1.0

做什么：下五子棋，目前可以实现先手后手的选择（但是黑白子不会换，只是改变了下棋顺序），对对局进行存档和读取（目前只能存下一局的数据）

怎么做：调用的库主要是c语言标准 I/O 库，比较特别的是easyx的库，头文件还引用了windows.h。

代码整体分成了三个源文件和一个头文件，其中头文件里是包含了所有的声明和引用，为了利于使用便把这些代码放在头文件里。game.cpp文件里面放的主要是和游戏运行有关的函数定义，比如判定胜负的函数、开始界面的动画、游戏内按键的交互（包含选择先后手、存档、和落子）、游戏内功能等使游戏可以运行的函数（但是里面不包含ai，只是单纯刨除ai后的整个游戏逻辑的实现）。这一部分代码实现的逻辑就是将落子的位置放在数组里面（因为多维数组和棋盘和坐标有很大的相似性，可以更加方便的将棋子表示出来，也很容易对棋盘进行遍历（虽然耗费很大）），然后先将棋盘打印出来，在棋盘的位置上利用坐标关系相应打印出棋子。

test.cpp只有一个main函数，是将game.cpp里的函数实现的一个平台（相当于game.cpp里的函数是基石，test.cpp就是搭建出来的平台的外观），这个源文件的用意主要是在按照游戏逻辑搭建游戏里的基础函数时便于即时运行（就是写完一个函数可以直接运行看看有没有问题，不用全部写完再来找问题，写完之后也就懒得删掉了）。

最后一个ai.cpp里面就是实现ai的估值等函数，其中给局势打分的函数写了两种（只用了一种，另一种还没舍得删）：一种是根据五子棋的一些行内局势的术语（如活三、眠四）来将连子的情况进行穷尽（用一个容量为五的一维数组将所有可能定义下来），随后利用两个循环对棋盘上每一个棋子进行估值（就是将数组正中央的区域空出来，然后遍历棋盘和数组进行一一比较），另一种就是根据遍历棋盘的每一个位置，根据位置上下左右五个棋子的位置来进行打分。目前我选用的是第一种，因为在实战中第一种的表现明显优于第二种，但是由于局势后期的复杂性，第一种方法并不灵活，会有误判局势的可能，而且数组容量有些小，导致虽然降低了穷举连子的情况的数量，但是在五子相连的情况出现时经常会被误判成四子相连的情况。

反思：比较的函数采用的是dfs的算法来遍历每一个位置的黑白棋的得分（其中有利得正分，无利得负分），目前估值的第二种方法在面对层数增多的情况时表现有所提升，但是还是不如第一种，第一种方法增加层数后表现直线下滑，目前还在思考原因。在搜索遍历中目前还没有采用剪枝的算法（主要现在看了很久但是还没有学会…），再尝试中反思感觉应该是数据结构选择数组来进行遍历体现不出决策里树的结构，可能用树的数据结构，通过链表来完成这一方面代码会更好一些。

后续更新：这次更新尝试了阿尔法贝塔剪枝的工作，但是最后代码改造失败了，ai只会在偏僻点落子，不符合预期，最终在遍历过程中采用了剪去偏僻节点（如外围位置）的遍历。本次工程把之前没有删掉的代码进行了清理，最终重新编写了估值的一系列代码，但是感觉ai的棋力有所减弱，攻击性变强但是同时防守能力被大幅削弱，同时本次更新修复了一些之前没注意过的bug（如先手如果抢占中间位置，ai会进行强行覆盖）。

随后增加了悔棋的功能，这个功能的实现利用了处理文件这方面的函数，实际上是每下一步棋就记录一次下棋的位置，如果选择悔棋就会将悔棋那一步之前已经保存好的存档再打印上来。

这次在网山找到了[(44条消息) AI五子棋第二篇-运用极大极小值算法书写AI三子棋，可拓展到五子棋（建议收藏）\_Ja\_King\_ZH的博客-CSDN博客](https://blog.csdn.net/Ja_king_/article/details/119295652?spm=1001.2014.3001.5506)这样一篇文章，我模仿里面的代码写了我的版本的极大极小值的剪枝（层数是两层的），其中全局评估函数还是借用了一下文章的代码（偷了个小懒），剪枝代码我阅读完他的代码后发现了之前自己对于这一算法的一个误区，像上文中对自己的代码介绍中提到的，我给棋局评估的思路是对每一步的棋子来获取得分，这样在评估时可以不用两次遍历，损耗就少了一点，但是它的本质我个人感觉更像是贪心算法，就是求取局部的最优解，而更好的方法应该是求全局的最优解，也就是求取下了好几步后的局面，而不是一步一步求解，这也可以解释为什么当我固定连子的情况时下棋的表现更佳，而在用回溯算法的时候表现很差。目前ai棋力就是突飞猛进了，但是我注意到源代码实际上还对偏僻处进行了剪枝，这个我还在尝试理解源代码的思路。