实验报告

一、绘制棋盘

首先学习了如何做一个windows窗口（定义窗口类结构（[WNDCLASS](http://bbs.fishc.com/thread-47123-1-1.html)） -> 注册窗口类（RegisterClass） -> 创建窗口（[CreateWindow](http://bbs.fishc.com/thread-47139-1-1.html" \t "_blank)） -> 显示窗口（[ShowWindow](http://bbs.fishc.com/thread-47206-1-1.html" \t "_blank)） -> 更新窗口（[UpdateWindow](http://bbs.fishc.com/thread-47210-1-1.html" \t "_blank)） -> 消息循环（[GetMessage](http://bbs.fishc.com/thread-47252-1-1.html" \t "_blank) -> [TranslateMessage](http://bbs.fishc.com/thread-47258-1-1.html) ->[DispatchMessage](http://bbs.fishc.com/thread-47260-1-1.html)）），学习了一些常用的API函数，此过程达到了绘制棋盘的目的。

二、人人对战

先用一个二维数组来储存棋盘所有位置的信息，0代表空位，1代表白棋，2代表黑棋，通过L-buttondown和光标来确定走子位置，每次走子后判断游戏是否结束（检查行、列、对角线、斜对角线）

三、AI算法设计

1、创造一个函数，传入当前棋盘信息，返回一个坐标（最优解）

2、对每个空位进行评分，评分最高即为最优解。

评分函数1.0

首先收集了所有的基本棋型辑对应分数并转化为字符串数组储存，其次遍历棋盘（遍历方式与判断输赢函数相似），把该位置范围内的落子情况也以字符串的形式储存，最后使用字符串对比函数（strcmp）得到该位置的价值并返回。  
评分函数2.0

修复bug：有些棋型是非对称的，因此在使用字符串对比函数时，增加一步，将字符串逆序，再对比一遍

评分函数3.0

增添了防守分数，弥补了先前不考虑对方先行的不足，具体实现方式：

将黑白子互换，得到一个防守分数，先前的进攻分数减去防守分数才是该位置的价值

评分函数4.0

增添了组合棋型额外加分，比如两个活三，两个眠四等，评分更加合理

评分函数5.0

将原有的字符串对比的方式更改为累计实时评分，即在遍历该位置范围内的棋盘的同时赋分，极大地减少了内存消耗，极大地加快了运行速度

3、博弈树的建立

遍历所有空位，我方先模拟走一步，对方再模拟走一步，如此循环三次，然后选出当前最高价值位置，返回给上一层，我方模拟落子为MAX层，对方模拟落子为MIN层，由底层向上返回，max层返回最高价值位置，min层返回最小价值位置（因为我方选择对自己最有利的位置，对方选择对我方最不利的位置）由此可以得到一个眼光较为长远的走法。

不足之处：对内存的消耗较大，计算量随博弈树层数增加而指数级增长，改进措施：alphabeta剪枝

每个博弈树节点额外增加alpha和beta两个值，alpha为下一层的最大值，beta为下一层的最小值，max层更新上一层的beta值，min层更新下一层的alpha值，每层向下同步alpha和beta值，当alpha》beta时，不必在遍历该节点（不会再有更好的位置了），由此实现了博弈树的剪枝，大大减小了计算量，进而增加了博弈树的层次，提高了棋力。

第二次改进：先通过贪心算法得到每个空位的初步价值，排序并储存在单链表中，这样就可以给博弈树的顶层传入一组有序的数据，以此来实现尽早剪枝的目的，计算量进一步减小。

第三次改进：很多位置其实一眼就能看出来价值很小不必再进行博弈树的计算，取通过贪心算法得到的链表中排名前十的位置传入博弈树顶层，进一步减少了计算量。

第四次改进：有些博弈树的节点和上一轮相同，存在重复计算的现象，因此可以把当前计算得到的棋盘价值信息储存起来，并进行排序，这样下一次走子的速度就会变快。

AI棋力的主要取决因素：博弈树层数和评分函数的合理性。

今后改进方向：1.训练AI函数，从而不断调整各棋型的分数。

2.储存一些经典棋局，减小计算量

3.增加算杀模块，先判断有无可形成死局，如有可能直返回该位置，这样算可以增加博弈树层数，眼光更加长远。