#### 致谢

首先感谢刘贵全老师，能够在您的指导下完成我的毕业论文是我莫大的荣幸，您在工作中对我的指导，您对数学敏锐的直觉、您的人格魅力、以及您对学术和人生的执著严谨的态度都给我留下了深刻的印象，再次向您表示最诚挚的感谢和敬意。

感谢实验室项天学长一直以来的支持与鼓励，让我在每个季节都能感受到春天般的温暖。

感谢楚凡科技提供UML绘图工具的试用，使我能够顺利写完程序框架部分。

同时，还要感谢李瑞、杨云帆、张曦文、纪元、彭晗和代映秋同学，你们和我一起学习，分享快乐，共同走过了一段美好的时光。

最后，深深地感谢我的父母，你们一直无条件的支持我，相信我，让我能够一直没有负担地去追求自己的梦想。

#### 内容摘要

棋类游戏因其独特的趣味性和复杂性一直以来都是人们锻炼思维和修身养性的有力工具之一。同时它也是人工智能算法研究的发源，更是检验新理论的试验场。为了更好地利用计算机资源，调动普通人学习机器智能的积极性，方便编程者快速搭建智能博弈平台，本文介绍了一个通用博弈平台的搭建过程，通过此方式向开发者提供一个方便统一的编程接口。这一方面更有效地利用计算资源，方便日后智能游戏的内核统一地随着理论升级；另一方面整合编程资源，将开发者的关注点从开发智能系统转移到扩充现有智能平台上来，有助于为现有智能系统快速引入新理论、新思想。

关键字:

人工智能 博弈系统 可配置 多人敌对搜索 象棋 跳棋 五子棋

#### Abstract

Because of its unique interesting characteristics and complexity, Chess Games has always been one of the useful tools for people to exercise thinking power and do self-cultivation. It is also the origin of artificial intelligence algorithm, further, the proving ground of new theories. In order to make better use of computer resources, mobilize the enthusiasm of ordinary people learning machine intelligence, and facilitate programmers to quickly build intelligent game platforms, this article describes the process of building a universal game platform, providing a package of convenient unified programming interfaces to developers in this way. This can on the one hand make the use of computing resources more efficient, facilitating future upgrade of the kernel uniformly as to growing intelligent game theory; on the other hand integrate programming resources, making the developer's focus shifted from the development of intelligent systems to expanding the existing intelligence platform, thus can do good to rapid introduction of new theories and new ideas to existing intelligent systems.

Keywords：

Artificial Intelligence, Intelligence System, Configurable, Adversarial Search of Multi-Players, Chess, Checkers, Backgammon

#### 目录

[第1章 绪论 8](#_Toc377930887)

[1.1 相关背景介绍 8](#_Toc377930888)

[1.1.1 通用编程平台的背景介绍 8](#_Toc377930889)

[1.1.2 智能博弈系统的背景介绍 8](#_Toc377930890)

[1.2 可配置智能博弈系统的作用和意义 9](#_Toc377930891)

[1.3 文章的后续框架 10](#_Toc377930892)

[第2章 博弈系统的特征提取和博弈平台的编程框架设计 11](#_Toc377930893)

[2.1 博弈系统的特征提取 11](#_Toc377930894)

[2.1.1 博弈的分类及其各自特点分析 11](#_Toc377930895)

[2.1.2 空间离散回合制棋类游戏的智能博弈分析 13](#_Toc377930896)

[2.2 博弈平台编程框架设计 14](#_Toc377930897)

[2.2.1 棋类游戏的各部件属性模型 14](#_Toc377930898)

[2.2.2 棋类游戏的规则模型 16](#_Toc377930899)

[2.2.3 棋类游戏走法的状态机模型 17](#_Toc377930900)

[2.2.4 棋类游戏的棋盘模型 20](#_Toc377930901)

[2.3 棋类博弈平台的面向对象分析-UML辅助 22](#_Toc377930902)

[2.3.1 辅助类和接口类设计 22](#_Toc377930903)

[2.3.2 基础类设计 24](#_Toc377930904)

[2.3.3 部件类设计 25](#_Toc377930905)

[2.3.4 全局设计，类的继承关系和依赖关系 30](#_Toc377930906)

[2.4 本章小结 30](#_Toc377930907)

[第3章 多角色敌对搜索内核的构造 31](#_Toc377930908)

[3.1 状态空间搜索算法应用于人工智能的不同形式及其特点 31](#_Toc377930909)

[3.1.1 状态空间搜索[2] 31](#_Toc377930910)

[3.2 敌对搜索简介 32](#_Toc377930911)

[3.2.1 最小-最大搜索 32](#_Toc377930912)

[3.2.2 alpha-beta搜索 33](#_Toc377930913)

[3.3 敌对搜索的多玩家扩展模型 34](#_Toc377930914)

[3.4 本章小结 36](#_Toc377930915)

[第4章 跨平台的GUI接口设计及其MFC实现 37](#_Toc377930916)

[4.1 跨平台GUI接口设计 37](#_Toc377930917)

[4.1.1 接口跨平台部分简介 37](#_Toc377930918)

[4.1.2 有人类玩家时着法生成器的扩充 38](#_Toc377930919)

[4.2 可配置的GUI棋盘模型 38](#_Toc377930920)

[4.3 GUI接口的MFC实现 39](#_Toc377930921)

[4.4 本章小结 39](#_Toc377930922)

[第5章 象棋、五子棋、跳棋的简单配置实践 40](#_Toc377930923)

[5.1 象棋规则和界面的配置 40](#_Toc377930924)

[5.1.1 规则配置 40](#_Toc377930925)

[5.1.2 界面配置及运行结果 41](#_Toc377930926)

[5.2 五子棋规则和界面的配置 42](#_Toc377930927)

[5.2.1 规则配置 42](#_Toc377930928)

[5.2.2 界面配置及运行结果 42](#_Toc377930929)

[5.3 六人跳棋规则和界面的配置 44](#_Toc377930930)

[5.3.1 规则配置 44](#_Toc377930931)

[5.3.2 界面配置及运行结果 45](#_Toc377930932)

[5.4 本章小结 46](#_Toc377930933)

[第6章 论文总结 47](#_Toc377930934)

#### 图目录

[图 1.1 关键词“博弈系统”的部分百度搜索结果 9](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930935)

[图 2.1国际象棋走法生成图 18](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930936)

[图 2.2跳棋走法生成图 18](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930937)

[图 2.3操作与动作的并行包含关系 19](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930938)

[图 2.4跳棋棋盘范围的定义 21](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930939)

[图 2.5 Windows系统下的GUI类图 22](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930940)

[图 2.6 CIGObejct类图 23](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930941)

[图 2.7 CIGRuleConfig类图 23](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930942)

[图 2.8 Stack模板类图（继承自Array类） 24](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930943)

[图 2.9 Array模板类图 24](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930944)

[图 2.10 Chessman类图 25](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930945)

[图 2.11 Player类图 25](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930946)

[图 2.12 Chessboard类图 26](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930947)

[图 2.13 Operation类图 27](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930948)

[图 2.14 ChessmanIndex类图 27](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930949)

[图 2.15 ChessmanIndexBoard类图 27](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930950)

[图 2.16 MotionGenerator类图 28](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930951)

[图 2.17 GraphSearchEngine类图 29](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930952)

[图 2.18 Game类图 29](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930953)

[图 2.19 所有类图 30](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930954)

[图 3.1 搜索中的裁剪 34](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930955)

[图 4.1 GUI类成员函数 37](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930956)

[图 5.1象棋走法图 40](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930957)

[图 5.2象棋配置结果 41](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930958)

[图 5.3五子棋走法图 42](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930959)

[图 5.4五子棋局面 43](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930960)

[图 5.5跳棋走法图 44](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930961)

[图 5.6 计算跳棋的的路程 44](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930962)

[图 5.7跳棋配置结果 45](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc377930963)

#### 表目录

[表 5.1 五子棋局面评估表 42](#_Toc377930964)

#### 重要定义目录

[定义 2.1参与者、玩家、策略、博弈、游戏 11](#_Toc377930965)

[定义 2.2结果集合、偏好函数 11](#_Toc377930966)

[定义 2.3博弈分类 12](#_Toc377930967)

[定义 2.4博弈的规则 12](#_Toc377930968)

[定义 2.5属性 12](#_Toc377930969)

[定义 2.6棋牌类游戏 12](#_Toc377930970)

[定义 2.7位置 13](#_Toc377930971)

[定义 2.8棋盘、棋子 13](#_Toc377930972)

[定义 2.9棋盘属性、玩家属性、棋手 13](#_Toc377930973)

[定义 2.10局面 14](#_Toc377930974)

[定义 2.11着法 14](#_Toc377930975)

[定义 2.12动作 14](#_Toc377930976)

[定义 2.13操作 14](#_Toc377930977)

[定义 2.14走法 14](#_Toc377930978)

[定义 2.15走法生成图 17](#_Toc377930979)

[定义 2.16常义棋盘、基 20](#_Toc377930980)

[定义 2.17棋路、直棋路 21](#_Toc377930981)

[定义 3.1状态树 31](#_Toc377930982)

[定义 3.2评估函数 32](#_Toc377930983)

[定义 3.3对某局面的评价（评估） 33](#_Toc377930984)

# 绪论

## 相关背景介绍

### 通用编程平台的背景介绍

随着计算机理论与实践的不断发展，怀抱着各种目的而使用计算机的人们几乎都可以使用各种软硬件工具而满足自己的需求。但同时软硬件使用者们也在经受着各种平台不兼容、移植代价大的灾难。这种情况对开发者而言尤甚：

硬件厂商为了让大家关注他们的平台，通过建立一些新的编程模型来“帮助”并把可怜的程序员绑定在他们的平台上。这样就产生了各种各样的编程模型。善良的程序员要么被误导，要么被强迫使用这些新的编程模型和各种软件。于是，创造新软件的趣味变成各个新平台改写现有软件的沉闷工作。

——Tim Mattson英特尔首席工程师

《OpenCL异构计算（第二版）》，清华大学出版社

第二版推荐序

可以试想，当编程人员都在为平台和移植发愁，代码的开发周期将是漫长而了无生趣的。而被兼容性的问题分散了精力，开发者将会在本该全神贯注的领域花费更少的时间。

OpenGL、OpenCL等一些通用平台的出现，为编程人员提供了统一的编程模型，不同硬件系统之间的沟壑由平台内部填平，解放了一部分生产力，节省了很多社会资源。

### 智能博弈系统的背景介绍

在人工智能领域日渐成熟的今天，智能博弈系统并不是什么新鲜事。智能博弈系统的普遍性不必特地考察，仅用百度搜索即可知道：

但是据本人所知，在本论文之前没有一个可配置的博弈平台问世。

图 .1 关键词“博弈系统”的部分百度搜索结果

## 可配置智能博弈系统的作用和意义

随着人工智能领域的日渐发展，棋类游戏中的智能博弈系统已经数见不鲜。棋类游戏因其独特的趣味性和复杂性一直以来都是人们锻炼思维和修身养性的有力工具之一。同时棋类游戏也是人工智能算法研究的发源，更是检验新理论的试验场。历史雄辩地证明，当兴趣与需求结合，人类往往显示出巨大的创造力，这使得智能博弈作为人工智能的一大重要领域经久不衰。

然而就当前飞入寻常百姓家的智能棋类游戏而言，一方面编程者水平参差不齐，使得机器智能潜力并未完全发挥；另一方面，编程者需要为每一个独特的棋类游戏重新写智能算法，配置显示界面，这是对社会资源的一种浪费。为了更好地利用计算机资源，调动普通人学习机器智能的积极性，方便编程者快速搭建智能博弈平台，一个通用博弈平台可以向开发者提供一个方便统一的编程模型。这一方面更有效地利用计算资源，方便日后智能游戏的内核统一地随着理论升级，另一方面可以整合编程资源，将开发者的关注点从开发智能系统转移到扩充现有智能平台上来，有助于为现有智能系统快速引入新理论和新思想。

## 文章的后续框架

这篇论文的下面部分将按照这样的顺序进行编排：第二章将详细的介绍本博弈平台的设计框架；第三章将介绍本文采用的敌对搜索算法；第四章介绍MFC下GUI接口的跨平台设计与具体实现；第五章使用本平台实地配置象棋、跳棋和五子棋，并分析系统的功能和效率；最后一章将总结整个毕业论文的工作。

# 博弈系统的特征提取和博弈平台的编程框架设计

要设计一个通用的棋类博弈平台，首先要结合各种现有的具体博弈实例分析博弈的分类，并提取出棋类游戏的共同特征，然后以编程概念清晰、性能优越、扩展性强、鲁棒性强等为依据设计编程模型。本章就博弈平台的特征提取和编程框架设计这两个紧密联系的主题给出解决方案。

## 博弈系统的特征提取

### 博弈的分类及其各自特点分析

在此我们不就博弈的数学定义进行深入探讨，仅取定义如下：[1]

定义 2.1参与者、玩家、策略、博弈、游戏

设定有N个“**参与者**”（后文也常称“**玩家**”）作为博弈的主体，对于每一个“参与者”，都有一个给定的“**策略**”集合. **博弈**（后文也称“**游戏**”）是一个函数， 定义为:

其中R为实数集，“×”表示卡氏积。

也就是说，如果我们知道了参与者的策略集合是什么，那么就可以有一个实数值与之对应。

定义 .2结果集合、偏好函数

我们可以把上面的方程拆成两个方程来进一步把它一般化。一个方程是正则形式（Normal form game）的参与者方程，描述策略决定结果的方式（**规则**的一部分，下文将定义“规则”）。另外一个方程描写参与者对于结果（outcome）集合的偏好（preference）。也就是：

这里是游戏（博弈）的**结果集合**（outcome set）。对于每一个参与者都有一个**偏好函数**（preference function）

上文的第二种定义鲜明体现了博弈的特征：玩家可能采取的动作由规则限制，而在规则的限制下，每个玩家都想要得到对自己最有利的结果。（注意这里并不要求像棋类游戏一样，每个玩家的规则和目标都是对称的）

博弈的分类根据不同的基准也有不同的分类。[1]

定义 2.3博弈分类

在当事人之间有没有一个具有约束力的协议意义上，博弈主要可以分为**合作博弈**和**非合作博弈**。

扩展博弈的定义，将博弈“复合”化：每一**阶段**（称为“**元博弈**”）每名玩家均在策略集合中选择策略，并产生单次元博弈的结果（称“**部分结果**”）。在单次结果满足一定条件时，整个博弈才结束。偏好函数也是对部分结果定义的。其实部分结果和最终结果的区别仅仅在于在达到最终结果时停止进入下一元博弈阶段。这称为**即时制博弈**。（将时间连续的博弈划分为细小的时间片，若不同玩家策略产生的结果冲突，由概率决定选择怎样的结果）

定义**回合制博弈**：每一阶段只有一名玩家的策略集合可能不是空集，（该阶段称为当前玩家的**回合**）而且回合按照玩家标号顺序地交换的即时制博弈。

按照参与人对其他参与人的了解程度分为**完全信息博弈**和**不完全信息博弈**。完全博弈是指在博弈过程中，每一位参与人对其他参与人的特征、策略空间及收益函数有准确的信息。如果参与人对其他参与人的特征、策略空间及收益函数信息了解的不够准确、或者不是对所有参与人的特征、策略空间及收益函数都有准确的准确信息，在这种情况下进行的博弈就是不完全信息博弈。

定义 .4博弈的规则

事先确定的影响产生策略集合和偏好函数的博弈特性。

定义 .5属性

为了方便讨论策略集合的生成，将玩家影响策略集合和偏好函数的关键状态分离出来，称其为**属性**。

定义 .6棋牌类游戏

* 对每个玩家而言，策略集合的产生规则绝对交换对称（常义棋）或概率上交换对称（常义牌）；
* 每个玩家的偏好函数交换对称且对特定结果的偏好函数值为2(获胜)，另一些结果为1(和局)，其余结果的偏好函数全为0(失败)；
* 是回合制博弈。

注：

1. 各种体育赛事、各种常义棋牌类游戏等基本均属于此定义。
2. 两人游戏时，若结果对一人的偏好函数值为2，则要求对另一人的偏好函数值为0；若结果对一人的偏好函数值为1，则要求对另一人的偏好函数值也为1。但是对多人博弈，则没有此限制。

为了讨论方便，将棋牌类游戏**进一步分类**：策略集合与**位置**相关的棋牌类游戏称**棋类游戏**（如足球、象棋等常义棋类游戏），其余称**牌类游戏**（如常义牌类游戏、“石头剪子布”）。

定义 .7位置

其中“**位置**”定义为玩家所拥有的一个或多个向量属性。（往往在对“策略”集合的形成有影响时，位置才对博弈有作用）

在位置相关的博弈中，可以根据位置可取值集合的**可数性**分类为**空间离散的**和**空间连续的**。

### 空间离散回合制棋类游戏的智能博弈分析

定义 .8棋盘、棋子

空间离散棋类游戏要求方便地表示其位置及各位置间的关系，可以定义一个**棋盘**作为索引。棋盘中每一个位置属性抽象成一个**棋子**，定义为所有与单个位置相关的属性集合。

棋盘可以是无穷大，也可以有有限范围，也可以有周期性边界条件，也可以有任意的相邻关系，并不一定满足实数的运算法则。但所有可能的位置集合必须可数，否则不能表示棋盘局部的全部位置信息。

定义 .9棋盘属性、玩家属性、棋手

将与所有与全体玩家有关的影响策略集合的属性归为**棋盘属性**统一管理，将其余属性归为每个**玩家的属性**。

注：

1. 每个玩家的策略集合和偏好函数都是对称的，所以属性内容一致。
2. 与策略集合和偏好函数均无关的属性（如玩家名字，棋子颜色等）对博弈没有任何影响，在定义上放在了玩家上，但是后文在不影响规则的原则下根据方便进行转移。
3. 为讨论方便，后文将“空间离散回合制棋类游戏”简称为“棋类游戏”，有时将玩家称为**棋手**。

## 博弈平台编程框架设计

上文定义了棋类游戏，本节将继续细化各部件的属性模型。我们将逐渐脱离棋类游戏的数学定义，将重点放在框架设计和模型构造上。

首先声明，棋类游戏的形式千差万别，这里只提取一些必要属性和一些常见属性。

### 棋类游戏的各部件属性模型

为了讨论方便，定义几个名词：

定义 .10局面

整个游戏中影响任一玩家策略集合的属性集合。

定义 .11着法

每个回合玩家所有的行动。（后文也称行动）

定义 .12动作

玩家当前回合内可以采取的单个策略。

定义 .13操作

泛指玩家影响局面的一类动作。

定义 .14走法

泛指玩家的着法，后文用于描述不同操作的顺序关系。

对于着法、动作、走法和操作的区别，在*2.2.3棋类游戏走法的状态机模型*中会有具体说明。其核心思想是：着法包含几个操作，而每个操作都以动作的形式具体体现。

下面描述棋类游戏各部件属性构成及必要解释：

* 棋盘的通用属性：
* 当前回合数
* 当前回合玩家（后文称“**当前玩家**”）
* 当前获胜玩家和当前失败玩家（在多人博弈中记录已经确定获胜或失败的玩家，方便确定何时到达博弈最终结果）
* 当前各棋子的位置
* 为了编程方便，加入当前并未在棋盘（玩家没有此向量属性）但实际上仍然存在的棋子（比如为了走一步而刚刚被拿起的棋子）
* 棋盘可能的扩展属性举例：
* 特殊格点及其作用（如军棋中的大本营，三国志游戏当中的城池或堤坝等）
* 当前季节（比如模拟战争游戏时影响战略实施和军队调度）
* 当前游戏总时间
* 历史着法（可能规则允许悔棋，也用于对如象棋中“不变作和”、“长将判负”等规则给予支持）
* 当前禁止着法（是规则中对每个玩家有同等约束力的限制，如围棋中“打劫”）
* 棋手间的结盟关系
* 棋手通用属性：
* 当前拥有的棋子
* 棋手可能的扩展属性举例：
* 个人已用时间
* 生命力（如游戏王等游戏中游戏中局部的失败或胜利能反映到玩家的生命力上）
* 棋手名字
* 棋手代表颜色
* 占据的格点（例如围棋）
* 当前行动力（有的规则可能限制棋手的总行动能力而并不限制具体动作）
* 棋子通用属性:
* 当前坐标
* 所属棋手
* 棋子可能的扩展属性举例:
* 名字
* 颜色（可能每个棋子有玩家颜色之外自己的颜色属性）
* 生命值（棋子可能拥有独立于玩家生命值之外的自己的生命值）
* 行动力（可能拥有独立于玩家的行动力）
* 特定行动消耗的行动力值

### 棋类游戏的规则模型

规则用来限制走法或者判断输赢（其实是决定了偏好函数在某些特殊局面下的取值）。其逻辑模型根据对各部件属性的限制分类如下：

* 全局通用规则：
* 初始局面（包括初始棋子位置等各属性的初始值）
* 玩家轮流顺序（玩家的排序）
* 棋盘范围
* 特殊的空间关系（如象棋九宫格、河界等）
* 使局面陷入循环的惩罚办法
* 局面随着回合数、游戏时间、玩家之间势力评估的变化（比如每过一年，玩家就老一岁，但是会有一年的收入等）
* 可能的全局扩展规则：
* 回合限制
* 总时间限制、玩家时间限制
* 玩家相关的通用规则：
* 每回合产生着法的规定
* 可能的玩家相关扩展规则：
* 行动力增长率（见上文玩家的行动力）
* 生命值增长率（见上文玩家的生命值）
* 棋子相关的通用规则：
* 本类棋子着法产生的具体规则（用于着法生成器产生满足规则的着法和在交互界面判断当前玩家的操作是否满足规则）
* 被拿起的条件和效果
* 被放下的条件和效果
* 吃子的条件和效果
* 被吃的条件和效果
* 可能的棋子相关扩展规则：
* 各动作消耗的行动力
* 棋子的行动力增长率
* 棋子的生命值增长率

### 棋类游戏走法的状态机模型

本段讨论走法规则的具体模型，也给出上文所述“着法（Move）、动作（Motion）、走法（Action）、操作（Operation）”四个概念的具体关系。

一般地说，走法是着法的模板，操作是动作的模板。也就是说一个着法是一个具体的走法，一个动作是一个具体的操作。

定义 .15走法生成图

用以操作为节点的有向图表示走法，用走法规定每一回合玩家做能采取的着法的操作构成，既方便制定规则，又方便统一不同棋类游戏着法的产生方式。例如：

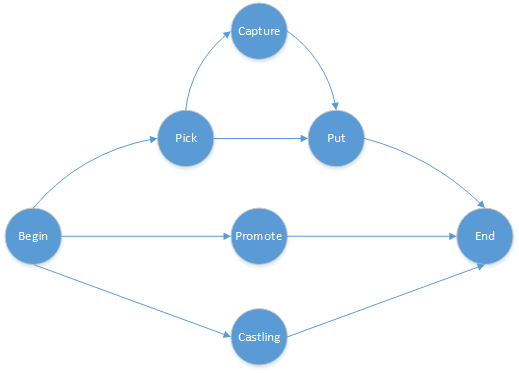
1. 国际象棋中，“一步棋”（一个着法）可以被如下的操作图生成：

图 .1国际象棋走法生成图

注：

其中，棋子的一次移动是由pick和put两个操作组成的，在put之前，可以通过capture吃掉即将到达位置的棋子。

promote指升变，castling指王车易位。

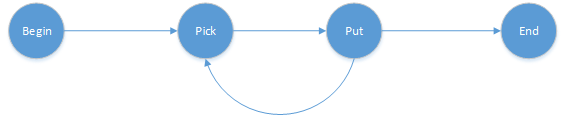
1. 跳棋中，“一步棋”（一个着法）可以被如下的操作图生成：

图 .2跳棋走法生成图

当我们要产生所有可能的着法时，需要深度优先搜索整个操作图，每次搜索到end节点，就将路径保存起来。但是应该注意，我们要保存的“路径”是动作的路径，而非操作的路径，而操作路径和动作路径的关系是每个操作路径上可能并行地包含多个动作路径，也可能有一些动作路径在操作路径里中断（搜索到该操作节点时检查发现没有合法的动作可以执行，则应该回溯）。

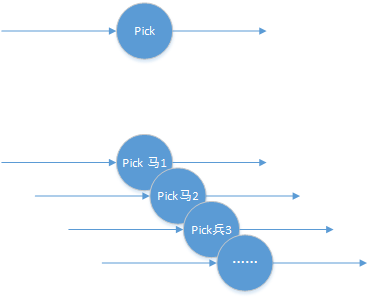
例如，实际由搜索生成的动作图的一部分举例如下：

图 .3操作与动作的并行包含关系

即，一个pick操作里可能包含很多具体的动作。

在搜索动作图时采用深度优先搜索，能完全地扫过走法生成图中所有并行的操作，而且采用深度优先搜索可以自然地体现走法的一般性与规则的特殊性的辩证关系，可以实现**规则的分层次管理**。

注：“规则的分层次管理”指所有棋子规则放在棋子类的函数内，所有棋手规则放在棋手类的函数内，而全局规则放在棋盘类的函数内。（详见2.3 棋类博弈平台的面向对象分析-UML辅助）如马1已经在四个方向均被“蹩马腿”，但是在搜索操作图生成走法的过程中，要先pick马1，然后在put或capture的时候调用棋子的onCapture或onPut回调函数，发现蹩马腿，onCapture或onPut回调函数返回false，知前方探测失败，此时可以退回pick，发现pick除了capture和put外并无后继节点，则返回pick的上一级——begin，发现已经搜索完马1的所有可能，则退出对马1 的搜索。

另一方面，每当搜索到end，知道一条路径顺利搜索完毕，保存当前动作路径，返回上一级继续后面的搜索。

整个搜索过程可以用“**运行栈**”和“**记录栈**”管理，“运行栈”是“横向”的，对应本操作包含的平行动作，“记录栈”是“纵向”的，对应已经探测过的操作路径和当前动作路径，以便在操作路径上探测到end节点时保存整个动作路径。

上面仅仅是形象的解释，严格的算法描述如下：

1. Begin作为操作节点入操作记录栈，当前记录栈清空，着法记录缓冲清空，开始新一层的搜索。非begin节点将被认为是一个错误。
2. 当开始新一层的搜索，若操作记录栈顶为end，则保存当前动作记录栈, 操作记录栈弹栈，返回上层搜索的f)；否则，操作和动作运行栈初始化，若操作记录栈非空，取定操作运行栈顶为当前操作，生成当前操作的后继操作，入本层搜索的操作运行栈。
3. 取定操作运行栈顶为当前操作。
4. 以当前局面为准，生成当前操作所有包含的动作，入本层动作运行栈。
5. 若当前动作运行栈非空，取定栈顶的元素为当前动作，入动作记录栈，棋盘进行当前动作指定的操作，若成功则跳转到b)，开始深一层的搜索，否则跳转到e)（也就是本步骤），继续搜索当前操作的其余动作。
6. 运行到此说明当前动作运行栈为空, 说明当前操作没有更多可用动作。若当前操作运行栈非空，操作运行栈弹栈，跳转到c)，继续当前层操作的搜索，否则当前操作运行栈空，说明上层搜索的所有后继节点已经搜索完毕，操作记录栈弹栈，返回上层搜索的f)。否则当前动作运行栈非空，棋盘恢复到深层搜索之前的状态，跳转到e)继续当前操作的搜索。

注：本算法适合用递归函数而不是循环做，可以借助程序的局部变量管理各层搜索的运行栈，令程序的运行栈结构与操作图的结构相对应

### 棋类游戏的棋盘模型

上段介绍了一个走法的生成，本段介绍常义棋盘模型的构造。

定义 .16常义棋盘、基

这里“常义棋盘”指棋盘上的点没有特殊的相邻关系，棋盘上的点集完全与一个二维整数向量集合的子集在平面上相邻关系的意义上同构。即棋盘中任意两点之间的可达路径均可落在用两簇平行且间距相同的直线或线段画出的格子的边界上。直线或线段的方向称为棋盘的**基**。

定义 .17棋路、直棋路

棋子所能行动的方向，棋路的方向并不一定都是基的方向，也不一定是直线。特别将直线或线段的贯通整个棋盘的棋路称为**直棋路**。

以上定义主要是为了方便棋盘的配置。我们可以只规定棋盘的范围、基的夹角、直棋路的方向和一些附加棋路，就能得到几乎所有的常义棋盘的逻辑结构和显示方式。

如五子棋，棋路两条，夹角90°，范围是一个正方形。

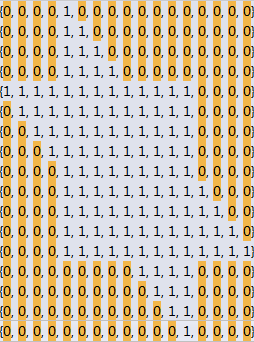
又如跳棋，基的夹角120°，直棋路有三条，两个分别为两个基的方向，另一个为两个基的向量和的方向。范围为下图矩阵中元素为1的地方，这样既实现了棋盘范围的任意配置，又实现了跳棋可以行走的三个方向。

图 .4跳棋棋盘范围的定义

但对于象棋九宫格等“局部棋路”的情形，应该特别规定，不能作为直棋路。

这里特别说明，本博弈系统仅支持有限大小的棋盘。而且为了方便快速判断一个坐标是否在棋盘上，棋盘的长和宽应均为2的某次幂。因为这样可以快速地用位与运算判断坐标值是否超过了棋盘范围。

## 棋类博弈平台的面向对象分析-UML辅助

上文已经介绍了博弈平台模型的设计框架，本节继续细化这一设计，向面向对象编程靠拢，规定一些必要的函数，在UML工具里画出类图。

为了程序的运行效率并兼顾可扩展性和代码的可维护性，采用C++作为编程语言，进而下文采用C++的术语而非UML术语来描述代码的构造。

### 辅助类和接口类设计

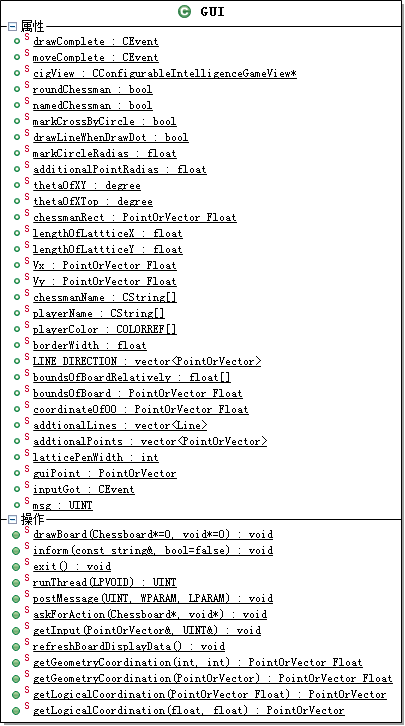
我们需要一个界面，为了使系统有较好的可移植性，独立出一个GUI类作为接口，而所有需要调用窗口显示功能的系统内部组件均调用GUI类的静态函数，使得移植时仅需要改动GUI类。

图 .5 Windows系统下的GUI类图

注：在第4章跨平台的GUI接口设计及其MFC实现中会有具体解释。

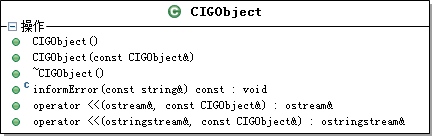
我们需要一个统一的报错机制，因此可以定义一个棋盘、棋手、棋子等类的共同基类CIGObject，并实现其informError虚函数。

图 .6 CIGObejct类图

为了能在报错时输出当前错误发出者的关键信息，可以覆盖informError函数，也可以均不覆盖informError函数，而各自实现operator<<虚函数并且在CIGObject类中的informError中调用operator<<函数。

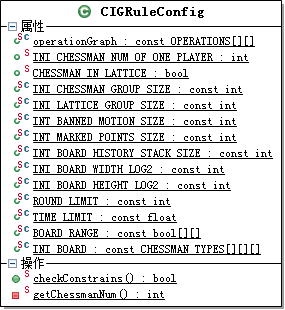
我们需要统一管理游戏的规则，引入CIGConfig类，定义相关的静态函数和静态变量。

图 .7 CIGRuleConfig类图

### 基础类设计

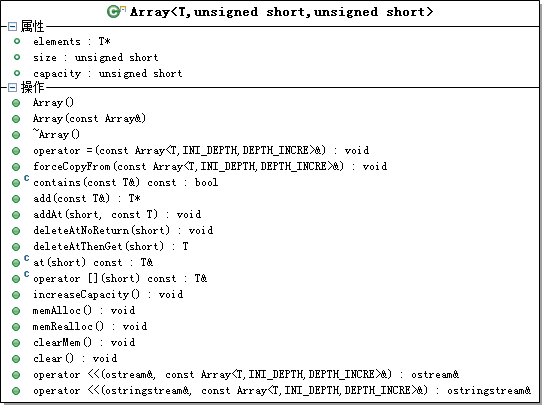
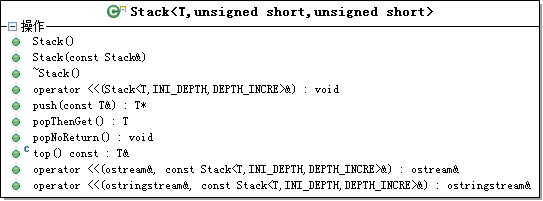
我们需要管理局面的信息，应该使用一些容器类。但是STL容器的功能不够强大，而且对于大容量数组的初始化需要依次调用默认构造函数，而很多情况下，仅需将对应存储空间memset即可正确初始化，所以自己构造一个专用的Array模板类和Stack模板类，实现元素标准的构造、析构、复制和快速的赋初值、删除、复制，并根据不同元素类型在调用时选择不同的函数集。

图 .8 Stack模板类图（继承自Array类）

图 .9 Array模板类图

### 部件类设计

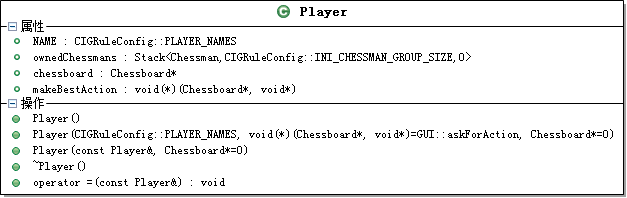
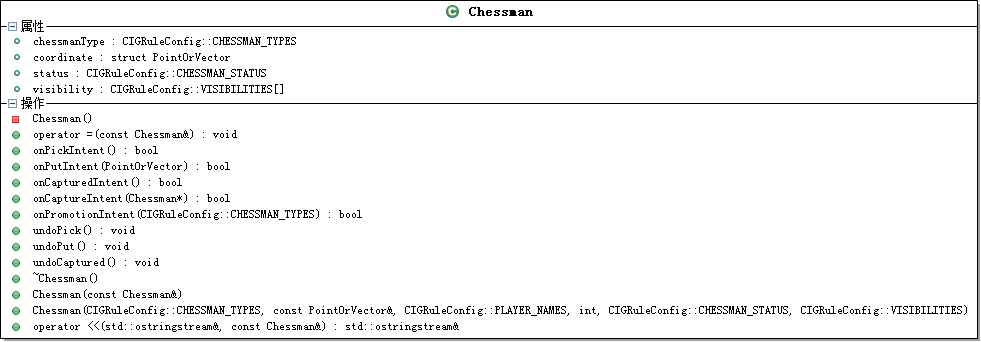
我们需要一个Chessboard类来管理棋盘属性和全局规则，一个Player类管理玩家相关属性和玩家相关规则，一个Chessman类管理棋子相关属性和棋子相关规则，一个MotionGenerator类产生当前所有的走法，一个GraphSearchEngine类包装状态空间搜索所需要的变量和函数，一个Game类将整个局面归拢。

图 .10 Chessman类图

图 .11 Player类图

* 对于Chessboard类：

为了对动态变化的棋子正确地寻址，应该记录棋子所属的玩家和在玩家内部的序号。棋盘存储的是整个序号的数组。

为了实现人工智能，需要对每回合的元博弈设置偏好函数，体现在evaluations数组上。具体做法将在第3章多角色敌对搜索内核的构造中进行介绍。

为了实现规则，在Chessboard类中设置onXXIntent系列函数，来判断是否可以进行某动作，并且内部调用相关棋子的onXXIntent函数，实现规则的嵌套。

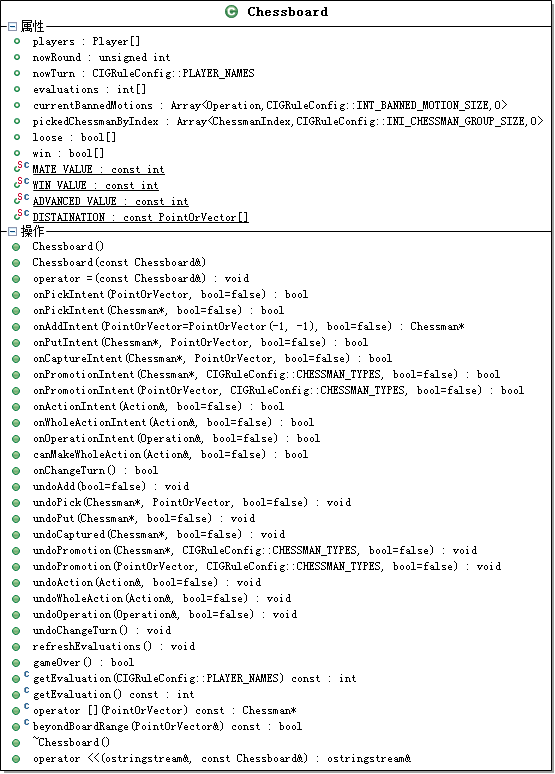
为了实现撤销一个着法，在Chessboard类中设置undoXXIntent系列函数，和undoAction函数。为了实现回合的轮换，在Chessboard类中设置onChangeTurn和undoChangeTurn函数。

图 .12 Chessboard类图

其中DISTINATION是一个辅助估值的数组，比如可以在跳棋中通过计算棋子与目的地的距离得到估值。

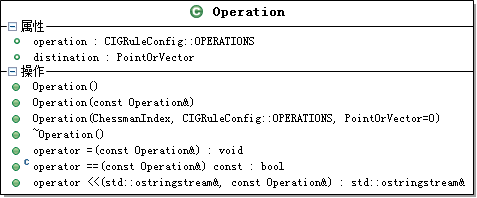
* 为了完全表示一个动作，引入一个Operation类。

图 .13 Operation类图

其中PointOrVector是一个整数2维向量类。

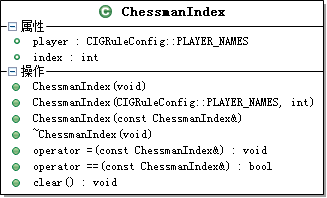
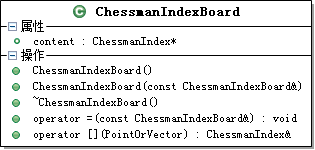
Chessboard类有一个实例成员chessmanIndexBoard，储存棋子的索引

图 .14 ChessmanIndex类图

图 .15 ChessmanIndexBoard类图

。

* 对于MotionGenerator类：

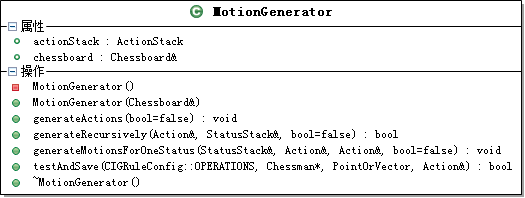
拥有一个棋盘引用作为当前局面。当前局面在构造函数中作为参数传递并禁止默认构造。

图 .16 MotionGenerator类图

类外部应该调用generateActions产生当前所有可能着法的数组。其bool型参数表示是否由鼠标输入着法。参见4.1.2有人类玩家时着法生成器的扩充。参数默认值为false。

generateRecursively函数即每一层的搜索函数，采用递归的方式产生各层动作并在每次到达END层时保存到成员变量actionStack中。参见2.2.3棋类游戏走法的状态机模型。

generateMotionsForOneStatus函数在每一层中由操作运行栈生成动作运行栈。而调用generateMotionsForOneStatus函数之前操作运行栈已经在generateRecursively函数中产生。参见2.2.3棋类游戏走法的状态机模型。

* 对于GraphSearchEngine类：

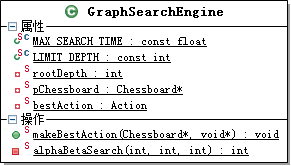
为了实现人工智能，将Player的makeBestAction成员函数变成函数指针，当玩家是机器时，将该指针赋值为GraphSearchEngine类的makeBestAction函数，否则玩家为人，指针赋值为GUI类的askForAction函数。

图 .17 GraphSearchEngine类图

GraphSearchEngine的具体内容解释见第3章多角色敌对搜索内核的构造。

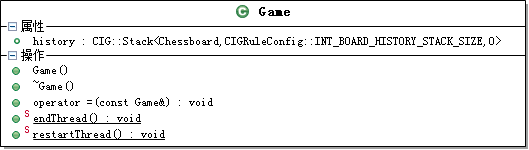
* 为了整合局面资源，加入Game类，现在只拥有历史棋盘和当前棋盘两个成员。

图 .18 Game类图

### 全局设计，类的继承关系和依赖关系

上文介绍了博弈系统的各个部件，本段将所有的类之间的结构用一张类图表示：

注：

* 所有标记均遵循UML2.0标准。
* 与MFC的关系以及MFC内部的类没有在图中标示完全。
* 泛化箭头颜色不同的原因是Array是模板类。
* Stack类是Array的子类，因为都是模板类，所以作图工具未给出泛化的箭头。

## 本章小结

图 2.19 所有类图

本章讲解了博弈系统的模型及其构造，体现了通用性和效率兼顾的设计思想，是整个系统的基础。

# 多角色敌对搜索内核的构造

本章将给出基于状态空间搜索的人工智能算法。这是本博弈系统采用的智能算法。

## 状态空间搜索算法应用于人工智能的不同形式及其特点

### 状态空间搜索[2]

定义 .1状态树

将每一个局面看作一个状态，将每一种当前可能的着法看作通向下一级状态的有向边，则从当前状态看过去，所有可能的着法各自产生一个状态分支，构成以当前状态为根节点的**状态树**。（不同路径上的重复节点不合并）我们搜索这个状态树，就可以得到通往目的地的最佳路径。

这里我们关心两个问题：

1. 搜索所期望达到的目标。
2. 搜索的方式（或称路线）。

根据搜索所期望达到的目标的不同，已经有不同的应用类型：

1. 单个玩家，使用最少的回合数，达到评估函数为2的局面（后文称**胜利局面**）：**盲目搜索或启发式搜索**。
2. 两个敌对的玩家，每一个都想达到自己的评估函数为2（而同时对方的评估函数为0）的局面（这是敌对搜索的**胜利局面**）：**敌对搜索**。

根据搜索方式不同，分为：

1. **广度优先搜索**，在状态树上广度优先搜索。总是能找到到达胜利局面的最短路径，但是消耗深度的指数级的内存。
2. **深度优先的回溯搜索**：状态树上有最大深度限制的深度优先搜索。通过回溯，也可以找到与最短获胜路径相差不大的获胜路径，占用内存仅仅为深度的线性函数。但是有可能在规定的深度上找不到胜利局面。
3. **迭代加深搜索**：既能满足线性存储要求，又能保证发现一个最小深度的目标节点，只是以牺牲常数倍的时间为代价。

## 敌对搜索简介

本节介绍一些关于两个敌对玩家博弈的状态空间搜索方法。

### 最小-最大搜索

对于一个复杂的回合制棋类游戏，在可以接受的时间内搜索出偏好函数不为0的走法不总是可能的。像象棋、围棋这样的棋类游戏，分支如此之多，能搜索的空间如此之大，除非在残局或官子阶段，以现有的计算水平，几乎不可能得到一定获胜的路线。但是我们想要在视野有限的前提下决定最可能对自己最有利的走法。

应该注意到如果沿用元博弈的定义，还是不能通过仅取离散的0,1,2三个值的偏好函数来描述当前局面下每个玩家的有利有害程度。这时需要扩充博弈的定义，加入**评估函数**：

定义 3.2评估函数

当局面并非任何一个人的胜利局面和失败局面，但是我们可以对局面有一个静态的评价，虽然不能保证评价高的一方一定会胜利，但是它代表了当前双方实力的相对大小，某种程度上反映了获胜的可能性。将对当前局面的评价量化，就得到评估函数。

这里要求评估函数满足一定的性质：

* 任意胜利局面的评估函数总是大于非胜利局面的评估函数；任意失败局面的评估函数总是小于非失败局面的评估函数。
* 对于棋牌类游戏，评估函数一定是关于棋手交换对称的。

为了使评估函数真正有效地起作用，还要求：

* 为了使程序能对当前局面快速地评估，评估函数关于一个动作的变化量仅是当前局面和当前动作的函数。
* 评估函数关于操作的波动总是较小，这样才能正确地引导搜索过程。

后文我们可以看到，评估函数会以两种形式出现：直接计算（称为**静态评估函数**），和根据后继玩家的评估函数推算。

定义 .3对某局面的评价（评估）

将当前局面带入当前活动的玩家的评估函数中，得到的评估值。

在两个敌对玩家的情况下，由于双方的评估函数轮换对称，我们总是可以将当前玩家的评估函数和对方玩家的评估函数相减规定为当前玩家新的评估函数，（可以根据定义验证结果仍然是评估函数）这样保证了双方评估函数互为相反数，称为**静态标准评估**。

我们总是将状态空间图搜索一定的深度来决定下一步的走法，那么非叶子节点的评估值可以由其后的搜索结果来确定，减弱了静态标准评估的粗糙对策略的不良影响。我们将非叶子节点的评价称为**标准评估**。

当拥有了标准评估的定义，我们假设博弈双方总是希望自己的评估函数在博弈过程中越大越好（实际上在这种假设下，每个玩家均是在寻求使自己目前局面评估最好的着法，而已经“忘掉”了胜败条件（偏好函数）），那么在只能进行有限深度的搜索的情况下，我们可以进行这样的搜索过程：以当前局面为根节点，用状态空间搜索展开所有可能的后继局面到规定的深度d，将深度为d-1的节点的标准评估取为其后继节点（叶子节点）中标准评估的最大值的相反数，这样搜索树深度“收缩”为d-1。继续上边的过程，直到深度为1，取使标准评估最大的动作作为下一步的动作。

注：

1. 上文为了描述简单，采用了广度优先搜索的形式。实际上棋类游戏的分支往往很大，搜索时采用深度优先。但是原理是相同的。
2. 从这个过程也可以看到，静态标准评估对搜索有很强的启发作用，但是由于最大-最小搜索的迭代特点，评估函数并不是太好时也能使机器得到不错的智力。

### alpha-beta搜索

最大-最小搜索已经包含了本博弈系统机器智力的全部内容。这里进一步的讨论是为了提高程序的效率，根据最大-最小搜索的特点对算法进行优化。

我们注意到在深度优先搜索的过程中，我们总是先沿某条路径到达一个深度为d-1的节点，然后依次搜索该节点的次级节点（叶子节点），并记录最大的评估值和最佳走法，直到本层全部被探测，则上级节点的标准评估变为该最大值的相反数，作为新的叶子节点。这个过程是递归的。

我们可以利用这个过程的先后顺序进行一些性能优化：

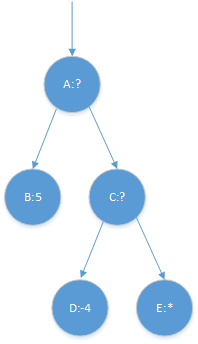
* 首先，参照图 3.1 搜索中的裁剪，假设我们已经探测过B节点的后继节点，并且确定该节点的标准评估值为5，那么在确定它右侧兄弟节点C的标准评估时，我们探测它的后继叶子节点D和E，在探测到D：-4时，已经不需要继续探测下一个节点E即可确定，A节点的评估应取为5。因为C的评估值不会大于4。这种过程被称作**最大-最小裁剪**。
* 为了更好地利用最大-最小裁剪，我们可以在搜索次级节点之前对它们进行**排序**。排序的依据可以根据不同的棋类游戏设置，比如象棋中可以将军和吃子的着法放在搜索队列的前面。但是为了将排序过程标准化，使之不需要根据具体博弈规则来配置，我们需要采取一种规则内容无关的排序规则，可以利用的方法之一就是在迭代加深的深度优先搜索中利用上一个深度的结果。具体方法未实现，如需要可以参照[3]。

图 .1 搜索中的裁剪

## 敌对搜索的多玩家扩展模型

在上一节我们知道了两个敌对玩家的状态空间搜索方式，对于多个玩家（比如跳棋的六个玩家），原静态标准评估的定义已经不适用：

* 我们还应该考虑多人之间的结盟、竞争和中立等关系。
  + 但是这些关系归根到底还是关系到获胜与否的规则的一部分。因为我们已经将获胜局面与评估函数的特定取值范围关联起来，并且假设静态评估函数取得足够好，能够引导玩家走向胜利，所以在多人敌对搜索过程中，我们仍然只需考虑玩家会选择令自己的评估函数最大的着法，且着法的选择和其余任何因素无关。
* 评估函数不是简单地取相反数就合理的。
  + 比如我们以玩家3的视角回顾搜索的过程：（假定最大深度为4）

玩家3展开-玩家4展开-玩家5展开-玩家6展开，（达到深度4限制）对每个玩家5的着法的孩子，玩家6选择自己的评估函数最大的一步，对每个玩家4的着法的孩子，玩家5选择自己的评估函数最大的一步……若评估函数只是简单地求相反数，就成了玩家5不顾自己的评估，只走让玩家6的评估尽可能小的着法，而玩家4又不顾自己的评估，走让玩家6的评估尽可能大的着法……这显然是不正确的。

所以我们可以发现多人敌对搜索与两玩家敌对搜索本质上的唯一区别：评估**不能简单地“标准化”**。即前驱玩家的评估并不是后继玩家所有后继着法最大评估的相反数。

正确的搜索过程应该是——后继玩家找到对自己最有利的走法（评估函数最大），前驱玩家的评估根据后继玩家选择的走法重新计算。

可以在不违背评估函数定义的情况下定义N玩家敌对棋类游戏的**标准静态评估函数**：

其中，为玩家j的任意一个静态评估。

这样定义的好处是：当任一玩家的评估函数与其余所有玩家的属性及其余所有玩家棋子的属性有关系（我们称每个玩家的静态评估函数**独立**）时，可以维护一个全体玩家的评估的数组作为上式的，然后玩家的标准静态评估函数计算形式被统一。但是由于它独特的计算方式，仅适用于所有人均为竞争关系的棋类游戏。

## 本章小结

本章给出了基于状态空间搜索的人工智能算法。这是本博弈系统采用的智能算法。在3.3 敌对搜索的多玩家扩展模型一节中，我们可以进一步体会到：**正向的敌对搜索是逆向的贪心搜索**。得到这个结论有助于更好地理解状态空间搜索的本质。

# 跨平台的GUI接口设计及其MFC实现

本章将给出跨平台的GUI接口设计及其MFC实现。还要实现其可根据博弈规则配置常义棋盘的功能。

## 跨平台GUI接口设计

### 接口跨平台部分简介

跨平台GUI接口为博弈系统内部提供与窗口线程的交互方式。

****图 4.1 GUI类成员函数列出了GUI类的成员函数，其中与窗口控制有关的函数为：

图 . GUI类成员函数

* drawBoard：绘制棋盘指针指向的棋盘，第二个参数是当前的着法（为了方便标出走法中含有的位置信息）
* inform：弹出内容为其1号参数的对话框，2号参数控制是否在对话框退出后退出整个程序。（比如遇到了不可恢复的严重错误）
* exit：退出整个程序。
* askForAction：向窗口线程请求输入下一个着法。（轮到了玩家走棋）

### 有人类玩家时着法生成器的扩充

在2.2.3棋类游戏走法的状态机模型中已经介绍了使用机器生成全部可能着法的机制。我们希望让人类玩家的着法生成也使用同样的模型，使得概念清晰，维护简单。

但是因为走法图是高度可配置的，直接套用整个生成过程必然要求在生成一步着法时弹出很多对话框以便玩家选择下一个操作。考虑到简单易用，我们尽可能地不询问玩家，而仅通过当前的输入猜测他的意图。

因此在GUI类中加入如下函数：

* getInput：阻塞式调用，向窗口线程请求当前的动作输入。第一个参数是被点击的棋盘上的坐标，第二个参数是动作类型，有POINT、UNDO、END三种，分别代表点击一个棋盘坐标、撤销当前未提交的着法（重新走棋）、结束本回合的着法。只有POINT类型才需要坐标值。
* getLogicalCoodination：由几何坐标得到棋盘坐标，用于翻译鼠标点击事件。

扩充后着法生成器在原着法生成算法描述的基础上加入以下规则：

每进入下一层的操作搜索，调用getInput等待输入。如果输入是UNDO，重新整个着法的搜索；否则如果是END，测试是否可以在当前着法下结束本回合，若可以，着法生成结束，否则相当于UNDO；否则输入为POINT，搜索本层所有可能的动作，如果只有一个动作含有被点击的点，选择这个动作并进入下一层搜索；如果没有动作含有被点击的点，相当于UNDO；否则有不止一个动作含有被点击的点，弹出对话框让玩家选择动作。

## 可配置的GUI棋盘模型

我们在2.2.4棋类游戏的棋盘模型中已经得到了常义棋盘的定义和逻辑模型，这里细化一下可配置的选项。

我们已经知道影响常义棋盘外形的一些变量：

基之间的夹角、直棋路的方向、任一基和屏幕上水平线的夹角、棋盘的范围。

另外，我们还需要支持局部棋路和棋盘上的辅助线、辅助点。

如象棋中九宫格的辅助线、楚河汉界上去掉棋盘的竖线，五子棋中的天元和星位。

辅助线采取异或的机制，也就是说在原本的棋盘上画线会消掉重合的部分，由此可以画出楚河汉界。

还需要配置是否画出直棋路，以及直棋路的交叉点上是否有标记。这样便于支持跳棋。

需要配置棋子是圆或方，棋子上是否写字，并为此准备圆或方的未写字的棋子图片各一个。如果棋子有字，则绘制棋盘时将字写在对应的棋子上；否则直接用GDI绘制圆或矩形代表无字的棋子。

考虑到棋子可能不是正方形或者正圆形，故设置一个“棋子矩形”属性，描述椭圆棋子的长短轴或者矩形棋子的长宽。为了保证显示效果，规定棋子的图片在棋盘上不能重叠，实现方法是将棋子的长宽和对角线向量依次在棋盘的两个基和两个基的向量和与向量差的方向上投影，比较模长。

这里不再一一指出上边描述的可配置项与GUI接口类的哪一个成员变量相对应，参见图 2.5 Windows系统下的GUI类图。

## GUI接口的MFC实现

创建MFC工程，将写好的系统内部类文件包括进来，配置MFC与GUI接口之间的通信方式。

本系统采用同步事件和消息队列两种机制来配置工作线程和窗口线程的通信。工作线程和主线程各自维护一个消息循环，配置消息的响应。在调试过程中偶尔发现内存泄露，应该是多线程的机制没有理解透彻，故不写具体实现方式，防止用错误的思想影响读者。

## 本章小结

本章介绍了跨平台GUI接口的设计，并实现了常义棋盘的显示配置。基本实现了设计目标。

# 象棋、五子棋、跳棋的简单配置实践

前面几章简要介绍了博弈系统的规则配置，下面结合具体例子介绍棋类游戏的配置。

总地来说，配置分为规则配置和界面配置两个方面。步骤总结如下：

* 规则配置：
  + 在CIGRuleConfig类中：配置棋盘长宽、配置棋盘范围、根据需要扩充操作枚举值、配置走法图。
  + 在MotionGenerator类中：改写generateMotionsForOneStatus函数，配置每个操作对应的动作。
  + 在Chessboard类中：改写onXXIntent函数，以体现博弈的规则并且规定更新evaluations数组的方法以产生静态标准评估。
* 界面配置：
  + 在GUI类中：配置棋盘和棋子的显示属性，包括基的夹角、棋路的方向、棋盘辅助线和辅助点、棋子的外形等。参见4.2可配置的GUI棋盘模型。
* 其余非常特殊的配置。

## 象棋规则和界面的配置

### 规则配置

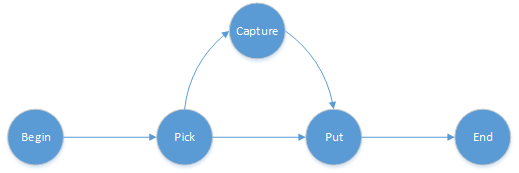
在CIGRuleConfig类中：选择16\*16的棋盘，在居中10\*9的位置上填充初始的象棋棋盘。需要使用的操作为BEGIN、PICK、CAPTURE、PUT、END，按照下图填写走法图的二维邻接数组：

图 .1象棋走法图

在MotionGenerator类中：改写generateMotionsForOneStatus函数，在“switch(operation)”语句的“case PUT:”和“case CAPTURE:”分支上根据棋子的类型规定走棋的方式。在“case PUT:”分支前加一句判断，如果上一个动作是CAPTURE，则本次的动作必须是PUT到CAPTURE动作对应的位置。

使用独立的静态评估，给不同的棋子在不同的位置赋予一定的分值，并且在Chessboard类中onXXIntent函数中移动棋子和吃子时更新总分值，如果被吃掉的是将，则被吃方除了减掉将的分值外，还要减掉一个非常大的值，来使静态评估满足定义：任一胜利局面的评估高于任一非胜利局面，任一失败局面的评估低于任一非失败局面。

因为没有特殊规则，Chessman类的onXXIntent使用默认设置。

### 界面配置及运行结果

棋盘界面的配置主要是画辅助线——消除河界上的竖线、加上九宫格的斜线。基的夹角为90°，无另外的棋路。需要画线。不标出线的交点。

采用写字的圆形棋子，边长和格子边长相同。配置不同玩家的棋子颜色。

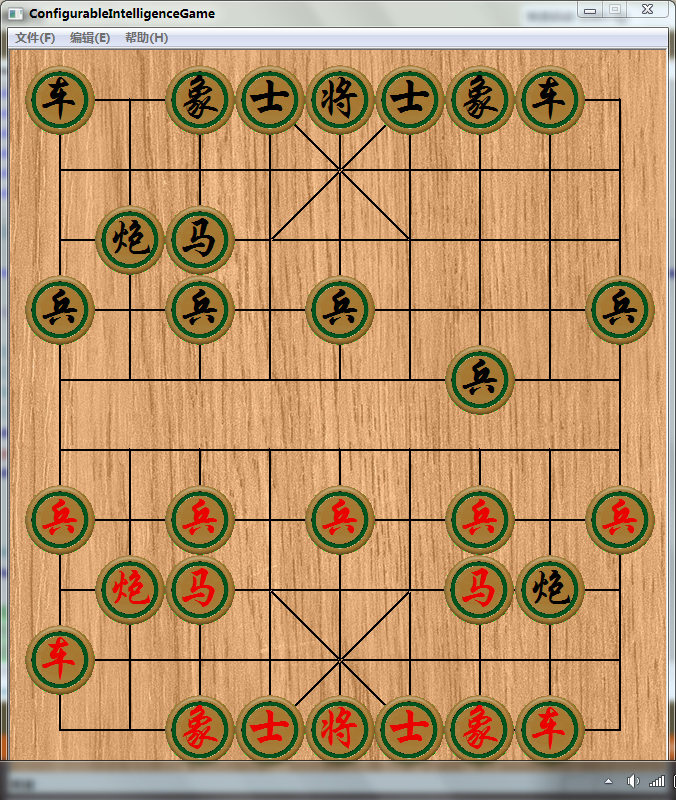
这便完成了象棋的配置，运行程序，调试正常，下图是其中的一个局面：

图 .2象棋配置结果

## 五子棋规则和界面的配置

### 规则配置

在CIGRuleConfig类中：选择16\*16的棋盘，在居中15\*15的位置上填充初始的五子棋棋盘。需要使用的操作为BEGIN、ADD、PUT、END，按照下图填写走法图的二维邻接数组：

图 .3五子棋走法图

在MotionGenerator类中：使用generateMotionsForOneStatus函数默认的ADD和PUT操作即可，无需改动。

使用非独立的静态评估，在（1,0），（0,1），（-1，1），（1,1）四个方向上给不同的敌我棋子排列形状赋予一定的分值，并且在onXXIntent函数中落子时更新分值。见下表：

注：MATE\_VALUE是一个很大的数，表示杀棋。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | 当前方向己方棋子个数 | | | | |
| 1 | 2 | 3 | 4 | >=5 |
| 当前方向对方子力情况 | 两端无子 | 30 | 100 | 1000 | MATE\_VALUE-1 | MATE\_VALUE |
| 一端有子 | 10 | 50 | 200 | 1000 | MATE\_VALUE |
| 两端有子 | 0 | 0 | 0 | 0 | MATE\_VALUE |

表 5.1 五子棋局面评估表

Chessman的onXXIntent没有特殊规则，故使用默认设置。

### 界面配置及运行结果

棋盘界面的配置主要是画辅助点——（7,7）、（3,3）、（3,11）、（11，3）、（11,11）。基的夹角为90°，无另外的棋路。需要画线。不标出线的交点。

采用无字的圆形棋子，边长和格子边长相同。配置不同玩家的棋子颜色。

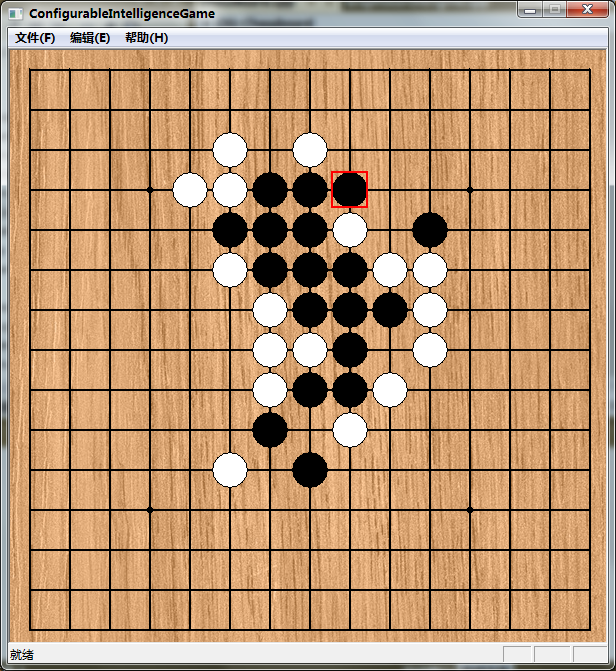
这便完成了五子棋的配置，运行程序，调试正常，下图是其中的一个局面：

图 5.4五子棋局面

## 六人跳棋规则和界面的配置

### 规则配置

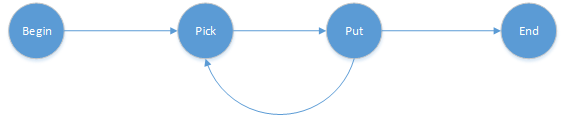
在CIGRuleConfig类中：选择32\*32的棋盘，在与图 2.4跳棋棋盘范围的定义相同的位置上填充初始的跳棋棋盘。需要使用的操作为BEGIN、PICK、PUT、END，按照下图填写走法图的二维邻接数组：

图 .5跳棋走法图

在MotionGenerator类中：改写generateMotionsForOneStatus函数，在“switch(operation)”语句的“case PICK:”分支上根据棋子本回合的着法确定走棋的方式：如果本回合没有PICK过棋子，则每个棋子均可PICK，否则只能PICK已经动过的棋子。在“case PUT:”分支上：可以沿着任一直棋路的方向落子，如果本回合没有过落子，则可以走一步或跳一步；否则本回合已经移动过一个棋子，如果上一步是跳，则本次也应该跳，否则上一步是走一步，不应该再走，返回false。

使用独立的静态评估，为了鼓励玩家迅速地将棋子移出大本营，把每个棋子到对角**路程**的平方的减少量作为分值，并且在Chessboard类中onXXIntent函数中移动棋子时更新总分值。

因为没有特殊规则，Chessman类的onXXIntent使用默认设置。

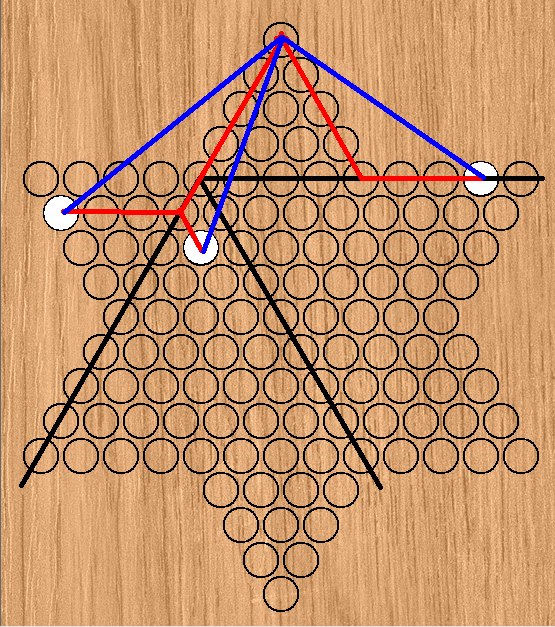
这里，**路程**需要精确计算：

图 .6 计算跳棋的的路程

上图中，黑色线代表三条直棋路，分别是x轴、y轴和（1,1）方向。（再次说明这里x轴与y轴正方向的夹角为120°）蓝色线代表当前棋子坐标与目标位置坐标的差向量（称作**相对位移**）。红色线代表棋子到目标的实际可走的最短路程之一。

那么我们可以发现，相对位移在第一、三象限的，可以走（1,1）方向以便同时在x和y方向朝目标靠拢，故路程为x和y两坐标绝对值中的最大值。而相对位移在二、四象限的，只可以走基的方向，故路程为x和y两坐标绝对值的和。

注：你可以发现跳棋棋盘的形状恰好不影响路程的计算。

### 界面配置及运行结果

棋盘界面的配置主要是定义附加直棋路方向（1,1）。基的夹角为120°。不需要画线。用圆圈标出线的交点。

采用不写字的圆形棋子，边长和格子边长相同。配置不同玩家的棋子颜色。

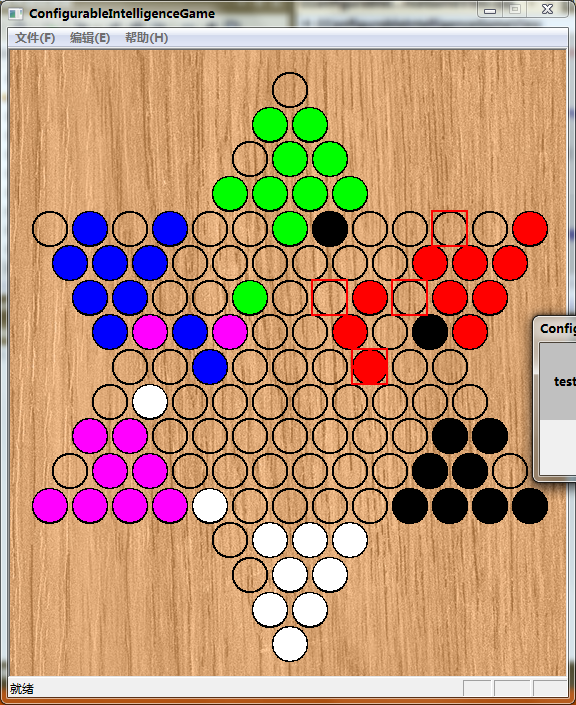
这便完成了跳棋的配置，运行程序，调试正常，下图是其中的一个局面：

图 .7跳棋配置结果

## 本章小结

本章配置了象棋、五子棋和跳棋，验证了设计的正确性，也证明了系统是可扩展的。

在残局的走法上，可能还需要更仔细地调整。评估函数可能取得有些粗糙，这些问题有待进一步研究。

# 论文总结

本篇论文主要介绍了空间离散的回合制棋类游戏的博弈平台设计。围绕着这个总目标，论文从博弈等相关的定义开始，介绍了博弈系统的框架设计、博弈内核的构造、跨平台GUI接口的设计。为了能让读者对本系统有更清晰的认识，也为了证明设计的正确性，在第5章介绍了三种常见棋类游戏的配置。

但是本设计还有很多有待改进的地方，比如评估函数的配置应该交给电脑自动完成（比如使用神经网络等），应该支持将现有的开局库和残局库导入到本系统的功能，可以考虑将统一层次的搜索采用并行计算完成、应该克服简单的敌对搜索引发的一些问题（如水平线效应[4]）、可以考虑在人类玩家进行思考时机器也进行思考以节省一些时间、设计一种机制检测重复局面防止重复搜索等等。由于我目前自身能力及毕业设计时间的限制，无法将这些一一完成。如果这些工作都能得到完成，相信这个博弈系统就已经走向成熟了。

#### 参考文献

[1] 维基百科 博弈论

[2] Nilsson N J. Artificial Intelligence: A New Synthesis[M]. Morgan Kaufmann, 1998.

[3] 象棋百科全书网《对弈程序基本技术》专题 迭代加深 Bruce Moreland / 文

[4] 象棋百科全书网《对弈程序基本技术》专题 什么样的结点需要搜索？全部还是选择性的？ David Eppstein /文 加州爱尔文大学(UC Irvine)信息与计算机科学系

[5] Booch G. The Unified Modeling Language User Guide, 2/E[M]. Pearson Education India, 2005.

[6] Blaszczak M. Professional MFC Visual C++ 5[M]. Wrox Press Ltd., 1997.

[7] Kruglinski D J, Wingo S, Sheperd G W. Programming Microsoft Visual C++[M]. Microsoft press, 1998.