目录

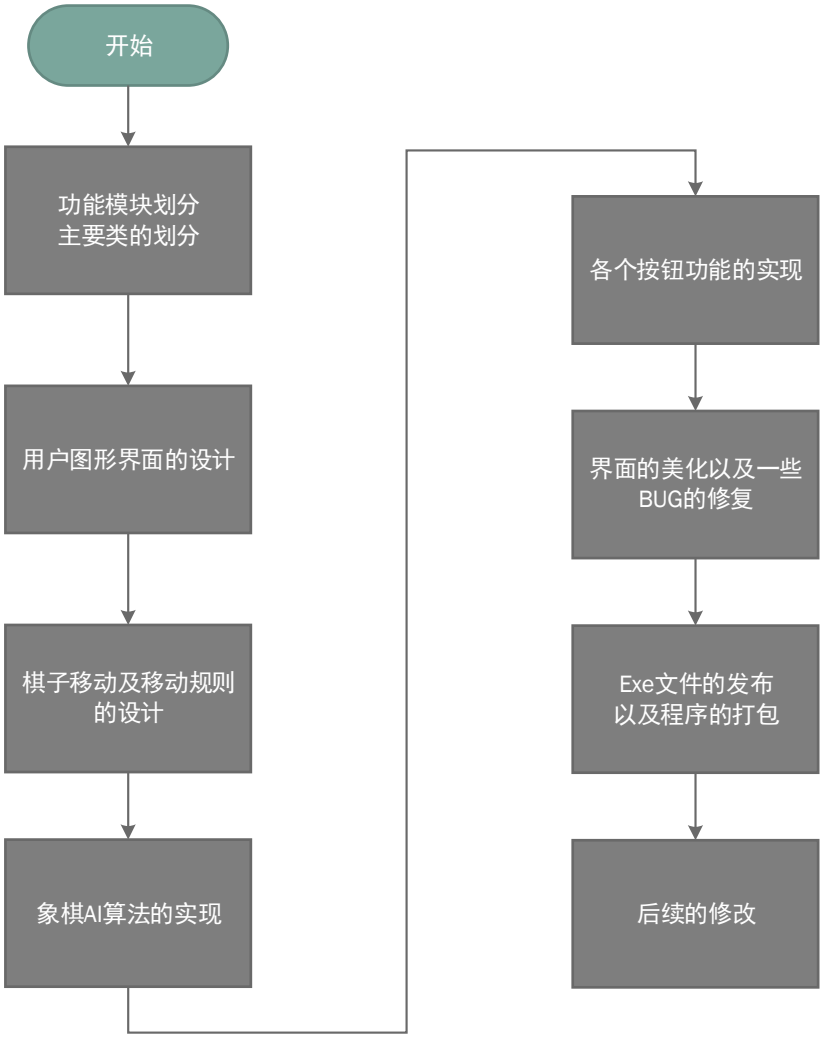
中国象棋软件需求与设计报告

1. 系统概述

1.1 软件用途

中国象棋一种历史悠久的中国棋类游戏，由两人轮流走子，以“将死”或“困毙”对方将（帅）为胜的一种棋类运动，有着数以亿计的爱好者。它不仅能丰富文化生活，陶冶情操，更有助于开发智力，启迪思维，锻炼辨证分析能力和培养顽强的意志。对局时，由执红棋的一方先走，双方轮流各走一着，直至分出胜负。本软件是一款基于中国象棋游戏规则而开发的象棋游戏，采用 C++ 的 Qt 图形库，并嵌入人工智能的博弈算法，实现了双人对战和人机对战功能，是一款休闲益智的小游戏。

1.2 软件开发过程



2. 系统需求说明

2.1 系统总体功能

该系统的总体功能为：实现了双人对战和简单的人机对战，并提供悔棋，新游戏，记录步法，音乐开关，难度设置，认输，保存棋局，载入棋局等功能。

2.2 环境需求

开发环境：Qt Creator 2.4.1 ,基于 Qt 4.7.4(32bit)

编译器：Qt 4.7.4 For Desktop - MinGW 4.4 (Qt SDK) 发布

运行操作系统: Windows xp/Win 7/Win 8（其他未测试）

占用空间：安装文件 28.3M 安装后 47.1M

运行时占用内存：13，212K --- 46384K

2.3 系统功能需求

(1) 双人对战：可实现单机版的双人对战；

(2) 人机对战:人与具备一定 AI 的电脑对弈,电脑根据人的步法自动搜索最佳的的下一步；

(3) 电脑难度设置：可以根据用户的需求设置不同的难度等级，主要有三种：低级，中级和高级；

(4) 悔棋功能：针对用户出现失误的情况，为用户提供两种悔棋方案：人机对战时，在电脑思考完成并移动棋子之后，用户点击悔棋按钮可以悔棋两步（一个回合），即撤销用户上一步操作；双人对战时，由于是单机操作，无法判断是哪个用户点击了悔棋按钮，因此此时点击悔棋按钮时只能悔棋一步，悔棋多步需要多次点击悔棋按钮；

(5) 双人对战/人机对战的两种切换模式：在主界面上点击 MM/MP 按钮时，会直接切换对战模式，而不会重新开始游戏；在游戏菜单中点击双人对战或人机对战会切换模式并重新开始游戏；

(6) 背景音乐：用户可以自定义背景音乐的开关，在游戏菜单->选项里面；

(7) 下棋音效：用户可以自定义下棋音效的设置，包括选中音效，落子音效和吃子音效，在游戏菜单->选项里面；

(8) 新游戏：用户可以在不改变任何设置下开始一次新的对局；

(9) 认输：在局面已经到了无法挽回的地步时，必败的一方可以选择认输；

(10)保存棋局：在用户觉得这盘棋下的很精彩或者有事外出不能继续时，可以选择保存已有的局面和下过的所有步法，以便恢复；

(11)载入棋盘：用户可以载入已经保存了的棋盘；

(12)显示棋步：游戏主界面的右侧会显示已经走过的所有步法；

(13)历史人物介绍：每次新的对局时，游戏双方会随机代表一位楚汉时期的历史人物，并附上人物的介绍，以增加游戏的趣味性，每方都用楚汉时期的九位代表人物可以选择。

(14)帮助，用户点击帮助后会打开用户手册，指导用户软件的使用；

(15)退出：用户点击游戏菜单的退出或者窗口右上角的“x”可以退出游戏。

3. 系统设计

3.1 系统级设计决策

本软件准备采用 Qt 图形库，主要是因为 Qt 图形库相对于 FLTK，在使用方面会比较灵活一些，而且自己也准备以此契机学习一个新的图形库。

软件将实现基本的双人对战和人机对战,时间足够的话再加入局域网对战模式。

3.2 系统总体设计

3.2.1 设计思想

本软件主要构思如下：

(1) 棋盘和棋子的表示

使用背景图片作为棋盘或者绘制棋盘,然后在棋盘窗口上画出 32 颗棋子(或使用图片)，棋子的移动本质上窗口对鼠标点击事件的响应,因此这里主要问题是对鼠标两次点击事件的处理。同时为了在鼠标点击时能够定位的点击的棋子，需要定义一个 10x9 的棋盘数组，其上以数字 -1 - 31 来代表没有棋子和 32 颗棋子。

（2）走子规则和 AI 算法

实现了基本的移动操作后可以添加走子规则，然后再添加 AI 算法。AI 算法将考虑新开一个线程来执行，这样在红方走完一步后立即启动 AI 搜索线程，搜索到正确的步法后根据结果移动棋子。

这些基本功能完成后，我将主要有四个类，首先是主窗口 MainWindows，然后主窗口上添加一个主游戏类(ChessGame)，在游戏主类初始化 32 颗棋子类(ChessMan)，并实现鼠标点

击事件和走子方法。最后在游戏主类中初始化一个 AI 引擎类，根据当前局面得到最佳的下一步。

（3）其他窗口部件

完成了以上模块后，再考虑加入其它窗口部件，如启动界面，界面上的按钮，步法文本框，菜单栏功能等等。

（4）最后是对界面的一些美化和修复一些 BUG，这样便完成该系统的设计和实现。

程序的关键技术在于对游戏中各个类的划分，首先需要一个窗口类，游戏主类在窗口中运行，然后主类中初始化其他类并完成功能。其他类之间应保持相对独立，各个类实现的功能也应该单一，各个类之间公用的宏定义变量和函数等可以在公共包含的头文件中声明。有了清晰的模块结构，可以提高代码编写调试的效率，同时可以很快定位。我花了两天时间才理清楚各个类之间的联系，并在其后又经历了一次代码重构，因此我体会到代码的结构是很重要的。

程序的难点在于 AI 算法的编写，如何进行搜索得到最佳的下一步，我准备采用老师提供的经过 α－β 剪枝的最小最大值搜索法,然后采用通用的 PVS 算法进行剪枝优化。后续再详细解释。

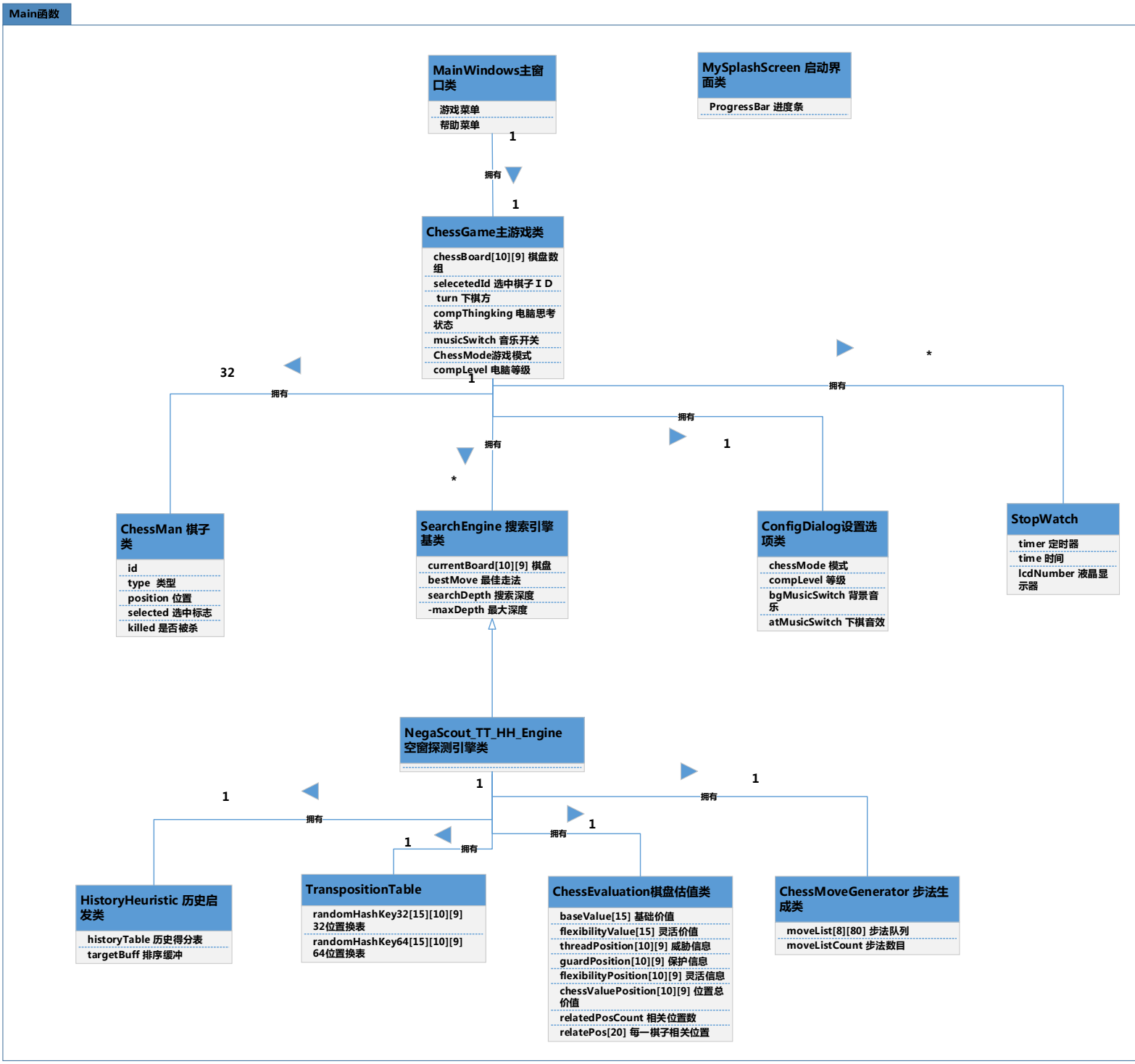
该软件中涉及到的算法主要是 AI 中的算法，其次是一个走步规则的简单判断算法，后续将详细解释。

该软件中涉及到的数据结构不是很多,首先是一个基本的棋盘数组来存储每个棋子所在的位置；然后需要一个结构来定义下棋的步伐，该结构中主要包括移动棋子的 Id，起点，终点和被吃掉棋子的 Id（没有则为-1）；还需要使用 Vector 来存放下过的所有步法，便于悔棋。

3.2.2 系统体系结构

系统类图如下：

Main函数


MainWindows主窗口类
游戏菜单
帮助菜单
MySplashScreen 启动界面类
ProgressBar 进度条
1
拥有
1
ChessGame主游戏类
chessBoard[10][9] 棋盘数组
selecetedId 选中棋子ＩＤ
turn 下棋方
compThingking 电脑思考状态
musicSwitch 音乐开关
ChessMode游戏模式
compLevel 电脑等级
32
拥有
1
拥有
*
拥有
拥有
*
1
ChessMan 棋子类
id
type 类型
position 位置
selected 选中标志
killed 是否被杀
SearchEngine 搜索引擎基类
currentBoard[10][9] 棋盘
bestMove 最佳走法
searchDepth 搜索深度
-maxDepth 最大深度
ConfigDialog设置选项类
chessMode 模式
compLevel 等级
bgMusicSwitch 背景音乐
atMusicSwitch 下棋音效
StopWatch
timer 定时器
time 时间
lcdNumber 液晶显示器
NegaScout_TT_HH_Engine 空窗探测引擎类
1
1
拥有
1
拥有
1
拥有
1
拥有
HistoryHeuristic 历史启发类
historyTable 历史得分表
targetBuff 排序缓冲
TranspositionTable
randomHashKey32[15][10][9] 32位置换表
randomHashKey64[15][10][9] 64位置换表
ChessEvaluation棋盘估值类
baseValue[15] 基础价值
flexibilityValue[15] 灵活价值
threadPosition[10][9] 威胁信息
guardPosition[10][9] 保护信息
flexibilityPosition[10][9] 灵活信息
chessValuePosition[10][9] 位置总价值
relatedPosCount 相关位置数
relatePos[20] 每一棋子相关位置
ChessMoveGenerator 步法生成类
moveList[8][80] 步法队列
moveListCount 步法数目

由该类图可知，主程序的 Main 函数中主要包括了两个窗口，首先是启动界面，显示加载进度条和软件信息；然后是主窗口 MainWindow，主窗口中包括菜单栏和中央窗口部件 ChessGame，ChessGame 是游戏的主窗口类，所有游戏的类和其他部件均在主类中实现。
主游戏类中主要包括 4 个类：32 颗棋子类，搜索引擎类，设置选项类，计时秒表类。
搜索引擎类是采用子搜索引擎继承基类的方法实现，这样便于以后扩展多个搜索引擎。本软件鉴于时间有限，只采用了一种引擎，即 NegaScount 加上置换表和历史启发的引擎，该引擎可以获得比 Alpha-Beta 引擎更快的搜索速度。

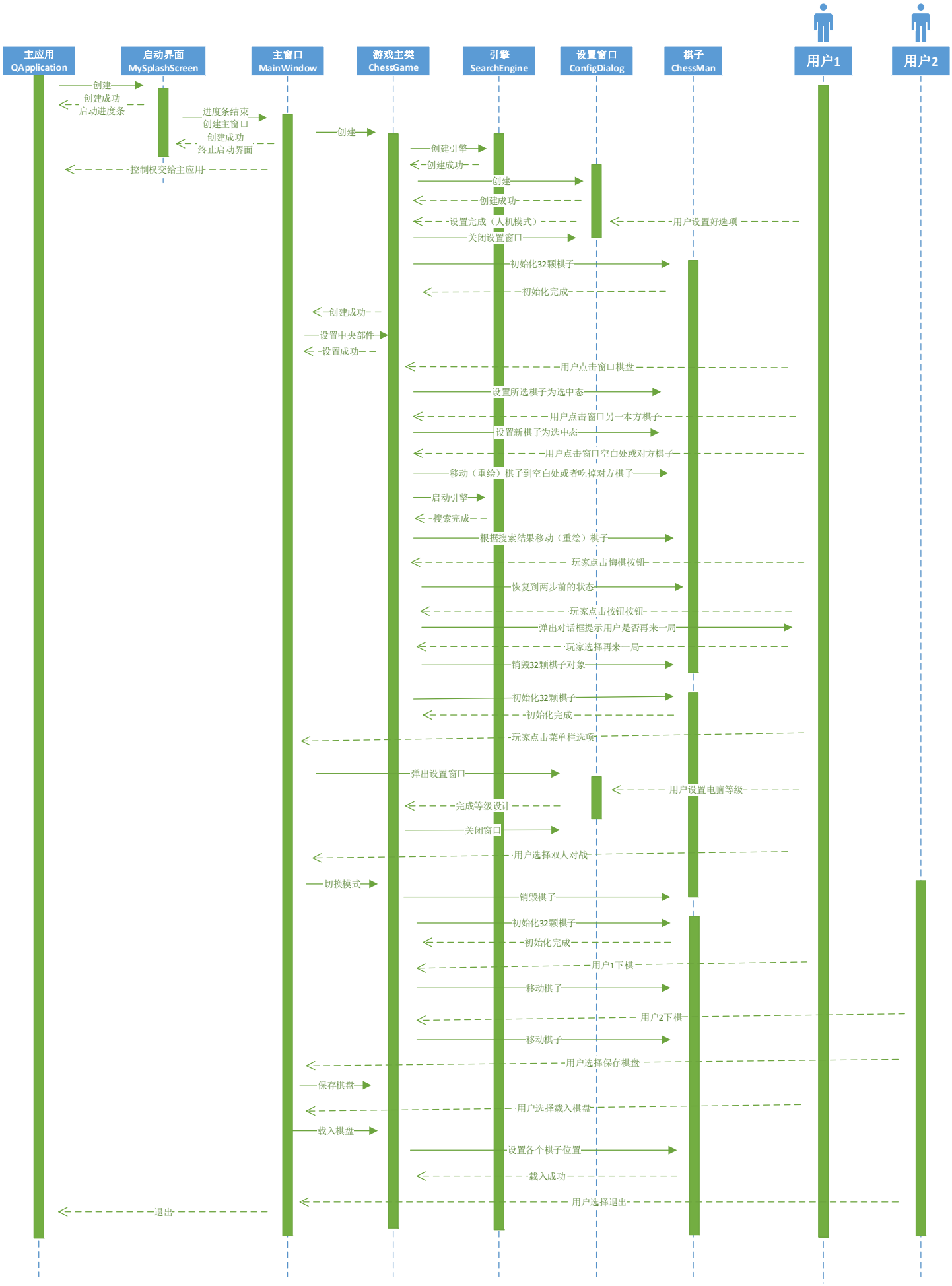
搜索引擎底下包括四个基类：分别是走法产生器，局面评估类，哈希置换表，历史启发类，他们构成了搜索引擎的核心；

设置选项类主要用于设置游戏的模式,难度和音乐开关等，主要继承自弹出对话框 QDialog；

计时秒表类主要用于在游戏中对双方用时进行计算,同时达到指示轮到哪方下棋的目的。

3.2.3 系统动态行为

系统执行中的顺序图如下：

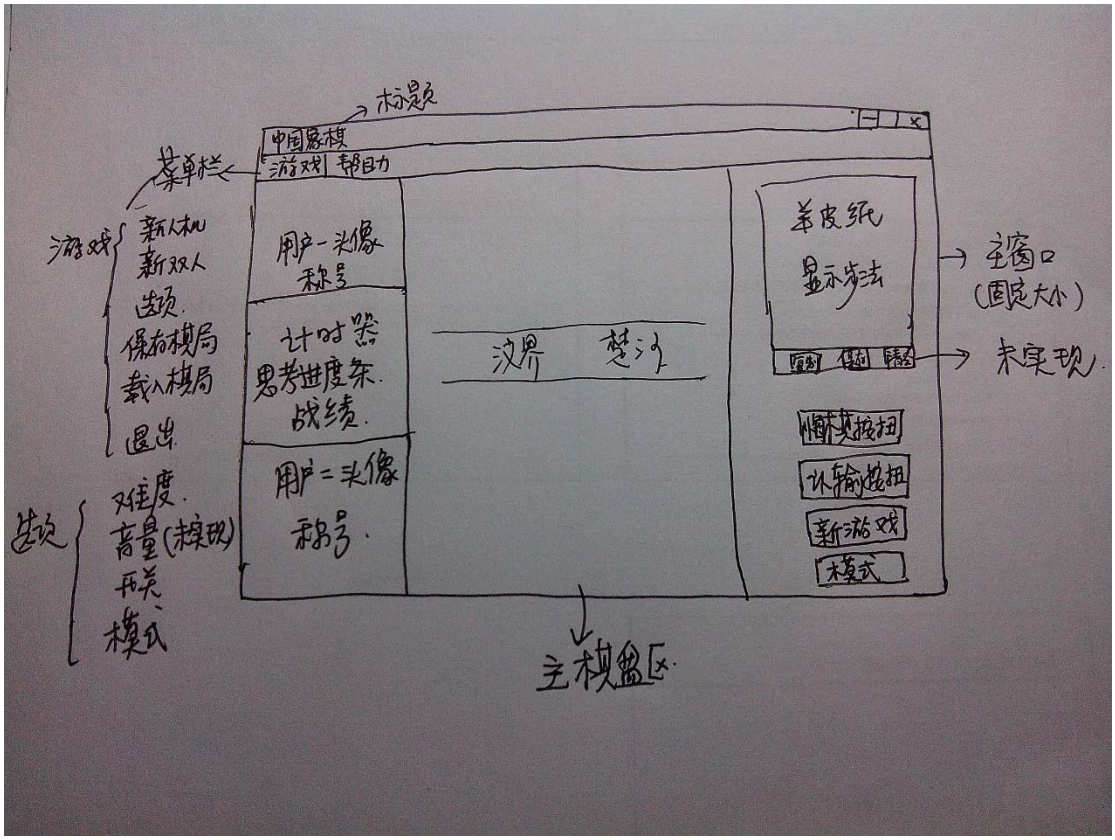

主应用
QApplication
启动界面
MySplashScreen
主窗口
MainWindow
游戏主类
ChessGame
引擎
SearchEngine
设置窗口
ConfigDialog
棋子
ChessMan
用户1
用户2
创建
创建成功
启动进度条
进度条结束
创建主窗口
创建成功
终止启动界面
控制权交给主应用
创建
创建引擎
创建成功
创建
创建成功
设置完成（人机模式）
用户设置好选项
关闭设置窗口
初始化32颗棋子
初始化完成
创建成功
设置中央部件
设置成功
用户点击窗口棋盘
设置所选棋子为选中态
用户点击窗口另一本方棋子
设置新棋子为选中态
用户点击窗口空白处或对方棋子
移动（重绘）棋子到空白处或者吃掉对方棋子
启动引擎
搜索完成
根据搜索结果移动（重绘）棋子
玩家点击悔棋按钮
恢复到两步前的状态
玩家点击按钮按钮
弹出对话框提示用户是否再来一局
玩家选择再来一局
销毁32颗棋子对象
初始化32颗棋子
初始化完成
玩家点击菜单栏选项
弹出设置窗口
用户设置电脑等级
完成等级设计
关闭窗口
用户选择双人对战
切换模式
销毁棋子
初始化32颗棋子
初始化完成
用户1下棋
移动棋子
用户2下棋
移动棋子
用户选择保存棋盘
保存棋盘
用户选择载入棋盘
载入棋盘
设置各个棋子位置
载入成功
用户选择退出
退出

由于游戏中有两种模式，因此在系统的顺序图中涉及到两位用户，一开始是人机对战，后来进行了模式的切换，切换成双人对战。系统的顺序图基本囊括了所有功能的操作。

该软件对于异常操作已经不予响应，（如新游戏下点击悔棋等），因此用户的异常操作顺序图不用画。

3.3 用户界面设计

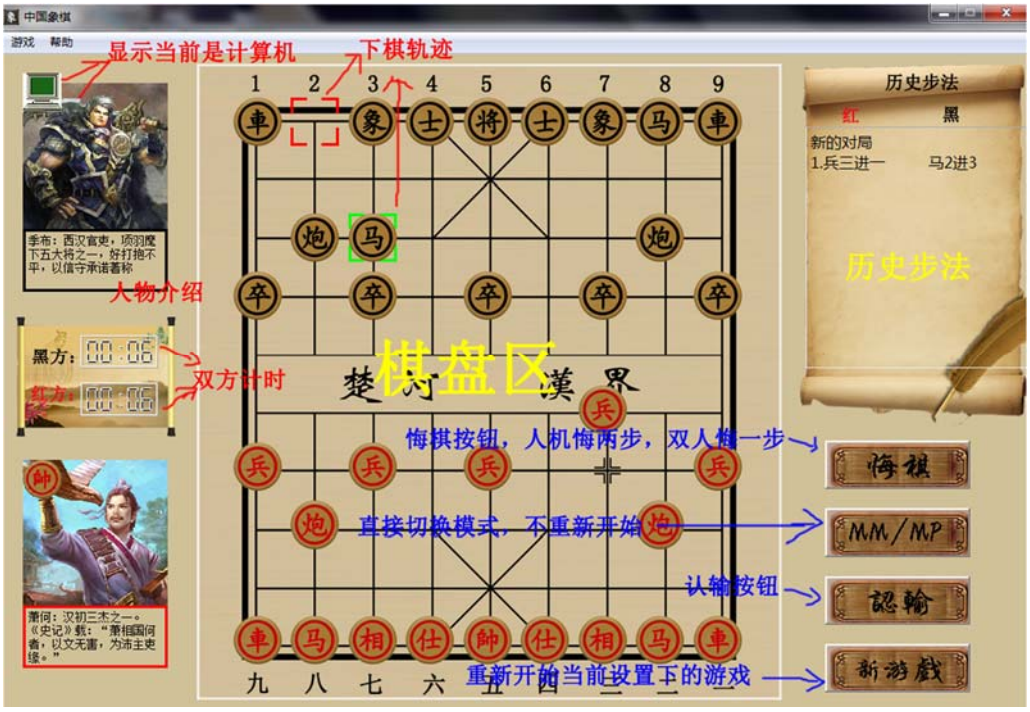
草图：



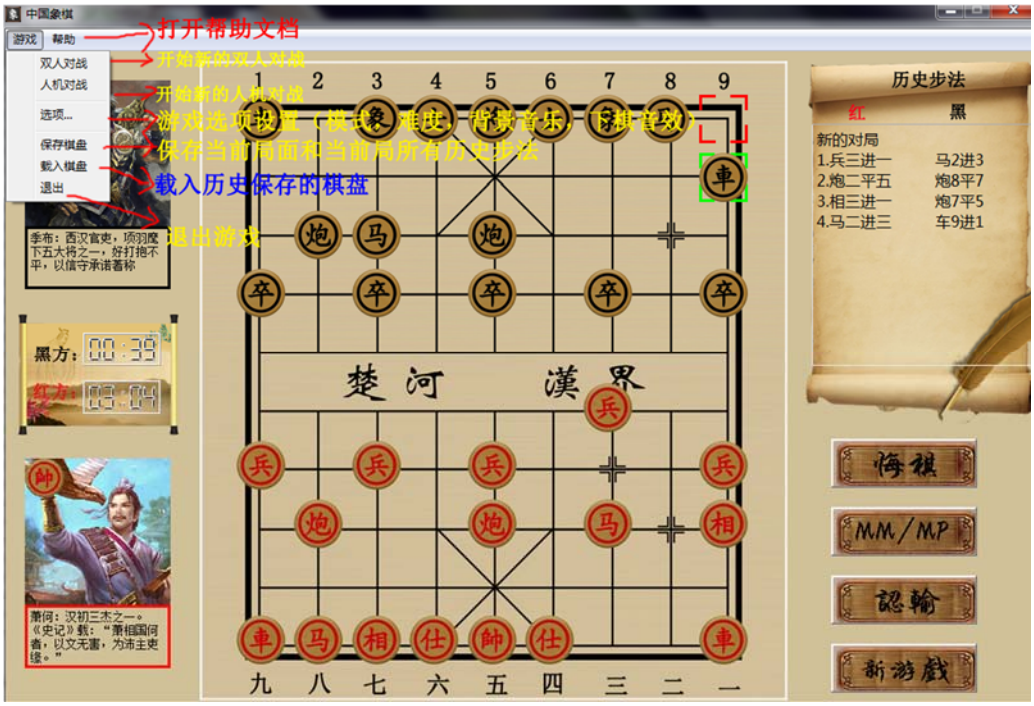
最终界面图：

启动界面如下图：

软件主界面，各功能区和按钮定义如下图：



软件的菜单栏按钮功能定义：



3.4 系统部件

以下是各个类的详解：

启动界面类(MySplashScreen)：

父类介绍：启动界面类是继承自 QSplashScreen，QSplashScreen 是 Qt 中用来显示启动界面的窗口部件，简单的 QSplashScreen 可以载入一组图片并显示它。要显示更多的功能，必须继承该类。我使用继承该类，并加入了进度条；

成员介绍：该类中只有一个成员变量，即进度条 QProgressBar；该类中的构造函数，参数是待显示的图片；

该类的两个成员函数，一个用于设置进度条的进度，一个用于在检测到进度条改变后重绘进度条，这两个成员函数通过信号槽机制绑定在一起，一旦设置进度，立即发出信号，槽接收到执行槽函数实现重绘。

主窗口类（MainWindow）：

父类介绍：MainWindow 是 Qt 中主窗口的基类，在该类中主要显示菜单栏，状态栏和中央窗口部件的设置。

成员介绍：首先是游戏菜单：包括双人对战，人机对战，选项，保存棋局，载入棋局，退出；然后是帮助菜单；然后在主窗口中需要有一个主游戏类 ChessGame。

棋子类（ChessMan）：

父类介绍：棋子类的父类是 QWidget。QWidget 类是所有用户界面对象的基类，基本需要显示的窗口部件，如按钮，窗口，进度条，MainWindow 等都是以它为父类。

成员介绍：该软件中，每颗棋子的成员变量包括：

chessManId：用 0-31 来表示棋盘上 32 颗棋子；

Type：棋子类型，由于在判断棋子的走法规则以及计算局面得分时，是依靠棋子棋子类型的，

因此加入棋子类型可以简化后面的表示；
Killed:用来判断棋子是否被吃掉，被吃掉的棋子不会显示；
Selected：用来判断棋子是否被选中，被选中的棋子将会画上绿色边框；
Position：棋子的位置，用于确定画棋子的位置
该类中只有一个重载的 void *paintEvent*(QPaintEvent *)函数，用于根据棋子的 killed，selected 和 position 画出棋子。

设置选项类（ConfigDialog）：
父类介绍：其父类为 QDialog，QDialog 类是对话框窗口的基类。
成员介绍：主要设置成员为游戏模式，电脑等级，背景音乐开关，下棋音效开关。

该类中主要是一些按钮，单选框和下拉框。确定按钮和取消按钮各绑定一个信号槽。当用户点击确定按钮后，会将用户当前的选择作为信号发送出去；当用户点击取消按钮后，会发出一个使用默认设置的信号。

这些信号主要在主游戏类中接收，在第一次游戏时，如果是收到用户自定义信号，即设置当前模式为用户选择；如果收到默认设置信号，即将游戏设置为人机对战，中等难度，音乐全开。在游戏过程中点开设置选项后，如果没有改变游戏模式，点击确定按钮设置为用户选择但不会重新开始游戏，如果点击取消不做任何改变。

秒表类（StopWatch）：
父类介绍：QWidget
成员介绍：QLCDNumber *lcdNumber 使用液晶显示屏类来显示时间；
定时器类 QTimer 用来计时；QTime 类用来记录时间。

```
void dispTime();          //显示时间函数
void resetTime();         //重置时间函数，将 QTime 重置为 0
void startTime();         //开始计时函数，启动计时器
void stopTime();          //停止时间函数，停止计时器
void addTime();           //每 1000ms 后收到信号，QTime 时间加 1s
```

秒表主要使用液晶显示类来显示时间。使用定时器 Timer 计时，Timer 每计到 1000ms，发出一个信号，this 收到信号后，会调用 addTime 函数，将时间 Time 类型实例 time 加 1， 更新 LCD 显示。

主游戏类（ChessGame）：
父类介绍：QWidget
成员介绍：主游戏类是最复杂的一个类，不是算法复杂，是其包含的内容多。因此仅对重要函数作说明。
1.游戏参数的设置：

当用户启动软件或者点击设置好会弹出设置选项对话框，这些在选项设置类里面已经说明过；

2.棋盘初始化函数：

该类中使用 chessBoard[10][9]来表示棋盘上的每一个点。棋盘初始化的时候会使用棋子类型来初始化棋盘数组，如下所示（B 为黑色，R 为红色）：

```
{B_CAR,B_HORSE,B_ELEPHANT,B_BISHOP,B_KING,B_BISHOP,B_ELEPHANT,B_HORSE,B_CAR},
```

{NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS},
{NOCHESS,B_CANON,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,B_CANON,NOCHESS},
{B_PAWN,NOCHESS,B_PAWN,NOCHESS,B_PAWN,NOCHESS,B_PAWN,NOCHESS,B_PAWN},
{NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS},
{NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS},
{R_PAWN,NOCHESS,R_PAWN,NOCHESS,R_PAWN,NOCHESS,R_PAWN,NOCHESS,R_PAWN},
{NOCHESS,R_CANON,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,R_CANON,NOCHESS},
{NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS,NOCHESS},
{R_CAR,R_HORSE,R_ELEPHANT,R_BISHOP,R_KING,R_BISHOP,R_ELEPHANT,R_HORSE,R_CAR}

在初始化时，用棋盘数组来初始化棋子类型，并让棋子 id 从 0 累加，用 for 循环遍历中的 chessBoard[i][j]的 i，j 来初始化棋子位置，同时用棋子的 id 来更新棋盘数组。这样初始化完毕之后棋盘上存储的是棋子 Id。且 Id<16 的为黑方棋子，大于 16 为红方棋子，-1 为没有棋子。

3.鼠标点击事件

我在主类中定义了一个 selectedId 的 int 型变量来存储选中棋子的 id，在点击棋盘，执行伪码如下：

If（没有棋子被选中 && 点击棋子 i d 与当前下棋方颜色相同）
　　选中该棋子并重绘；
e lse if（(点击的是空白处 &＆ 选中棋子颜色与下棋方相同） || （点击的是对方棋子 && 选中棋子与下棋方相同））
　　调用走子函数；
else if(点击本方棋子 && 选中为本方棋子)
　　取消前一个棋子选中状态，选中当前棋子；

4.棋子走步规则判断函数：

```
bool canMoveToDst(const int board[10][9],QPoint& from,QPoint& to)
```

注：由于该函数在主游戏类，走法产生器和局面评估函数中都需要使用，因此我将其定义到所有类公用的头文件中，但我仍在此处解释。

首先排除终点不在棋盘范围内，起点和终点为同色棋子的情况；
然后棋子按走子规则分为以下几类：车，马，炮，相或者象，士或者仕，将或者帅，兵或者卒。

为了简化判断，我们首先将**横走一步的权重设为 10，纵走一步的权重设为 1**，总的权重为 10x + y；车不考虑权重，马权重为 21 或者 12，炮不考虑权重，相或者象权重为 22，士或者仕权重为 11，将或者帅权重为 1 或者 10，兵或者卒权重为 1 或者 10；

这样对于某个特定棋子，在棋盘上搜索可以走的位置时，首先对权重进行判断，权重不对的直接淘汰。权重正确的再判断其他限制，各棋子的其他限制如下：
车：起点和终点之间棋子数是否为 0；
马：马脚处是否有棋子；
相：象眼处是否有棋子，以及终点是否过河；
炮：起点和终点之间棋子数为 0 且终点为空格，或者 起点和终点之间棋子数为 0 且终点为对方棋子；
仕：双方士是否在各自的家内；
将：双方将是否在各自的家内；

兵卒：在自己家内是只能直着向前走，过了河可以前进和左右走

注：此处需要说明的是对于将军会面的处理方式，将军会面的那一步棋是可以走的，但哪一方主动产生使将军会面的情况，哪一方判输；

5.电脑引擎搜索线程

一开始我没有另开一个线程供电脑搜索最佳步法，发现我走完一步之后轮到电脑走的时候会变得非常卡，这是因为搜索占用了主线程，因此后来我将电脑搜索另开一个线程处理。这样的话需要考虑线程间数据传递的问题。

因此需要将当前棋局作为参数传入新开线程供电脑计算，并且使用信号槽机制，一旦电脑搜索完成，立即发出搜索完成的信号，并且将搜索得到的步法作为参数传回，主线程收到信号后，调用电脑下棋方法完成电脑思考过程，即：

```
connect(computerThread[num],SIGNAL(computerMoveSignal(const int,const int,const int,const int)), this,SLOT(computerRun(const int,const int,const int,const int)));
```

在这个过程中，需要注意两个问题：一是电脑在思考线程中不能让用户有其他的操作，比如用户想悔棋，这个时候是不能响应的，必须等到电脑下棋完成，这可以通过设一个电脑思考状态为解决；第二个是线程的创建和复用问题，一开始我是想只用一个新开的线程，在电脑思考完成后马上结束线程，但是后来发现 即便我发出了线程终止的命令 terminate()，线程也不会立即终止，因此我最后使用的方法是每次轮到电脑下棋都新开一个线程，这就涉及到第三个问题，线程间数据的同步，我的软件功能基本正确，但偶尔也会出现电脑不动的情况，这是因为棋盘没有及时更新，电脑算出的仍是上一次局面的最佳步法。

6.悔棋函数

悔棋函数的实现比较简单，在每次走子时，都将步法记录到一个 Vector 数组中，每次在悔棋时 ，先确认电脑不在思考中，然后从数组中取出上一次的棋步，然后更新棋盘和棋子，再将该步法从 Vector 中删除。

7.新游戏函数

新游戏时，先删除原来的 32 颗棋子，然后调用初始化棋盘方法，即可。

8.保存棋盘

先通过打开一个文件存储位置对话框并获取存储文件名称，然后依次将轮到谁下棋标志 turn，所有棋子的状态信息，Vector 步法数组中所有走过的步法 写入文件中即可。

9.载入棋盘

按写入文件的顺序重新载入这些变量并设置他们即可。

10.播放背景音乐

播放背景音乐时，由于我已经使用了 QSound 用来播放下棋时的音效，因此不能再在下棋音效上叠加别的声音（只能播放一个），因此我使用 Phonon::MediaObject 多媒体类来播放音乐，并且为了不干扰主线程我新开了一个线程播放。

步法生成类（ChessMoveGenerator）：

父类介绍：无

成员介绍：主要定义了一个二维数组 moveList[8][80]，第一维是搜索的深度，即在当前局面下走的步数，第二维是每一步可能的走法。

然后定义了 7 个成员函数，依次产生帅/将，兵卒，车，马，象，仕，炮的走法，并将这些走法加入走法数组。

在走法产生函数中，对于每一颗棋子，分别遍历其可能达到的位置，然后调用前面提到的判断能否移动的函数，如果能移动将其加入步法数组。如象的遍历如下：

```
//产生相/象的走法
void ChessMoveGenerator::generateElephantMove(const int board[10][9],QPoint &from,int nPly)
{
    int i = from.y();
    int j = from.x();
    //插入右下方的有效走法
    to.rx()=j+2;
    to.ry()=i+2;
    if(canMoveToDst(board,from,this->to))
        AddMove(from,this->to,board[from.y()][from.x()],nPly);

    //插入右上方的有效走法
    to.rx()=j+2;
    to.ry()=i-2;
    if(canMoveToDst(board,from,this->to))
        AddMove(from,this->to,board[from.y()][from.x()],nPly);

    //插入左下方的有效走法
    to.rx()=j-2;
    to.ry()=i+2;
    if(canMoveToDst(board,from,this->to))
        AddMove(from,this->to,board[from.y()][from.x()],nPly);

    //插入左上方的有效走法
    to.rx()=j-2;
    to.ry()=i-2;
    if(canMoveToDst(board,from,this->to))
        AddMove(from,this->to,board[from.y()][from.x()],nPly);
}
```

局面评估类（ChessEvaluation）：

父类介绍：无

成员介绍：动态局面的评估需要考虑所有棋子的所有信号。如下图所示

```
protected:
    int baseValue[15];                  //分别存放每一颗棋子基本价值
    int flexibilityValue[15];           //分别存放每一颗棋子灵活性分值
    int threadPosition[10][9];          //存放每一位置被威胁的信息
    int guardPosition[10][9];           //存放每一位置被保护的信息
    int flexibilityPosition[10][9];     //存放每一位置上棋子的灵活性分值
    int chessValuePosition[10][9];      //存放每一位置上棋子的总价值
    int relatedPosCount;                //记录一棋子的相关位置个数
    QPoint relatePos[20];               //记录一棋子的相关位置
```

我们设定每颗棋子的基本价值，每种类型是不同的，兵 100，士 250，象 250，车 500，马 350，炮 350，王无穷大;对于兵卒而言，没过河跟过了河也是不同的，过了河的兵卒是要加分的。

同时设定每颗棋子的灵活性分值，也就是每多一个可走位置应加的分值，兵 15，士 1，象 1，车 6，马 12，炮 6，王 0。

然后我们遍历棋盘数组，对于某一颗棋子，先搜索他能达到的点，这就是它的相关位置，对于遍历每一个相关位置，如果当前位置是空白，灵活性增加；如果当前位置是本方棋子，本方棋子被保护的分值增加；如果当前是对方棋子，当前棋子被威胁增加，即分值减少。

这样遍历之后，统计每个棋子和位置的分值，得到某一步法总的得分。

置换表类（TranspositionTable）：
父类介绍：无
算法分析：当棋弈引擎开始对一个局面进行分析时，它经常是“试验”以不同次序去走但到达同样局面的棋，程序于是把这样的局面以及对局面的评价值保存在内存中，一旦碰到以别的走棋次序但到达同样局面的变化时，换句话说，当计算“另一个”变化但到达的局面其实之前已经出现过时，程序就省下了再次估值的时间了。

哈希表用在棋弈程序中，作用很大。举例，计算如何走棋时，我可能先走马后走车，也可能先走车后走马，假如对手各自相对应的应着不变，那么不同次序走完两步棋后到达的局面是一模一样的。

如果我已经对局面进行了计算估计，那么再次出现相同的局面就可以节省时间。哈希表的用途就是在这种情况下迅速查找之前已经完成了的工作(已经计算过的估值)。
我使用的哈希表中元素的定义如下：

```
//哈希表中元素的结构定义
typedef struct HASHITEM
{
    long long checksum;      //64位校验码
    ENTRY_TYPE entry_type;   //数据类型
    short depth;             //取得此值时的层次
    short eval;              //节点的值
}HashItem;
```

把一个 64 位的数据校验码，加上数据的类型（主要为精确，上边界，下边界），局面的估值和当前估值的搜索深度作为一个局面评估的唯一标识，这样，每次进行评估前，我先搜索哈希表，看表中的局面是否与当前要求相符，相符就不用在此估值。从而节省时间。

成员介绍：该程序中使用了 32 位和 64 位哈希表，并分别用随机数初始化他们，随机数可以减小错误的匹配。每进行一步操作后，会调用根据所给走法，生成新的哈希值的函数（void HashStepMove(ChessStep* move,int currentBoard[10][9])），并计算当前局面的哈希值（void calculateBoardHashKey(int currentBoard[10][9])），然后查找当前局面是否在哈希表中（int lookUpHashTable(int alpha, int beta, int depth,int tableNum)），若查找到直接返回局面的估值（若为确切值，返回确切估值，若为搜索边界，返回边界）；若找不到，将当前局面存入哈希表（void TranspositionTable::enterHashTable(ENTRY_TYPE entry_type,short eval,short depth,int tableNum)）；最后撤销当前走法（void TranspositionTable::undoHashStepMove(ChessStep* move,int chessManId,int currentBoard[10][9]))。整个哈希表的搜索过程结束。

历史启发类（HistoryHeuristic）：
父类介绍：无
算法分析：历史启发是迎合 alpha-beta 搜索对节点排列顺序敏感的特点来提高剪枝效率的，即对节点排序，从而优先搜索好的走法。

国际象棋程序的经验证明，历史表是很好的走法排序依据。那么，什么样的走法要记录

到历史表中去呢？我的算法中为：

A. 产生 Beta 截断的走法；

B. 不能产生 Beta 截断，但它是所有走法(v > Alpha)中最好的走法。

一个走法记录到历史表，我使用的是经验值——深度的平方。

成员介绍：该类中主要实现的操作有：

```
void mergeSort(ChessStep* from, int n,bool direction);
//对当前走法队列进行归并排序
void enterHistoryScore(ChessStep *move,int depth);
//将某一最佳走法汇入历史记录表
int getHistoryScore(ChessStep *move);
//取某一走法的历史得分
void resetHistoryTable();
 //将历史记录表清空
```

搜索引擎基类（）：

父类介绍：无

成员介绍：这是一个搜索引擎的基类，该类中声明了一个用于被子类实现的纯虚函数：

```
virtual void searchOneGoodMove(const int board[10][9]) = 0; //搜索最佳棋步
```

这个引擎包含了搜素引擎的基本元素，即走法生成器（ChessMoveGenerator），估值核心（ChessEvaluation）。

空窗探测引擎类（NegaScout_TT_HH_Engine）：

父类介绍：SearchEngine

算法介绍：AI 博弈的核心思想并不复杂，实际上就是对博弈树节点的估值过程和对博弈树搜索过程的结合。在博弈的任何一个中间阶段，站在博弈双方其中一方的立场上，可以构想一个博弈树。这个博弈树的根节点是当前时刻的棋局，它的儿子节点是假设再行棋一步以后的各种棋局，孙子节点是从儿子节点的棋局再行棋一步的各种棋局，以此类推，构造整棵博弈树，直到可以分出胜负的棋局。

博弈程序的任务就是对博弈树进行搜索找出当前最优的一步行棋。对博弈树进行极大极小搜索，可以达到这一目的。极大极小搜索，是因为博弈双方所要达到的目的相反，一方要寻找的利益恰是一方失去的利益，所以博弈的一方总是希望下一走是儿子节点中取值最大者，而另一方恰恰相反。这便形成了极大极小过程。

在这个过程中，最为重要的是搜索算法，高效的搜索算法可以保证用尽量少的时间和空间损耗来达到寻找高价值的走步。

对于极大极小过程的搜索过程中，并不需要处理所有的分支，通过一定的原则，可以削减部分分支，而不影响计算结果。

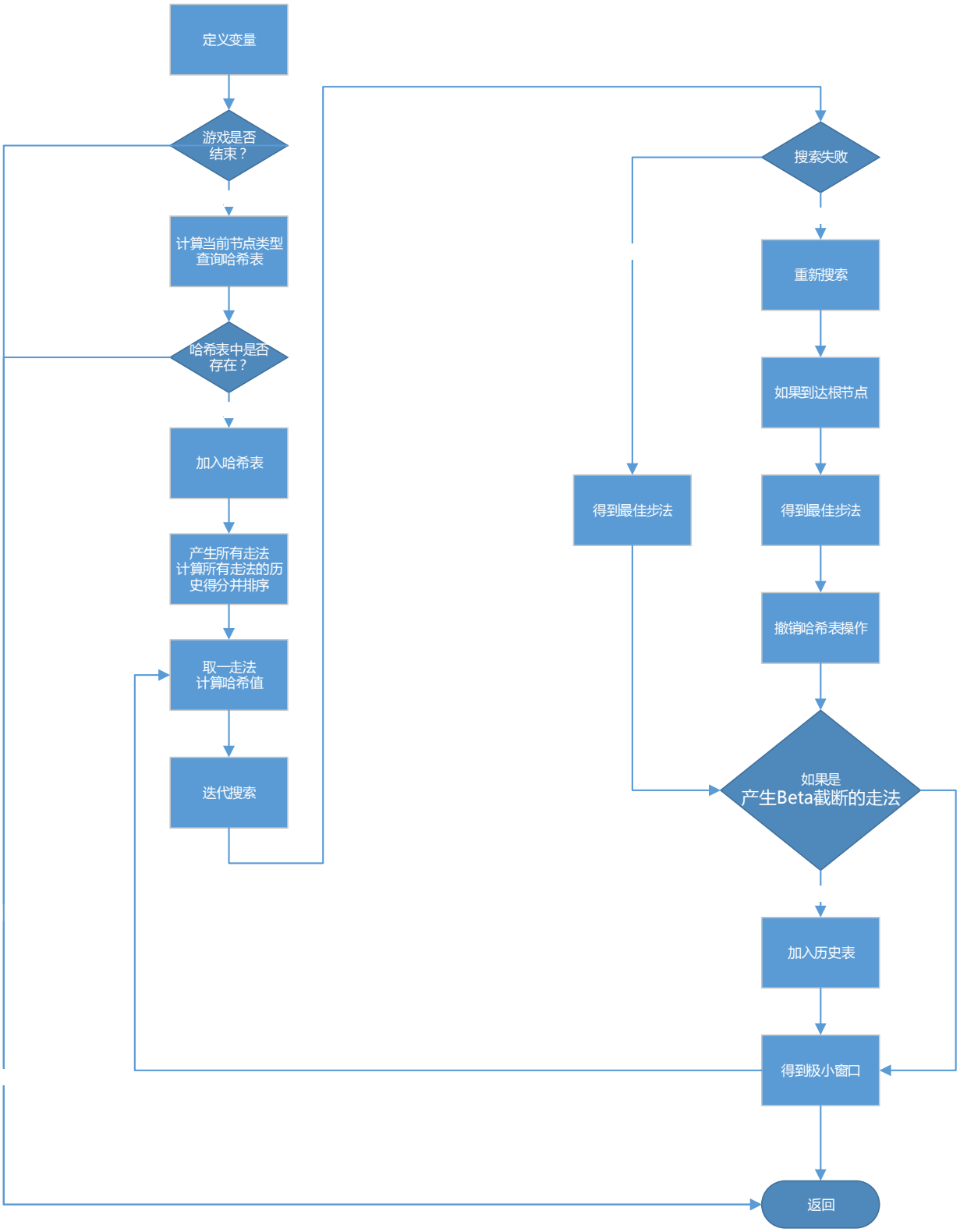
NegaScout 算法也被称为极小窗口搜索算法或 PVS 算法，它的原理是用极小的窗口来限制剪枝范围。过程如下：

在根节点处，假定第一个儿子节点为主变量，也就是假定它为最佳走步，对它进行完整窗口(a,b)的搜索并得到一个返回值 v，对后面的儿子节点依次用极小窗口（也被称为是零窗口）(v,v+1）来进行搜索，如果搜索返回值大于零窗口，则证明这一分支亦为主变量，对它进行窗口为（v+1,b）的搜索，可是如果返回值小于零窗口，这一分支就可以忽略，因为它的最佳走步还不如已有的走步。

极小窗口搜索采用了极小的窗口，剪枝效率最高，并且只对主变量进行大窗口的搜索，

所以大部分搜索动作是有效的，搜索产生的博弈树也很小。

成员介绍：该类中主要方法是对搜索引擎基类的搜索步法虚函数的重载。该函数的搜索过程执行如下：



3.5 系统出错处理设计

(1) 开始新游戏后不能悔棋：解决方法是在每次开始新游戏后将存储历史步法的数组 Vector 清空，点击悔棋时先判断 Vector 是否为空，为空直接 return；

(2) 保存棋盘时，如果用户点击保存棋盘，然后又点击取消，此时应该不保存棋盘，但我的原来的程序仍然会将文件名从 mainWindow 传到 ChessGame 主类中：解决办法是给 FileDialog 的确定按钮添加响应事件，如果用户点击的是确定按钮，才保存棋盘；

(3) 线程不同步导致的 AI 计算错误，有时出现 AI 走的是玩家的子：解决办法是在每次电脑计算出应该走的步法之后都对此步法进行判断是否正确，如果不正确重新搜索；

(4) 需要同时播放两种不同的音乐，即背景音乐和下棋音效，但 QSound 无法完成：解决办法是使用两个不同的类来播放，即 QSound 和 Phonon。

(5) 需要背景音乐循环播放，同时不干扰主线程：解决办法是新开一个线程播放背景音乐，但是我无法解决循环播放的问题，我尝试使用信号槽，并找了很多方法，都没有实现。

(6) 软件偶尔会出现玩家走完一步后，电脑不能继续下棋的现象，遇到此情况，请重新开始游戏或者重启软件。

(7) 偶尔出现死机的ＢＵＧ，需要重启程序。

4. 感想与总结

本次实验是我有史以来写的最大的一个程序，开发时间也是最长，前前后后花了大概两个星期，中间也遇到了很多问题，现在终于完成了，虽然还是存在很多小问题。

一开始对 Qt 的开发过程并不是很清楚，闷着头在那里乱写，根本不知道写出来是什么意思，后来我静下心来好好找了一个《Qt 学习之路》教程开始看，一天之后我基本理解了 Qt 的应用程序框架，然后我好好的想了想象棋类该怎么划分，一开始就只有三个类，主窗口，主游戏和棋子，实现了基本的人机对战之后我再开始加入 AI，AI 成功之后我再进行界面的美化。这个过程中，我也明白了一定要细心，要写一部分跑一部分，把功能积少成多。

在开发的过程中，我也明白了很多以前不理解的知识，比如说用指针和不用指针构造类的区别,用指针构造的类需要手动释放内存,不释放会造成内存泄露,但指针对象更加灵活。在 Qt 里面，如果你把指针类指定父对象，那么在父对象消失时同时也会释放内存，这样也就不需要你手动释放了。但是这些指针一定要管理，非 QWidget 类中的指针一定要记得释放，毕竟软件的内存占用是很重要的指标。

同时在 Qt 的开发过程中我养成了良好的编程风格，比如把所有的公用常量和公用方法定义到一个公共的头文件中便于管理和修改（chessdefine.h）。

同时我知道了类的设计是很重要的，一开始我没有划分好类，导致两个类需要互相调用，就很尴尬，而且会报错。后来我的类基本上都是独立的，在主类中调用。

目前，我的软件最大的问题还在于 AI 不够强大，鉴于时间有限，我现在只是采用了带置换表和历史启发的空窗探测搜索法。现在我的 AI 表现在开局走的较差，将军局面处理较差，有时会出现死将等。开局的问题可以通过加入开局库来解决，即事先存储好很多经典的开局，在开局时搜索相应的步法。应外还需要加入杀手着法和长将判负策略等手段来提高 AI

棋力，暂时没有时间去修改它。

总之，我付出了很多，也收获了很多，要做的也还有很多。

最后总结一下本系统设计的几个关键点：

（1） 类的划分
（2） 内存管理
（3） AI 算法实现
（4） 代码风格（注释和常量宏定义）
（5） 界面风格
（6） 认真的态度