#### 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。

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。

#### 目录

[第1章 绪论 6](#_Toc374308473)

[1.1 相关背景介绍 6](#_Toc374308474)

[1.1.1 可配置编程平台的背景介绍 6](#_Toc374308475)

[1.1.2 智能博弈系统的背景介绍 6](#_Toc374308476)

[1.2 可配置智能博弈系统的作用和意义 6](#_Toc374308477)

[1.3 文章的后续框架 7](#_Toc374308478)

[第2章 博弈系统的特征提取和博弈平台的编程框架设计 8](#_Toc374308479)

[2.1 博弈系统的特征提取 8](#_Toc374308480)

[2.1.1 博弈的分类及其各自特点分析 8](#_Toc374308481)

[2.1.2 空间离散回合制棋类游戏的智能博弈分析 10](#_Toc374308482)

[2.2 博弈平台编程框架设计 10](#_Toc374308483)

[2.2.1 棋类游戏的各部件属性模型 11](#_Toc374308484)

[2.2.2 棋类游戏的规则模型 12](#_Toc374308485)

[2.2.3 棋类游戏走法的状态机模型 13](#_Toc374308486)

[2.2.4 棋类游戏的棋盘模型 16](#_Toc374308487)

[2.3 棋类博弈平台的面向对象分析-UML辅助 18](#_Toc374308488)

[2.3.1 辅助类和接口类设计 18](#_Toc374308489)

[2.3.2 基础类设计 20](#_Toc374308490)

[2.3.3 部件类设计 21](#_Toc374308491)

[2.3.4 全局设计，类的继承关系和依赖关系 24](#_Toc374308492)

[2.4 本章小结 25](#_Toc374308493)

[第3章 多角色敌对搜索内核的构造 26](#_Toc374308494)

[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 26](#_Toc374308495)

[3.1.1 状态空间搜索[2] 26](#_Toc374308496)

[3.2 敌对搜索简介 27](#_Toc374308497)

[3.2.1 最小-最大搜索 27](#_Toc374308498)

[3.2.2 alpha-beta搜索 28](#_Toc374308499)

[3.3 敌对搜索的性能优化及其多人多机扩展模型 28](#_Toc374308500)

[3.4 本章小结 28](#_Toc374308501)

[第4章 基于嵌套状态机和图搜索的着法生成器构造 29](#_Toc374308502)

[4.1 可配置的着法搜索模型 29](#_Toc374308503)

[4.2 图搜索算法的深度优先搜索实现 29](#_Toc374308504)

[4.3 本章小结 29](#_Toc374308505)

[第5章 跨平台的GUI接口设计及其MFC实现 30](#_Toc374308506)

[5.1 跨平台GUI接口设计 30](#_Toc374308507)

[5.2 可配置的GUI棋盘模型 30](#_Toc374308508)

[5.3 GUI接口的MFC实现 30](#_Toc374308509)

[5.4 本章小结 30](#_Toc374308510)

[第6章 象棋、五子棋、跳棋的简单配置实践 31](#_Toc374308511)

[6.1 象棋规则和界面的配置 31](#_Toc374308512)

[6.2 五子棋规则和界面的配置 31](#_Toc374308513)

[6.3 六人跳棋规则和界面的配置 31](#_Toc374308514)

[6.4 本章小结 31](#_Toc374308515)

[第7章 论文总结 32](#_Toc374308516)

[第8章 参考文献 33](#_Toc374308517)

#### 图目录

[图 1国际象棋走法生成图 14](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305054)

[图 2跳棋走法生图 14](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305055)

[图 3操作与动作的并行包含关系 15](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305056)

[图 4跳棋棋盘范围的定义 17](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305057)

[图 5 GUI类图 18](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305058)

[图 6 CIGObejct类图 19](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305059)

[图 7 CIGRuleConfig类图 19](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305060)

[图 8 Stack模板类图（继承自Array类） 20](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305061)

[图 9 Array模板类图 20](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305062)

[图 10 Player类图 21](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305063)

[图 11 Chessman类图 21](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305064)

[图 12 Chessboard类图 22](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305065)

[图 13 Operation类图 23](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305066)

[图 14 ChessmanIndex类图 23](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305067)

[图 15 ChessmanIndexBoard类图 23](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305068)

[图 16 GraphSearchEngine类图 24](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305069)

[图 17 Game类图 24](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305070)

[图 18 所有类图 25](file:///E:\课题\双学位毕业论文\论文\毕业论文-石守谦.docx#_Toc374305071)

#### 内容摘要

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，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，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空气。

关键字:

人工智能 博弈系统 可配置 最大最小搜索 alpha-beta搜索 扩充搜索

#### Abstract

Undone

Keywords：undone

# 绪论

## 相关背景介绍

### 可配置编程平台的背景介绍

### 智能博弈系统的背景介绍

## 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，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，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空气。

## 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；第四章介绍通用棋类着法生成器的设计；第五章介绍MFC下GUI接口的跨平台设计与具体实现；第六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：[2]

设定有N 个“**参与者**”（后文也常称“**玩家**”），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。 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，描述策略规定结果的方式。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。[2]

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**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**属性**。

定义**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”）。

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

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。

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：

**局面**：整个游戏中影响任一玩家策略集合的属性集合。

**着法**：每个回合玩家所有的行动。（后文也称**行动**）

**动作**：玩家当前回合内可以采取的单个策略。

**操作**：泛指玩家影响局面的一类动作。

**走法**：泛指玩家的着法，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。往往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
* 当前回合操作者（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）
* 当前季节（比如模拟战争游戏）
* 当前游戏总时间
* 历史着法（可能规则允许悔棋，也对如象棋中“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
* 棋手名字
* 棋手颜色
* 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:
* 颜色
* 生命值
* 行动力
* 行动消耗的行动力

### 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）。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
* 生命值增长率
* 棋子相关的通用规则：
* 本类棋子着法产生的具体规则
* 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（Action）、动作（Motion）、走法（Option Graph）、操作（Option）”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。

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：

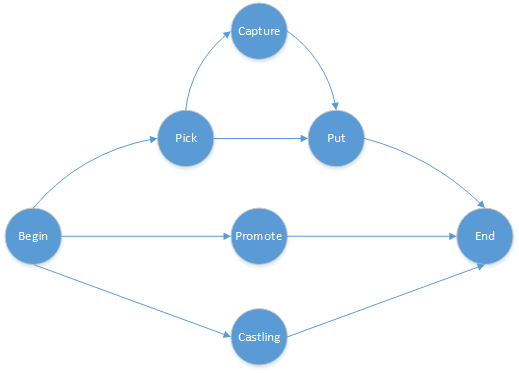
1. 国际象棋中，“一步棋”（一个着法）可以被如下的操作图生成：

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。

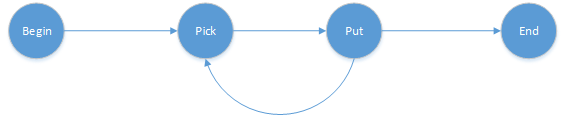
1. 跳棋中，“一步棋”（一个着法）可以被如下的操作图生成：

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”是动作路径，而非操作路径，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。

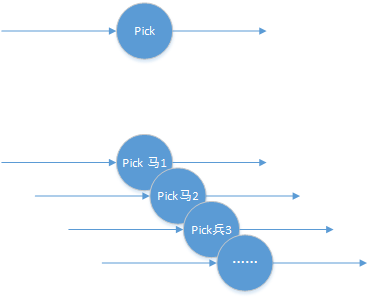
例如，实际由搜索生成的动作图的一部分举例如下：

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，能完全地扫过所有并行的动作，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。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，以便在探测到end节点时保存整个路径。

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。
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，操作和动作运行栈清空，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，本层搜索的入操作运行栈。
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)，继续搜索当前操作的其余动作。
6. 若当前动作运行栈空, 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。（这里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）

### 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。

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的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**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。

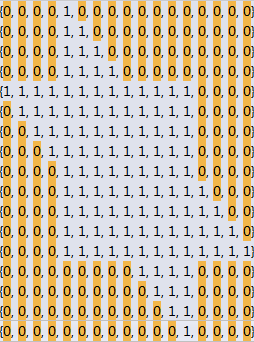
又如跳棋，基的夹角120°，直棋路有三条，两个分别为两个基的方向，另一个为两个基的向量和的方向。范围为下面矩阵为1的地方，这样既实现了棋盘范围的任意配置，又实现了跳棋可以行走的三个方向。

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。

## 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，并采用C++的术语来描述代码的构造。

### 辅助类和接口类设计

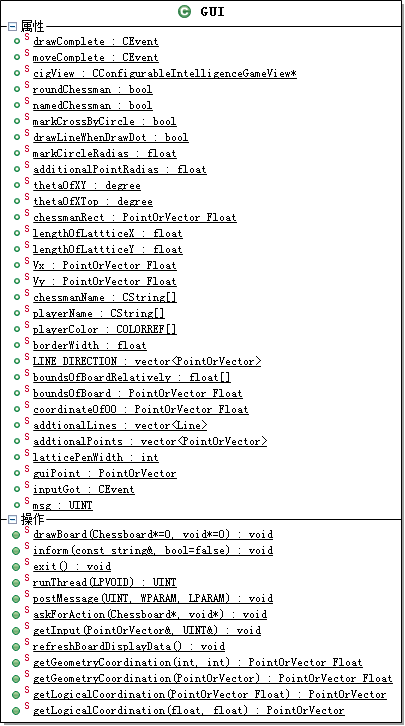
我们需要一个界面，为了使系统有较好的可移植性，独立出一个GUI类作为接口，而所有需要调用窗口显示功能的系统内部组件均调用GUI类的静态函数，使得移植时仅需要改动GUI类。

图 5 GUI类图

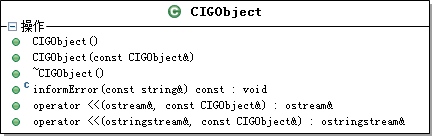
我们需要一个统一的报错机制，因此可以定义一个棋盘、棋手、棋子等类的共同基类CIGObject，并实现其informError虚函数。

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。

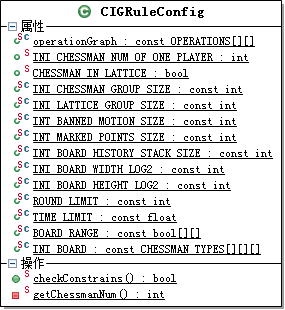


图 7 CIGRuleConfig类图

### 基础类设计

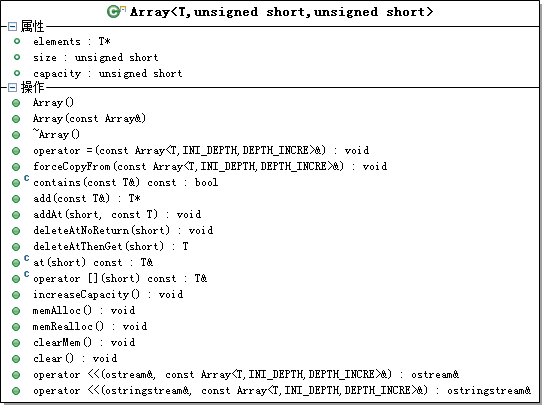
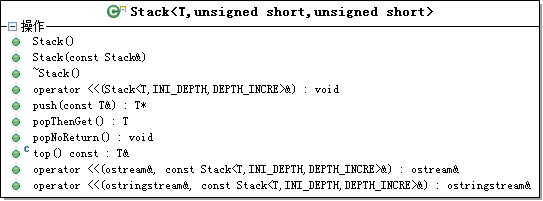
我们需要管理局面的信息，应该使用一些容器类。但是STL容器的功能不够强大，而且对于大容量数组的初始化需要依次调用默认构造函数，而很多情况下，仅需将对应存储空间memset即可正确初始化，所以自己构造一个专用的Array模板类和Stack模板类，实现元素标准的构造、析构、复制和快速的赋初值、删除、复制，并根据不同元素类型在调用时选择不同的函数集。

图 8 Stack模板类图（继承自Array类）

图 9 Array模板类图

### 部件类设计

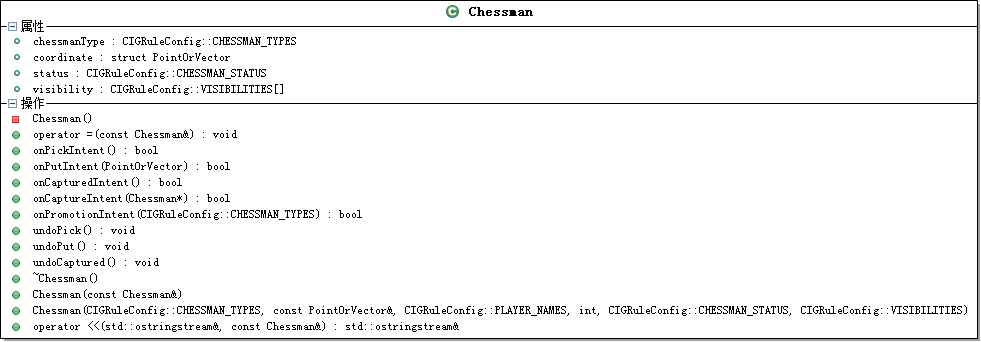
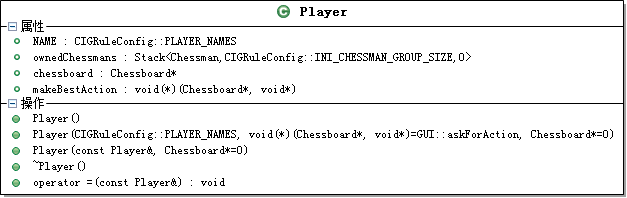
我们需要一个Chessboard类来管理棋盘属性和全局规则，一个Player类管理玩家相关属性和玩家相关规则，一个Chessman类管理棋子相关属性和棋子相关规则。

图 10 Player类图

图 11 Chessman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。

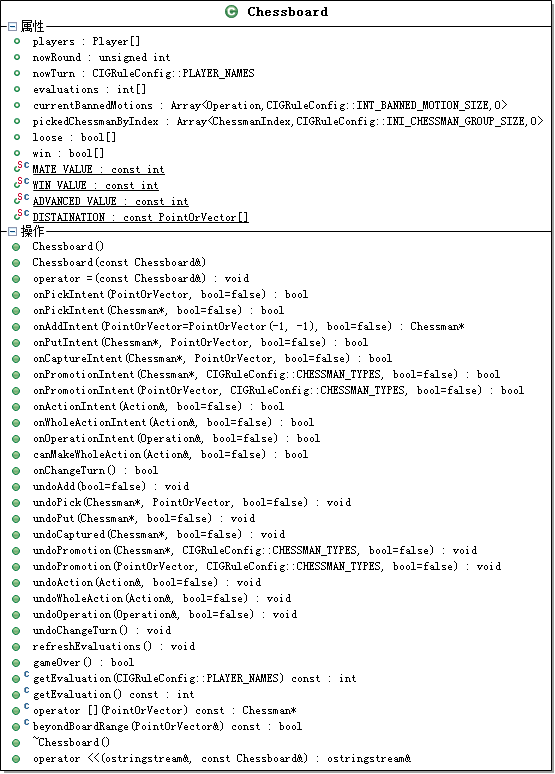
为了实现撤销一个着法，在Chessboard类中设置undoXXIntent系列函数，和undoAction函数。为了实现回合的轮换，在Chessboard类中设置onChangeTurn和undoChangeTurn函数。

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。

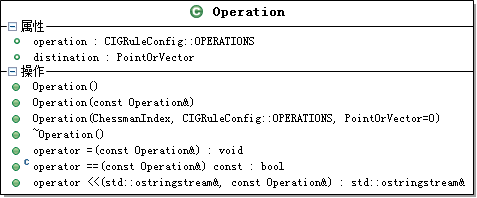
为了完全表示一个动作，引入一个Operation类。

图 13 Operation类图

其中PointOrVector是一个整数2维向量类。

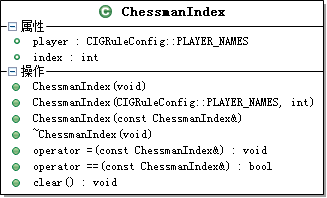
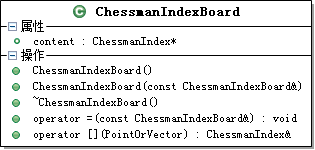
Chessboard类有一个实例成员chessmanIndexBoard，储存棋子的索引。

图 14 ChessmanIndex类图

图 15 ChessmanIndexBoard类图

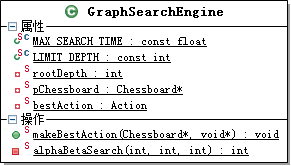
为了实现人工智能，将Player的makeBestAction成员函数变成函数指针，当玩家是机器时，将该指针赋值为GraphSearchEngine类的makeBestAction函数，否则玩家为人，指针赋值为GUI类的askForAction函数。

图 16 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。

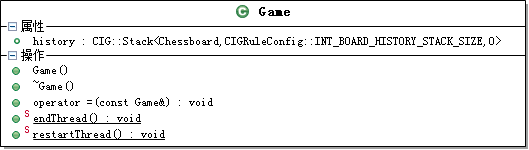
为了整合局面资源，加入Game类，现在只拥有历史棋盘和当前棋盘两个成员。

图 17 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 18 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]

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。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，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：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，除非在残局或官子阶段，分支如此之多，能搜索的空间如此之大，根本不能得到一定获胜的路线。

但是我们要在视野有限的前提下决定最可能对自己最有利的走法。这时需要扩充博弈的定义，加入**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。

定义对某局面的**评价**：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，避免了静态标准评估的粗糙导致的策略出错。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，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。这样搜索数深度变为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结果。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：

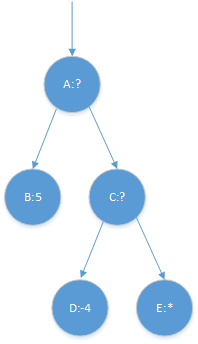
* 首先，参照图 19 搜索中的裁剪，假设我们已经探测过B节点的后继节点，并且确定该节点的标准评估值为5，那么在确定它右侧兄弟节点C的标准评估时，我们探测它的后继叶子节点D和E，在探测到D：-4时，已经不需要继续探测下一个节点E即可确定，A节点的评估应取为5。因为C的评估值不会大于4。这种过程被称作**最大-最小裁剪**。
* 为了更好地利用最大-最小裁剪，我们可以在搜索次级节点之前对它们进行排序。排序的依据可以根据不同的棋类游戏设置，比如象棋中可以将军和吃子的着法放在搜索队列的前面。但是为了将排序过程

图 19 搜索中的裁剪

## 敌对搜索的性能优化及其多人多机扩展模型

## 本章小结

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。

# 基于嵌套状态机和图搜索的着法生成器构造

## 可配置的着法搜索模型

## 图搜索算法的深度优先搜索实现

## 本章小结

# 跨平台的GUI接口设计及其MFC实现

## 跨平台GUI接口设计

## 可配置的GUI棋盘模型

## GUI接口的MFC实现

## 本章小结

# 象棋、五子棋、跳棋的简单配置实践

## 象棋规则和界面的配置

## 五子棋规则和界面的配置

## 六人跳棋规则和界面的配置

## 本章小结

# 论文总结

本篇论文主要介绍了。围绕着这些核心工作，论文介绍了。为了能让读者对有清晰的认识，论文依次介绍了，随后等，而论文的主要工作放在了最后几章。该论文最后介绍了

但是还有很多有待改进的地方，比如等等。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，。

# 参考文献

[1]

[2]人工智能课本