二次	第三次	平均值
1	69	143	94	102
2	436	380	416	411
3	23	33	28	28
4	3	1	1	2

Colin Fahey 算法最终平均得分为 1491 分。

由此数据可以分析得出三种算法的特点：

Pierre Dellacherie：Pierre Dellacherie 算法得分比 Colin Fahey 算法稍低，但一次消除 3 行和四行的数量比 Colin Fahey 算法多，同时，消除 1 行的数量则比 Roger Llima 算法少得多，在运行时，可以看到 Pierre Dellacherie 算法一直使局面保持在较低水平面上，因此，它是一种同时兼顾防守与攻击的算法。

Roger Llima：Roger Llima 算法消除 1 行的数量出奇的多，正是因为如此，使其在整个过程中始终保持超低的水平面，但得分却很低，因此，Roger Llima 算法是一种防守算法。

Colin Fahey：Colin Fahey 算法得分最高，它消除 1 行的数量非常低，主要消除 2 行，因此在整个过程中常常运行在非常危险的高位，但同时消除 3 行和消除 4 行的数量没有 Pierre Dellacherie 算法多，因此攻击性并不算强，因此 Colin Fahey 算法是一种得分算法。

三种算法的攻击性排序如下：

Pierre Dellacherie ＞ Colin Fahey ＞ Roger Llima

三种算法的防守性排序如下：

Colin Fahey ＞ Pierre Dellacherie ＞ Roger Llima

由此得出结论：Pierre Dellacherie 算法在攻击中使用，Colin Fahey 算法在防守时使用。

随后我们对PDColin算法进行了等条件下的测试工作，其测试结果如表3.5所示：

表 3.5 PDColin 算法下落测试表

消除行数	第一次	第二次	第三次	平均值
1	144	127	94	122
2	365	379	380	375
3	41	38	55	45
4	1	0	5	2

PDColin 算法最终平均得分为 1488 分。

分析主要的数据我们可以看出，在攻击性上：

Pierre Dellacherie > PDColin > Colin Fahey

在防守上：

Pierre Dellacherie < Colin Fahey < PDColin

PDColin 算法基本达到了我们的设想，攻守基本平衡。

第 4 章 基于 NPG 算法和 PD-Colin 算法的 Tetris

4.1 系统框架

系统整体架构图 4-1 所示，玩家可以控制游戏平台进行传统俄罗斯方块游戏，同时也可以改变算法，让 AI 模块操作游戏平台，进行自动游戏。玩家还可以使用自动操作、快速操作、手工模拟功能，这些操作 AI 模块并不能控制。

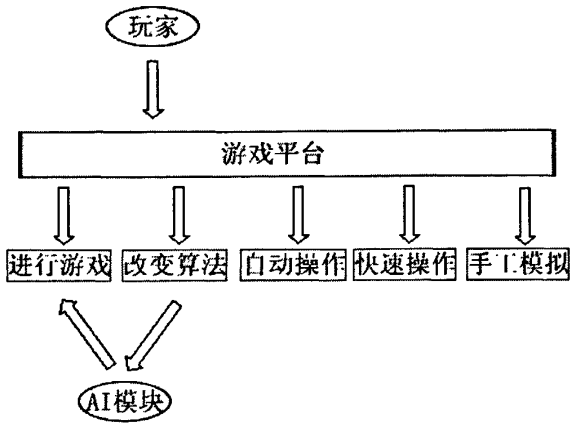

图 4.1 系统框架图

4.2 功能设计

俄罗斯方块作为一款益智游戏（Puzzle Game），场景非常简单，既不需要制作庞大的地图，给玩家所操纵的角色行走，这就可以避免地表层以及地图上的阻挡元素——建筑层的图片资源的涉及，又不需要使用动态的远景图片来营造风格迥异的场景主题。其次俄罗斯方块也没有复杂的对象层数据，人物的动作、画面中出现的动画等都是对象层所覆盖的范畴。益智类游戏不像 ACT 类游戏一样，它没有人物走、跳、跑、攀爬、下蹲、攻击等大量的动作状态，因而在容量上省下了大量的人物动作图片资源。

对游戏的框架部分，我们要提供进行游戏的场地，因此需要给用户提供一个图形界面窗口，还需要显示相关信息，这一部分应该是游戏的基本部分。游戏的主体界面主要由图形界面窗口、信息显示组件、游戏控制按钮组成。

我们需要对用户的选择进行反应，以控制游戏进行、操纵方块动作、判断游戏进度和显示相关信息。因此，需要对这个图形界面窗口的事件进行定义和实现。

游戏界面上的所有动作以及当前游戏的状态等参数，通过我们设定的相关功能按钮以及状态判断标志，传达到游戏内部。我们需要对这些信息进行筛选、分

析和处理，给出合理的反应，使游戏顺利进行。

还需要添加一些附加功能，这些功能的目的或者是为了方便用户的操作，或者是为了界面的美观。

首先，要实现一个游戏封面，它和主窗口同时生成。在主窗口建立之后，随即将主窗口隐藏，在封面出现的时候启动计时器，在指定的时间长度后将封面窗口隐藏起来[43]，让主窗口显现出来即可。这个功能主要是为了游戏的美观。

同样是为了美观，我们定义了圆形按钮组件。

其次，要实现用户对游戏环境的设置。这一部分主要包括方块种类选择、背景音乐、音效设定。我们还要制作简单的帮助，制作“关于”窗口显示游戏相关信息，这些都是从方便用户的角度考虑的。为了实现这些功能，可以在主窗口部分采用一个多页显示组件。

最后，为了调动游戏者的积极性，需要定做一个积分榜，使得游戏结束时能够判断玩家的分数在存档里是不是前 10 名，如果是，则将用户的分数存档。

综上，游戏需要的功能需求有：图形的界面布局（长方形主框，图形预览，Score，Level）；各种 Shape 的数据结构的存储和随机生成；图形界面的绘制以及各控件的显示（主图形，预览图形显示，Score，Level，游戏说明）；时钟控制图形更新（主图的下降）；图形的变换控制；边界控制和图形的保存；消行，得分，升级。

4.2.1 游戏平台

要实现 AI 操作的游戏程序，首先需要设计一个平台，这个平台首先可以让人类玩家进行标准的俄罗斯方块游戏，这样可以很好的把算法从系统中分离出去。

以 MFC 为框架构建一个可供人类玩家使用键盘进行游戏的基本模块，包括难度调整和计分系统。难度调整方式为当难度增大时，方块下落速度也相应增大，计分系统记录游戏中消去的行数，和玩家获得的分数。玩家在游戏中的消行动作一次最少消去一行，最多消去四行。一次消行所得分数必须拉开距离，一次消去多行比多次消去相同数量的单行得分高。计分系统不但记录所得分数，还会记录每种消行的次数，方便测试每种算法的性能。一次消多行的次数越多，算法的攻击性越强。

4.2.2 自动操作

为了在人工和 AI 中切换对游戏平台的控制权，或者更换 AI 算法，需要一个控制开关，当自动操作开启时，程序调用目前选择的 AI 程序，接管游戏平台的控制权进行游戏，当自动操作关闭时，游戏控制权回归键盘，由玩家进行游戏。

自动操作未启动时，游戏玩家可以进行算法的选择和游戏参数的更改，如修改下落速度及预测个数等，然后点击保存，保存成功后，在下一次自动操作开启

时生效。一旦自动操作开始运行，则不允许再进行更改动作。

4.2.3 快速操作

虽然 MIT 证明了 Tetris 的永恒下落是个 NP 完全问题[44]，不过在尽可能长的时间内，一个相对稳定的算法，可以进行上万次下落而不会使游戏结束。同样，对一个算法进行评估也是通过上万次测试，最后通过统计学得出的，因此无论是测试算法的健壮性还是比较两个算法的性能差异，我们都需要进行大量的重复性下落。

真正需要高效计算的无非以下几个过程：把某个方块固定到指定位置（仅在逻辑计算的时候，递归过程中使用的）；判断当前方块是否可以下在某个指定位置上（变形、移动、坠落等等都用此方法）；判断当前形势，根据定义好的判断方案和当前形势保存在的数组，得到一个分数；递归中的传值过程：传递当前局势和下面将要出现的方块。而实际中掉落、攻击对方等等既需要效果，又不涉及复杂运算或频繁运算的过程，不需要高效执行。

但是，正常情况下，即使最高难度的游戏仍然需要至少零点一秒的下落时间（达到肉眼几乎看不清的程度），上万次下落共需要数千秒，如果再以不同的算法进行测试，可能需要数小时。为了快速得到数据，需要使 AI 操作游戏时加快速度，当自动操作开启时，选择极速操作，可以让程序飞快地运行，可以在一分钟内的到数千次乃至数万次的下落数据。

4.2.4 手工模拟

人类玩家的操作速度和操作特点和 AI 有很大不同，如果程序需要扩展人机对战功能，AI 超高速的按键显然对人类玩家是个极大的不公，也使人类玩家面临着巨大的挑战。

当 1990 年“深思”与卡斯帕罗夫战成两次平局后，国际大师们在赛后复盘时感叹，卡斯帕罗夫已经不是在与机器，而是与另一位大师在较量。如今，即使是国际象棋大师，在与电脑 5 分钟的快棋赛中已几乎没有获胜的可能；在 30 分钟的快棋对抗赛中也很难获胜；只有在更长时间的比赛中，如各方在 40 步棋内有 2 小时的思考时间并附加许多有利于人的规则后，高手才有赢的机会。更有无数的棋手在网上与电脑程序对弈，其中也包括大师级的棋手，即使战局持续很长时间，但很少有人能赢过电脑。

最引人注目的“人机大战”当属俄罗斯国际象棋大师卡斯帕罗夫与“深蓝”的较量。在 1996 年 2 月的比赛中，卡斯帕罗夫以 4：2 的比分获胜。但其后研究人员把“深蓝”加以改良，次年 5 月再度挑战卡斯帕罗夫，最终“深蓝”以 3.5 - 2.5 的比分战胜了卡斯帕罗夫。

因此当手工模拟开启时，AI 的操作频率被控制在一个人类玩家也可以达到的

水平，使对战成为可能。只有在自动操作开启时，手工模拟才能生效。

如果只是为了测试 AI，开启手工模拟可能会在方块层数较高的时候使得 AI 操作不到位，发生 AI 性能降低的情况。

4.2.5 基于 NPG 算法的参数调整

俄罗斯方块中，即将下落的图形对整个游戏过程具有很强的影响力，例如：如果我们的候选图形始终是 I 形，这将大大的简化游戏过程，使游戏几乎无限时间的进行下去，使得游戏丧失挑战性。

以往的做法是随机生成 3-4 个候选图形，但这样的设计有一个很大的问题，无法体现游戏的优越性，没有根据当前局面的特点(如玩家已获积分、已玩时间，当前局面，游戏的难度等等)，给出一个对应的候选列表。

我们可以将 Tetris 中的这部分 AI 当做大型游戏中的 NPC，将当前局面的特点等同于大型网游中的 NPC，于是，我们可以将第二章中提到的 NPG 算法引入到候选列表的生成过程。

利用 NPG 算法，对游戏参数进行自动调整，对盘面进行全面的评估，生成不同的候选图形，增强游戏的挑战性和趣味性，经过实践证明，是可行的。

4.2.6 算法选择

本论文是为了研究不同的评估算法对俄罗斯方块自动运行性能的比较分析，所以，在增强程序的可扩展性的基础之上，AI 使用的算法可以由玩家进行选择。

俄罗斯方块游戏的 AI 由于无法深层次递归，必须重单步策略[21]。

洞的维度：（越少越好）

实洞：无论如何都无法通过任何移动消除的洞

虚洞：可以通过左右移动消除，但是如果纯下落无法消除的洞

高低关系的维度：

是否有平台：连续 2 个以上一样高的为平台，越多越好；

是否有 1 个高度差的高低台：左高右，右高左，至少各有一个，否则略微不好；

是否有 2 个高度差的高低台：左高右，右高左，可以各有一个，再多了就不好了；

是否有 3 个以上的高低台：可以有一个，再多了就不好了；

总得高度差：在 4 个高度差之内是可以接受的，否则越大越不好；

总高度：总高度越低越好；

与对手的关系维度：

对手是否有要攻击的行，如果有，则总高度计算时加上 对手可能攻击的行数。

对手是否比较危险，且自己有攻击的行，如果有，可以主动攻击对方。

策略选型的维度：

重攻击：攒行为主，一行消除惩罚，两行几乎不奖励，三行四行大奖励 ；

重平衡：攻守平衡，一行几乎不奖励，2，3，4 行均奖励，平房级别关系 ；

重防守：防守为主，一行也奖励，尽量消除，甚至不计出现一下小空格；

当下落差不多再横移发生的时候，大多有两种情况：

程序开始时，给出的是无论怎么放都有虚洞的子，如果不使用此种战术，则容易出现实洞。

程序运行过程中，因候选图形形状与目前局面不匹配，故意制造虚洞，以便与接下来的方块配合，消除虚洞。

综合分析这两点：程序初始运行中，少量的实洞并不影响当前局面，可以通过快速消行减少该影响。中途形状不合适能从评价中得到体现，不需要刻意制造此种移动。但是，这种情况不是绝对的，当程序运行很长时间后，加入单步判断，将使程序运行效率大大提升，减少因缺少横移判断导致卡壳的情况。

化简以后我们将不再判断虚洞相关的问题，只要有洞，就是实洞，可以降低很多运算。

本 AI 部分的程序体共包含了 4 个部分。

评价参数定义部分：定义相关评价参数的数值，参数数值定义的越合乎逻辑，AI 技术的效果越好[45,46] ，我们采用 NPG 算法对参数进行调整。

评价函数实现部分：实现各种维度的评价，此处注意要使用定义参数的值，同样的，该值可以用 NPG 算法进行优化，不能使用内部值，否则不易于以后微调。

AI 调用之前的处理部分：判断是否需要进入 AI，以及 AI 前的一些处理变量的清理工作。

AI 实现部分：形成一个递归算法（本文可以不用递归，因为最多就两层）。

更好的 AI 程序框架。

标准评价函数部分：定义几种标准的状态值获取函数 。

读取用户自定义策略类：在处理前预先读取 5 所描述的内容，转存在内存中形成一个逻辑处理片段 。

AI 调用之前处理：同上一种框架 。

AI 实现：同上一种框架。

程序外部编写实现某局面评价的策略：如评价参数定义，评价使用的函数，各种情况的条件等。

考虑到以上各个因素，结合当今世界 Tetris 界广泛使用的几种评估策略，我们最终选定了三个评估算法，分别是：Pierre Dellacherie、Roger Llima、Colin Fahey。

三种算法都使用不同的特征参数，因此侧重点也不同。不同的算法可以使用快速操作清晰的测试出性能差异。

同时对三大算法进行分析测试后，在 Colin Fahey 和 Pierre Dellacherie 的基础上提出了攻守更加平衡的 PD-Colin 算法，并进行了比较和验证。

4.3 详细设计

详细设计的类和部分代码见附录 I，本节只做出针对性说明。

4.3.1 数据存储

游戏中有大量相关的数据需要存储,如编辑器产生的关卡数据,人物,物品属性的配置等等,并且目前的程序界，很少针对每一名玩家设计一种自己的数据格式。程序员需要一套文件格式，一套解析，处理代码能够处理所有的程序。由于一局游戏中产生方块数量巨大，而每个方块由于初始化需要，会存储所有种类方块的形状数据，因此所有方块形状数据使用位存储在一个整型数组中，在需要初始化某一方块形状时，将其位存储形式从整形数组中截取出来，转换成 0\1 矩阵存储方式[47]。

因为传统解决方案有着其天生的缺点，Excel 对于越来越大的游戏,越来越多的游戏数据有些吃力，而使用数据库方案对于光是保存游戏数据(仅指编辑数据，不包括游戏的运行时数据)又显得太过臃肿。我们需要的存储特点为：

1.有着文本存储的方式，便于开发期调试和修改，也能直观的查看到改变和开发期的版本控制 merge。

2.有二进制存储方式，在发布后可以使用这样的方式，减少 parse 时间，并且，从文本方式切换到二进制方式是有很简单解决方案的.

3.parse，序列化的代码编写要足够的简单，起码不会随着数据量的扩大而扩大到难以掌握的地步.

4.多语言支持，你总能在你喜欢的语言中使用

5.在数据之间建立关联，嵌套关系要足够的简单，这样才能简单的表示丰富的内容，而不是通过另外再建一套数据来表示关联。

基于以上要求，则需要我们的数据存储格式尽可能的简单和通用，同时要求效率要比较高，所以我们选择了二进制存储。

4.3.2 方块创建

一个方块的创建过程如图 4.2 所示，首先游戏窗口类调用一个方块类的 NewShape 方法和 InitData 方法，初始化方块，接下来方块类现调用 DrawCell 画出单元格，在调用 Draw 画出整个方块的形状。

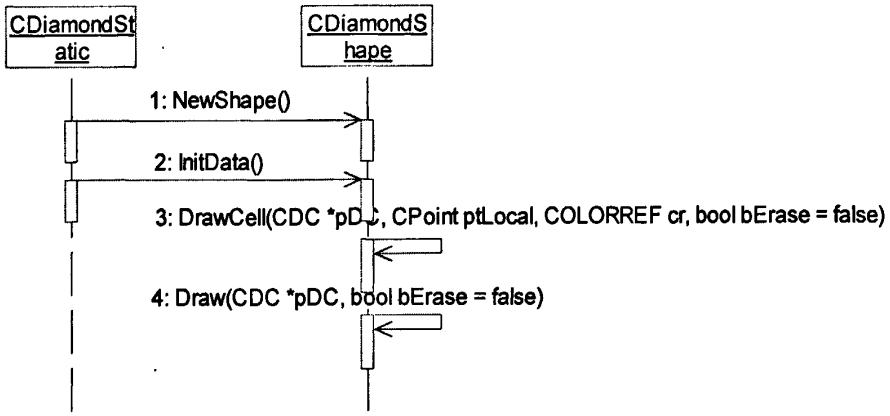

图 4.2 方块创建过程图

假如我们先用数组定义好物体的形状，然后移动变化，那显得过于麻烦。可以看到这些物体都是更小的方块组成的。(我们用四个小方块组成一个大的物体)。用四个小方块，通过相对位置的不同可以变化成各种各样的不同形状。

我们最终目的是在二维数组中画不同形状的物体，因此，必须事先了解物体的位置。如果我们对二维数组的每个元素再抽象出一个数据结构，在这个数据结构中保存当前它在二维数组中的位置(行列值)。每个物体都是有边界的(把四个小方格全部盛下的最小矩形边界)，我们把这个物体的四个小方块当前在 block 中的行列值，记录下来作为边界，放置在上下左右中。下面很多地方是用到这几个值的：

我们规定每个小方块的实际大小是 (24*24)像素。

因此，该游戏区域实际上是在 height = 18*24 ，width = 10*24 大小的矩形内(要注意数组的行列和矩形宽高的对应，col 对应的是 width， row 对应的是 height)。

生成物体的过程即为小方块产生对应的行列值，且该行列值并非随机的，必须遵循一定的限制。例如：我们让第一个方块的行列值为 <-1，4> ，然后依据这个方块在它的上下左右四个方向随机生成第二个方块，然后再根据第二个生成第三……根据第三个生成第四个。每个方块都是上一个方块的上下左右四个方向中非上一个方块的地方，此种生成方式为正确的，相反，则应舍弃。知道了上一个方块的位置，我们只要一个相对于上一个方块位置的偏移就好了。偏移方式我们采取如下规则：

(0，1) ---- 右 (-1，0) ---- 上

(1，0) ---- 下 (0，-1) ---- 左

我们随机在这四个偏移量中拿出一个，生成下一个物体，但是又不能和上一

个物体重叠。通常的方法为：这次生成的偏移量不等于上一次生成偏移量的取反。

4.3.3 方块下落

每个方块下落过程如图 4.3 所示，首先调用 SetpShape 开始下落过程，接下来调用 ShapeValid 进行下落合法性检查，如出现越上界的情况，说明游戏结束，调用 KillTimer 和 EndGame 结束游戏。如移动不合法，则操作无效，直到本次下落，若没有其他合法操作，则调用 SetTimer，准备进行下一次下落。若操作合法，无越界情况，则直接调用 SetTimer，进行下一次下落。

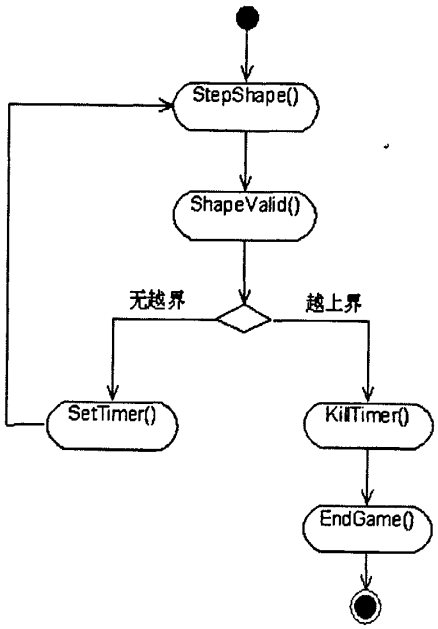

图 4.3 方块下落过程图

方块的自然下落采用采撷时钟中断服务程序的做法[35]。时钟中断大约每秒钟发生 18.2 次。处理时钟中断分为 3 步：

1. 截获原时钟中断向量；
2. 定义新的中断服务函数（即中断服务程序）。
3. 设置新的时钟中断向量。

4.3.4 方块旋转

俄罗斯方块共有七种共十九个图形，对于旋转，我们只需要逐个写出旋转后的图案便可以实现。对于方块翻转来说，当检测到玩家按下翻转键时，可以先检测便修改原始数组中的值，然后检测方块即将达到的新位置是否有其他方块。如果没有，则刷新画笔，在下落位置画出新方块即可。如果存在其他方块，说明方块不能翻转，则将已经修改过后的数组恢复即可。

方块的翻转方法较多，本论文采用坐标变换法[36]，将 Board 与二维平面坐标对应起来，然后直接在翻转前的基础上进行修改，针对不同的方块做不同的变化。

同时注意检测边际范围，当翻转后的图形发生越界时应限制该种图形变换。

4.3.5 方块消行

方块消行的前提是判断当前方块是否已到底。到底有两种概念，一种是已经到达底部，一种是下面碰到了另外的方块。当判断底部加入当前方块已满后，可以进行消行工作。根据积分规则，对总分进行修改，然后擦除当前行，对当面局面中所有方块的坐标进行修改。

具体代码见附录。

4.3.6 方块升级

方块升级依赖当前积分，当积分达到某一阈值时，进入下一个阶段，此时，游戏的速度，游戏的局面都将发生一定的变化。同时，为增加玩家的乐趣，本游戏设计了手动修改，玩家可以自行选择游戏的难度。

4.3.7 程序界面

程序整体界面如图 4.4 所示，可通过菜单进行操作，也可使用工具栏内相同的选项进行操作。工具栏内的选项自左向右分别为：新建游戏，自动操作，手工模拟，一步到位，快速操作，Pierre Dellacherie 算法，RogerLlima 算法，Colin Fahey 算法，PDColin 算法和帮助。

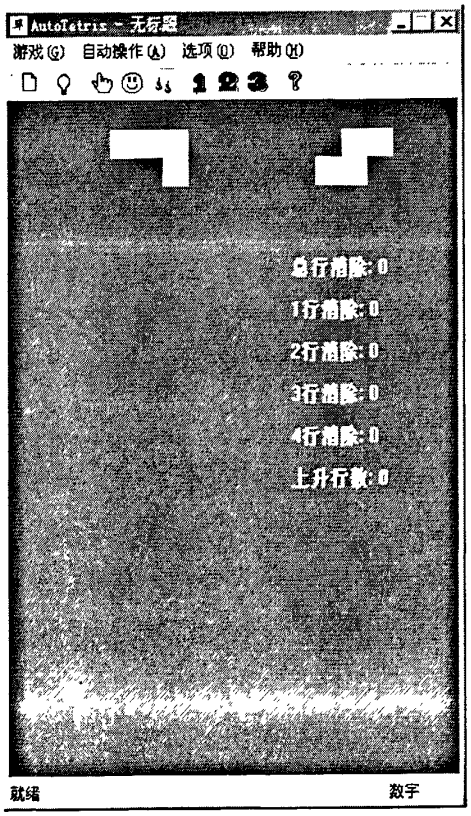

图 4.4 程序整体界面

工具栏下方左侧为游戏区，游戏区是由 10×20 个单元格组成，在 15 行上方有一条红线，所有 AI 的变形和移动操作都是在方块到达红线之前完成的，换言之，方块一旦下落接触红线，就开始没有任何变化的快速下落过程（这是四格方块变形所需要的最小空间）。

游戏区右侧的是统计系统和下一个方块的提示器。下一个方块(如果有需要也可以改为下两个方块)将显示在提示区。统计系统可以统计目前共消去的行数，和每种消行形式出现的次数，极大地方便了数据的统计。同时统计区对得分进行了统计，使得程序能量化算法的攻击力。

键位设定界面如图 4.5 所示，分别对人工操作和自动操作所涉及键位进行了设定

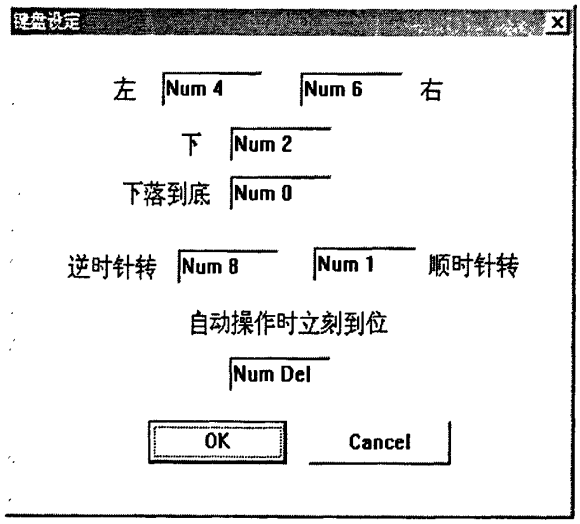

图 4.5 设定键位界面

游戏选项设置界面如图 4.6 所示，对游戏界面，和游戏中需要使用的一些功能性参数进行了详细的设定。

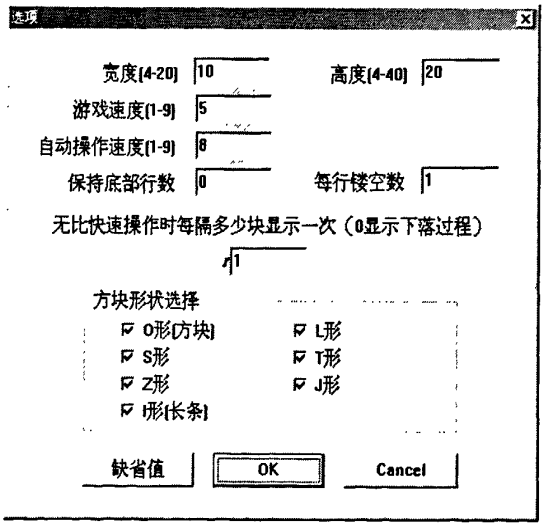

图 4.6 游戏选项设置

结 论

遗传算法是一种随机全局优化算法，属于自适应的全局优化概率搜索算法。GA 由三大算子组成，分别是：选择、交叉和变异。GA 具备一定的智能性，在搜索过程中，算法能够自动获取、积累相关知识，而且其搜索过程为自适应控制过程，搜索完成后，根据相应计算，能达到最优解。

游戏中，玩家模型是建模不可预测性的最大来源。要让玩家体会到独一无二、与众不同，就必须针对玩家的特点，建立对应的有挑战性的敌人，而不是呆板的 NPC。但是，游戏玩家的行为具备很强的自主性，难以预测。

遗传算法能够解决一定的问题，在给出一定的初始建模方法后，通过遗传算法，搜索到表现最好的解法，然后将之与玩家进行匹配，每位玩家分配到的解法都有可能不同，这样就达到了个性化的目的。

而神经网络在游戏开发中，表现出了很强的适应能力，如对地图的边缘检测、搜索回路等。基于贪婪策略，我们选取 BP 神经网络建立自主角色 NPC。

本文首先分析了遗传算法与神经网络在游戏人工智能中国内外研究现状，对本课题的研究背景与意义进行了详细的说明，分析了遗传算法与人工神经网络相结合，应用于游戏设计中取得的成果，以及这些应用中常见的问题。

遗传算法存在的主要问题：第一、种群初始化对于遗传能否得到最优解至关重要，种群选取不合适，得到的最优解有可能是错误的。第二、变异过程不可控，无法保证整个算法完全收敛，应用于游戏中，一旦发散，将导致整个游戏崩溃，给游戏公司造成重大损失。

BP 神经网络存在两大问题，第一、输入参数的确定：BP 网络的第一阶段为计算前向输出，其需要接受节点输入，本文将虚拟游戏的环境特征参数作为其输入节点，以此为基础，进行学习。在游戏设计中，环境变量往往以三种形式存在：枚举变量、布尔变量、集合变量。但是，神经网络无法对这三种数据类型进行直接处理，必须对之进行转化。第二、权重的确定：BP 网络的第二阶段即反复调整连接权矩阵，权重的确定直接影响神经网络的收敛性，输出变量的值也直接受权重的影响。必须为每一个输入参数赋予一个权重，确保其能对应一个输出变量。实际上，权重要想确定成功，还必须引入一个激励函数。具体到 NPC 自主角色的神经网络系统中，需要引入一个 S 型激励函数。

经过反复的实验，本文最终选取两个函数作为我们的激励函数，反 tansig 函数作为隐层节点的激励函数，反函数作为输出节点的激励函数。最后，本文确定了输出节点的输出函数。

遗传算法应用在游戏中，最重要的步骤有：数据编码、初始化种群、适应度函数、选择、交叉和变异。本文在总结普通遗传算法的实现步骤的基础上，结合游戏开发中的特色，采用遗传算法与神经网络相结合的方式。

遗传算法与神经网络的结合也带来了新的问题，编码方式过于单一，未能考虑神经网络的激励函数、操作算子太固定，无法保证算法收敛到最优解的概率为1，等等。

经过反复实验，本文采取两大策略解决以上问题，第一，采用混合编码方式，使其更加适应游戏的设计。第二，采用特殊的操作算子，使其收敛性加快。引入网络包的思想，对编码进行分割。在混合编码的基础上，对交叉算子和变异算子分别进行改变，得到新的操作算子。在混合编码和混合操作算子的基础上，本文提出了一种可以运用于游戏开发中的新算法 NPG 算法。对算法的思想、实现步骤以及算法的流程进行了说明，并给出了其在游戏运用中的方法。

本文编写了一个简单的人工智能游戏俄罗斯方块，以此作为算法实践的载体。虽然俄罗斯方块游戏相对简单，选择俄罗斯方块游戏而没有选择其他更复杂的游戏，主要原因是大型游戏都属于团队合作的成果，本文不好拿来引用，一个人开发一个大型网游困难太大，加上游戏之间有些原理是相通的，能在俄罗斯方块游戏中运行良好的 AI，拿到其他游戏中，稍加修改，也是可用的。

俄罗斯方块游戏中，很重要的一大算法即评估算法，本文分析了世界上三大主流评估算法，包括 Pierre Dellacherie、Roger Llima 以及 Colin Fahey 算法，经过数以千计的下落测试，我们分析得出，在攻击性方面 Colin Fahey 算法最强，Pierre Dellacherie 最弱。在防守性方面，Pierre Dellacherie 最强，Colin Fahey 算法，Roger Llima 攻守相对平衡，但其容易陷入搜索过度，形成死循环。

本文综合 Colin Fahey 算法和 Pierre Dellacherie 在攻守方面的基本思想，提出了一种新的评估算法 PD-Colin 算法，经过反复实验，达到了攻击力强，防守能力突出的特点。

本文将遗传算法与人工神经网络复合的新算法 NPG 算法应用到游戏控制器中，与 PD-Colin 评估算法相结合，对玩家水平进行评估，生成合适的游戏局面，玩家水平高，得到的一系列候选落子将对当前局面适应度比较差。玩家水平低，得到的一系列候选落子，对当前局面的适应度能力好，能让玩家玩得更长久。

本文第四部分，对游戏的概要设计和功能设计给出了必要的说明，对游戏中重要函数与功能给出了详细的代码说明，并附带了游戏的界面。

本文针对俄罗斯方块研究了一个模板，但是，由于经验不足，模板并没有成为唯一的估值参数。而且，本模板的计算速度没有获得实质的提高，对搜索深度的增加非常不利。另外，本文没有采用模板估值中比较常见的 MPC 搜索算法进行事先剪枝，搜索深度最多只能达到 6 层。

参考文献

[1] Galyean, Blumberg. AI for Game Developers. AI Game Programming Wisdom, 2001, 12(41): 151-164

[2] Galyean, Blumberg. Reynolds CW Flocks, Herds: A Distributed Behavioral Model. AI Game Programming Wisdom, 1999, 3(4): 161-168

[3] David M Bourg , Glenn Seemann. AI for Game Developers. O'Reilly: 125-136

[4] 张玉孔. 电脑游戏中的人工智能. 科技信息, 2007, 18(13): 149

[5] 王秀坤, 刘健男. 优化博弈问题评估函数参数的自适应遗传算法. 计算机工程与应用, 2009, 45(20): 42-44, 51

[6] 张小川, 陈光年, 张世强, 孙可均, 李祖枢. 六子棋博弈的评估函数. 重庆理工大学学报: 自然科学, 2010, 2: 64-68

[7] 岳金朋, 冯速. 博弈树搜索算法在中国象棋中的应用. 计算机系统应用, 2009, 9: 140-143

[8] David M Bourg , Glenn Seemann. AI for Game Developers. O'Reilly: 125-136

[9] Mat Buckland. AI Techniques For Game Programming. PRIMIER Press: 27-31

[10] Morris Daniel. Neural-Net AI and Fuzzy Logic. PC Gamer, 1999, 7(13): 82

[11] F S. J. Welsh. Perceiving Information: building better game AI by seeing ideas. Master's thesis, University of Technology, Sydney, 2004, 27(11): 60-78

[12] A. Nareyek. AI in Computer Games. ACM queue, 2004, 1(10): 58-65

[13] J. Laird. Using a computer game to Develop Advanced AI. Computer, 2001, 34(7): 70-75

[14] Hancock John. Lavigating Doors, Elevators, Ledges, and Other Obstacles. AI Game Programming Wisdom. Charles River Media, 2002, 13(15): 80-89

[15] 席裕庚, 柴天佑. 遗传算法综述. 控制理论与应用, 1996, 13(6): 697-708

[16] 王慧, 刘宝坤. 一种改进的遗传算法应用. 天津理工学院学报, 1998, 14(4): 62-66

[17] Peng Huo, Simo C. K. Shiu, et. Application and Comparison of Particle Swarm Optimization and Genetic Algorithm in Strategy Defense Game. Proceeding of the 2009 Fifth International Conference on Natural Computation, 2009, 7: 557-579

[18] Jong-Yih Kuo, Yuan Cheng Ou. An Evolutianary Fuzzy Behaviour Controller Using Genetic Algorithm in RoboCup Soccer Game. Proceeding of the 2009

Ninth International Conference on Hybrid Intelligent Systems, 2009, 6: 283-301

[19] TE. Revello, R. McCartney. Generating war game strategies using a genetic algorithm. Proceeding of the Evolutionary Computation on 2002, 2002, 1(2): 125-141

[20] Anurag Bhatt. In search of no-loss strategies for the game of using a customized genetic algorithm. Proceeding of 10th annual conference on Genetic and evolutionary computation, 2008, 23(21): 170-185

[21] Bengio, Hinto. Learning Deep Architechtures for AI. Foundations and Trends in Mechine Learning, 2009, 2(1), 37-51

[22] Rumelhart, Bernard Widrow, Michael A. Lehr. The basic ideas in neural networks. Communications for the ACM. 1994, 37(3), 661-677

[23] Kaixiang Peng, Xuli Zhang. Classification Technology for Automatic Surface Defects Detection of Steel Strip Based on Inmproved BP Alogorithm. Proceedings of the 2009 Fifith International Conference on Natural Computation. 2009, 185-197

[24] Colin P. Fahey. http: //www. colinfahey. com/tetris/tetris_en. html

[25] Heidi Burgiel. How to lose at Tetris. Mathematical Gazette: p. 1997, 11(7): 194

[26] 丁承民, 张传生等. 遗传算法纵横谈. 信息与控制, 1997, 26(1): 40-48

[27] P. Aksenov. Genetic algorithms for optimising chess position scoring. Master's Thesis, University of Joensuu, Finland, 2004, 7(6): 50-58

[28] Dewdney K A. Exploring the field of genetic algorithms in primordial computer sea full of flibs. Scientific American, 2005, 253(5): 21-32

[29] Wu Junxin, Guo Zhe. A class of active queue management algorithm based on BP neural network, Proceedings of the 21st annual international conference. 2009, 111-123

[30] Yahya H, Lakmal D, Kaspar Althoefer. Stability analysis of a three-term backpropagation alogorithm. Neural Networks. 2005, 18(10): 215-227

[31] Yuemei Xu, Hong Zhang. Study on the Improved BP Algorithm and Application. Proceedings of the 2009 Asia-Pacific Conference on Information Processing. 2009, 616-625

[32] 蒋宁, 翟玉庆. 一个基于神经网络和遗传算法的游戏自主角色的设计. 计算机应用, 2007, 5: 1280-1282

[33] 杨科选, 梁昔明. 遗传算法在游戏开发中的应用. 计算机系统应用, 2009, 5: 128-130

[34] Masanori Sugisaka, A genetic algorithm approach used to generate the neural

network structures, Proceeding of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Kyongju, South Korea, 1999, 1(2): 763-768

[35] Srinivas M, Patnaik L M. Adaptive probability of crossover and mutation in genetic algorithms. IEEE Trans on SMC, 1994, 24(4): 656-667

[36] 方约翰. 人工智能在计算机游戏和动画中的应用——认知建模方法. 北京: 清华大学出版社, 2004

[37] 樊会元, 王尚锦. 遗传算法引入进化方向算子的一个改进及应用. 西安交通大学学报, 1999, 33(5): 45-48

[38] S. Y. Kung, M. W. Mak. Biometric Authentication: A Machine Learning Approach. Prentice Hall Press. 2001, 15(17): 221-229

[39] C. Lebiere. Fuzzy neural network structure identification based on soft competitive learning. International Journal of Hybrid Intelligent Systems. 2007, 4(4): 65-78

[40] Fahlman. Design of structural modular neural networks with genetic algorithm. Andvacnces in Engineering Software. 2003, 34(1): 171-191

[41] Anurag Bhatt. In search of no-loss strategies for the game of using a customized genetic algorithm. Proceeding of 10th annual conference on Genetic and evolutionary computation, 2008, 23(21): 170-185

[42] Amine Boumaza. On the evolution of artificial Tetris players. Proceeding of the 5th international conference. NewYork, 2005: 93-101

[43] Heidi Burgiel. How to lose at Tetris. Mathematical Gazette: p. 1997, 11(7): 194

[44] Erik D. Demaine, Susan Hohenberger and David Liben-Nowell, Tetris is Hard, Even to Approximate. In Proceedings of the 9th International Computing and Combinatorics Conference (COCOON 2003) , 2003, 11(15): 111-123

[45] Reinforcement Learning Playing Tetris . Yael Bdolah & Dror Livnat, 2001: 27-37

[46] Niko B¨ohm, Gabriella K´okai, Stefan Mandl. An Evolutionary Approach to Tetris. MIC2005: The Sixth Metaheuristics International Conference. 2007, 22(23): 118-131

[47] G. Kendall and G. Whitwell. An evolutionary approach for the tuning of a chess evaluation function using population dynamics. In Proceedings of the 2001 Congress on Evolutionary Computation, pages 995--1002. IEEE Press, World Trade Center, Seoul, Korea, 2001, 15(7): 128-141

致 谢

自确定题目后，便开始努力查资料，做准备，由于是在职研究生，不像脱产的研究生，每天从早到晚都是工作，甚至晚上还要加班，毕业设计的时间很难保证。通常是晚上等家人都已休息以后，才去阅读文献，翻译文章，参阅代码，挤出时间去做实验、写代码，时间很紧，工作方面压力也非常大。

公司的合作伙伴中间有游戏开发的项目，自己对这方面也比较感兴趣，遗传算法和神经网络在AI中的运用，这几年也是一大热门，不少专家在这方面做出了突出的贡献。不过，在研究过程中，比较诧异的发现，很多研究只能看到思想，无法对其进行验证，能够借鉴的地方很少，这也给研究过程带来了不少困难。

选择俄罗斯方块游戏而没有选择其他更复杂的游戏，主要原因是大型游戏都属于团队合作的成果，自己不好拿来引用，加上游戏之间有些原理是相通的，能在俄罗斯方块游戏中运行良好的AI，拿到其他游戏中，稍加修改，也是可用的。

研究中，首先要感谢我的导师林亚平老师。本文及课题是在我的导师林亚平老师指导下完成的。在毕业设计期间，无论是毕业设计的选题、开题，还是在设计和撰写的过程中林老师都对我进行了指导，使我的课题可以顺利完成。他能在百忙之中抽出时间来指导我，对此我深表感谢。

另外，在论文写作过程中，不少同事也给了我很多帮助。由于很多资料都是英文的，他们在文献翻译上给了出了不少意见和建议。这省去了我不少时间，非常感谢他们。

特别感谢我的父母和妻子。他们给予我殷切的期望和鼓励，并且在工作期间给予我亲切的关怀，让我可以一心投入工作和毕业设计之中，无后顾之忧。

最后感谢各位专家百忙之中对于本文的审阅！

余颖

2011年4月

附录 A 本文用到的关键代码

CDiamondShape 类是方块形状类，类声明如下：

```
class CDiamondShape : public CObject
{
public:
    CDiamondShape();
    virtual ~CDiamondShape();
    CDiamondShape( const CDiamondShape &sp );
    operator=( const CDiamondShape &sp );
public:
    //画某一种方块，bErase = true 则擦除原方块，覆盖成新方块，否则
    //为画新方块
    void    Draw( CDC *pDC, bool bErase = false );
     //画某一种方块的一个单元格
    static void  DrawCell( CDC *pDC, CPoint ptLocal,
                        COLORREF cr, bool bErase = false );
public:
    //初始化一个方块，包括颜色和形状数据
    void    NewShape();
     //初始化方块形状数据，把方块形状从位存储形式转化为数组形式
    void    InitData();
private:
    int     m_nID;          //方块 ID
    int     m_nShape;       //方块形状编号
    CPoint  m_ptBase;       //位置
    int     m_nData[3][3];
    COLORREF m_crData[5];
     friend class CDiamondStatic;
};
void CDiamondShape::InitData()
{
   int nData[][4] = {
```

```
        //      { 000111000, 010010010, 000111000, 010010010 },
        { 0x038, 0x092, 0x038, 0x092 },
        //      { 010010011, 000111100, 110010010, 001111000 },
        { 0x093, 0x03c, 0x192, 0x078 },
        //      { 010010110, 100111000, 011010010, 000111001 },
        { 0x096, 0x138, 0x0d2, 0x039 },
        //      { 110110000, 110110000, 110110000, 110110000 },
        { 0x1b0, 0x1b0, 0x1b0, 0x1b0 },
        //      { 110011000, 001011010, 110011000, 001011010 },
        { 0x198, 0x05a, 0x198, 0x05a },
        //      { 011110000, 010011001, 011110000, 010011001 },
        { 0x0f0, 0x099, 0x0f0, 0x099 },
        //      { 000111010, 010110010, 010111000, 010011010 }};
        { 0x03a, 0x0b2, 0x0b8, 0x09a }
    };

    int Data = nData[m_nID][m_nShape];

    //有 Cell 为 1，无 Cell 为 0
    m_nData[0][0] = (Data & 0x100)>>8;
    m_nData[0][1] = (Data & 0x80)>>7;
    m_nData[0][2] = (Data & 0x40)>>6;
    m_nData[1][0] = (Data & 0x20)>>5;
    m_nData[1][1] = (Data & 0x10)>>4;
    m_nData[1][2] = (Data & 0x8)>>3;
    m_nData[2][0] = (Data & 0x4)>>2;
    m_nData[2][1] = (Data & 0x2)>>1;
    m_nData[2][2] = (Data & 0x1)>>0;
}
```

CDiamondStatic 类继承于 CStatic 类，是游戏平台的窗口类，是游戏呈现的载体，类声明如下：

```
class CDiamondStatic : public CStatic
{
public:
    //      构造函数：
```

```
        //        初始化随机种子；
        //        初始化区域数据；
        //        新建下一个方块；
        //        设置游戏状态；
    CDiamondStatic();
  virtual ~CDiamondStatic();
  public:
    //开始游戏
    void            BeginGame();
//暂停游戏
    void            PauseGame();
    //继续游戏
    void            ResumeGame();
    //把方块放到游戏界面中，开始第一步
    void            BeginStep( CDC *pDC );
    //在界面中的方块的下一步动作，下落或结束下落
    void            StepShape();
    //结束下落
    void            EndStep( CDC *pDC );
    //结束游戏
    void            EndGame();
    //方块方向改变
    bool            ChangeShape( int nMode );
    //得到当前游戏状态（开始、暂停、结束）
    int             GetState();
    //加速游戏
    void            SpeedUp();
    //减速游戏
    void            SpeedDown();

    static DWORD WINAPI Robot(LPVOID lpParameter);
  private:
    int             m_nPoints;          //得分
    int             m_nElapse;          //消行
    int             m_nState;           //当前游戏状态
```

```
    CDiamondShape    m_Shape;          //当前操作方块
    CDiamondShape    m_ShapeNext;      //下一个方块
    bool             m_Data[ROW][COL];
    COLORREF         m_crData[ROW][COL];
  private:
    //画带有阴影的文字
    void    DrawShadowText( CDC *pDC, CRect rt, CString strText );
    //画当前局面中以固定的格子
    void    DrawFixLine( CDC *pDC, int nRow, bool bErase = false );
    //越界检验
    bool    ShapeValid( CDiamondShape sp );
    //慢行时消行
    void    ClearLine( int nTop, int nBotton );
    //消行时被消行的闪烁效果
    void    FlashLine( CDC *pDC, CUIntArray &anRow );
  protected:
    //{{AFX_MSG(CDiamondStatic)
    afx_msg void OnPaint();
    afx_msg void OnTimer(UINT nIDEvent);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
  };
```

这其中，对于路点来说又有：

```
// 路点:元素有目的坐标,先横移还是先纵移动,移动的块形状
// 默认都是先横移后纵移动
//移动前需要先调整形状
//如果希望移动后调整形状的,必须写两个路点,第一套是移动到位置不变形状,第二个路点是变形状,目标地点相同.
public struct WayPoint
{
    public XYPair TheAimPoint;//目标点的 xy 坐标
    public bool FirstHorizontal;//首先横向移动还是纵向移动（默认都是横向）
    public SetOfTetris TheAimSet;//当前的方块类型（包含转动到的样
```

子，这个对象中横条和竖条是两个对象）

```
        public WayPoint(XYPair theAimPoint, SetOfTetris theAimSet)
        {
            TheAimPoint = theAimPoint;
            TheAimSet = theAimSet;
            FirstHorizontal = true;
        }
        public WayPoint(XYPair theAimPoint, SetOfTetris theAimSet, bool
firstHorizontal)
            : this(theAimPoint, theAimSet)
        {
            FirstHorizontal = firstHorizontal;
        }
    }
```

获取原时钟中断向量方法如下：

```
void interrupt(*oldtimer)();//定义保存原中断向量的指针变量
oldtimer=getvect(0x1c);//截获原时钟中断向量
```

getvect()可以根据表示中断号的参数来获取相应的中断处理函数的入口地址。0x1c 就是时钟中断的中断号码。

定义新的中断服务函数：

```
void interrupt newtimer()//新的时钟中断服务程序
{
  Int leveltemp,time=11;//若增大 time 的值，则整体上难度偏低
  Leveltemp = time-level;//随着难度等级的提高，方块会下落更快
Count++; //记录时钟中断次数的计数器
If(count>=leveltemp)
{
  Addtobuffer(DOWN);//向键盘缓冲区中放入一个下移指令
  Count=0;
}
(*oldtimer)();//接着执行原来的中断服务程序
}
```

Interrupt 是一个保留字，表明函数 newtimer()是专用于中断处理的函数。这个函数里使用了一个全局的计数器变量 count，如果函数每执行一次则此变量就会加 1。当计数变量 count 超过规定的次数 levertemp，便向键盘缓冲区中

放入一个 DOWN 指令，对自然下落的处理和玩家按下键盘后的处理可以等同看待，addtobuffer()是一个自定义的函数。之后不可或缺的一步便是执行原来的中断处理程序。

//AI 逻辑内 计算当前逻辑存储的场景的某一个位置是否可以放一个具体的块

//参数 theArray：当前逻辑存储的场景

//参数 position:要放的位置的左上角坐标

//参数 setOfTetris：要放的图形对象（这个没有再做优化，其实也可以做）

```
private bool insidePositionIsPermit(List<int> theArray, XYPair position, SetOfTetris toAddSet)
{
    for (int i = 0; i < 4; i++)
    {
        try
        {
            if (toAddSet.EachRow[i] == 0) continue;
            if (i + position.Y >= rowsCount) return false;
            //之所以用 0x7fc00fff 不用 0xffc00fff 是考虑当前是有符号整数，否则会报一个警告，大概意思是：符号位进行位运算不太好。
            if (((theArray[i + position.Y] << 12 | 0x7fc00fff) & (toAddSet.EachRow[i] << (columnsCount + 8 - position.X))) != 0) return false;
        }
        catch //注意，把例外写在外面和写在里面 效率差了很多很多倍
        {
            return false;
        }
    }
    return true;
}
```

CRobot 类是控制 AI 的类，负责各算法类于游戏平台的连接，算法类通过 CRobot 类控制游戏，并获取当前局面信息，类声明如下：

```
class CRobot
{
public:
  //增加新算法
```

```
        static void AddStrategy(CStrategy * pStrategy);
        //初始化算法
        static void Initialize();
        //为当前算法取名字
        static void SetCurrentStrategyByName (string strategyName);
        //得到当前算法
        static string GetCurrentStrategyName ();
        //得到当前算法对象指针
        static CStrategy * GetCurrentStrategy();
        //选择下一个算法
        static void SelectNextStrategy ();
        //得到下一步的最佳走法
        static void GetBestMoveOncePerPiece (
            STBoard & currentBoard, // 当前局面
            STPiece & currentPiece, // 当前方块
            int nextPieceFlag, // 0 == 没有下一个方块了
            int nextPieceShape, // 0 == 此方块没有下一个方向了
            int & bestRotationDelta, // 旋转方向{0,1,2,3}
            int & bestTranslationDelta // 位移{...,-2,-1,0,1,2,...});
    private:
      //算法列表
        static vector<CStrategy *> mListpSTStrategy;
        //当前算法名字
        static string mCurrentStrategyName;
    };
        /* * * * * * * * * * * * * * * * * * * * * * *
        * 内部函数：判断当前方块是否已到底，并且消行等相关的工作
        * * * * * * * * * * * * * * * * * * * * * * */
        void CSkyblue_RectView::IsBottom()
        {
         //到底有两种概念：1 是已到底部，2 是下面碰到了另外的方块
         int x1,x2,x3,x4;
         int x,xx,yy,i;

         x1 = ActiveStatus[0][0];
```

```
        x2 = ActiveStatus[1][0];
        x3 = ActiveStatus[2][0];
        x4 = ActiveStatus[3][0];


        //是否为底部的判断
        //1。到达游戏区域的底部
        //2。与接触面正下方的小方块区域为被占用状态
        if  (x1>=m_iRow-1  ||  x2>=m_iRow-1  ||  x3>=m_iRow-1  ||
x4>=m_iRow-1)
            m_isBottom = TRUE;
        else
        {
            for (i=0;i<4;i++)
            {
                if (InterFace[m_currentRect][i] > -1)
                {//取当前下坠物有接触面的方块

                    //获取有接触面的小方块的编号
                    x=InterFace[m_currentRect][i];
                    //根据编号获取 ActiveStatus 中该小方块的整下方的坐
标
                    xx=ActiveStatus[x][0]+1;
                    yy=ActiveStatus[x][1];
                    //判断该接触面整下方的小方块区域是否为被占用状态
                    if (GameStatus[xx][yy]==MAP_STATE_NOT_EMPTY)
                        m_isBottom = TRUE;
                }
            }
        }

        BOOL m_bIsSucced;
        int k,j;
        int m_iMuch=0; //本次销掉的行数
```

```
//计分规则：一次销掉一行，加 100 分，一次销掉两行，加 400 分，三行，900 分
//例如销掉 x 行，则分数为：x*(x*100)

if (m_isBottom)
{
    //判断是否已得分
    for (i=0;i<m_iRow;i++)
    {
        m_bIsSucced = TRUE;
        for (j=0;j<m_iCol;j++)
            if (GameStatus[i][j]==MAP_STATE_EMPTY)
                m_bIsSucced = FALSE;

        //如果得分，则销掉此行
        if (m_bIsSucced)
        {
            for (k=i;k>0;k--)
                for (j=0;j<m_iCol;j++)
                    GameStatus[k][j] = GameStatus[k-1][j];
            //第 1 行清零
            for (j=0;j<m_iCol;j++)
                GameStatus[0][j]=MAP_STATE_EMPTY;

            m_iMuch += 1;
        }
    }

    if (m_iMuch>0)
    {
        m_iPerformance += 1;
        //刷新游戏区域
        CRect rect1(m_iStartY, m_iStartX, m_iStartY+300, m_iStartX+360);
        //InvalidateRect(&rect1);
```

```
                //刷新分数区域
                CRect rect2(m_iStartY+320, m_iStartX+180, m_iStartY+440,
m_iStartX+200);
                //InvalidateRect(&rect2);
                Invalidate(FALSE);
            }
        }
    }
```