黑白棋

王冠杰PB19081611

实验内容

黑白棋(reversi), 又叫苹果棋, 翻转棋, 是一个经典的策略性游戏. 一般棋子双面为黑白两色, 故称"黑白棋". 因为行棋之时将对方棋子反转, 变成己方棋子, 故又称"翻转棋". 棋子双面为红, 绿色的称为"苹果棋". 它使用8*8棋盘, 由两人执黑子和白子轮流下棋, 最后多方为胜方. 随着网络普及, 黑白棋作为一种最适合在网上玩的棋类游戏正在逐渐流行起来. 游戏规则略.

实验要求

- 使用MCTS算法实现miniAlphaGo for Reversi
- MCTS算法部分需要自己实现, 尽量不使用现成的包, 工具或者接口.
- 博弈过程中miniAlphaGo每走一步所花费的时间以及总时间需要显示出来.
- 需要有简单的图形界面用于人机博弈交互

实验思路及代码实现

黑白棋游戏主体

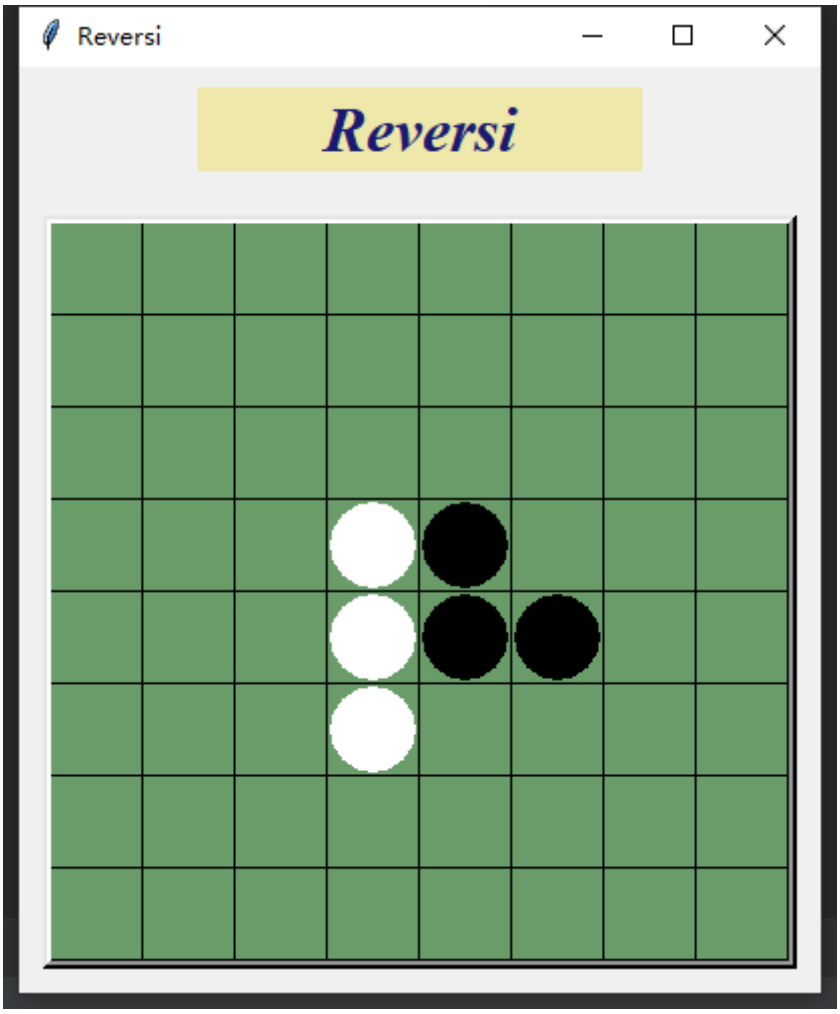
黑白棋游戏主体是由Reversi类来控制，其中调用了封装好的ReversiBoard类来实现对于棋盘的实现

```
class Reversi(Tk.Frame):

    def __init__(self, master=None):
        Tk.Frame.__init__(self, master)
        self.master.title("Reversi")


        # title
        l_title = Tk.Label(self, text='Reversi', font=('Times', '24', ('italic',
'bold')), fg='#191970', bg='#EEE8AA', width=12)

        l_title.pack(padx=10, pady=10)

        self.f_board = ReversiBoard(self)
        self.f_board.pack(padx=10, pady=10)
```

由于需要调用的函数较多限不一一列出，进列出函数名与对应的功能：

- put_stone：落子
- ai：MCTS算法实现的AI棋手
- getInitBoard：得到初始棋盘
- copyBoard：复制当前棋盘
- countStone(board,turn)：对输入棋盘状态，统计对应turn的棋子数量
- checkPlaceblePosition(board,turn)：获得输入棋盘状况下，对应turn能够放置棋子的位置
- updateBoard(board,turn,i,j)：检查将棋子在当前状态下（board和turn）放置在（i,j）处是否合理，如果可以放置则更新棋盘

用户UI




```
pointed: ( 5 , 4 )
loop time:  3.004997730255127
MCTS's answer:  (5, 3)
```

棋盘和AI棋手所用时间分开显示，用户默认执黑棋

会在下方显示AI棋手使用的总时间以及选择的落子点

MCTS实现查找下一步最优落子点

MCTS分为四个部分, 选择, 扩展, 模拟, 反向传播. 具体实现代码较多，并且伪码也在教材中给出，下面着重讲一下一些根据实际情况进行的处理和微调。

1. 查找深度限制：最大查找深度只有32层，这是为运行效率的考量
2. 探索实践限制：除了深度限制外也需要对探索实践进行限制，我这里选用的时间是3s

实验结果分析与总结

自己实现了MCTS算法, 也使用tkinter做了游戏的用户界面, 有计算和显示miniAlphaGo每一步所花费的时间. 但是由于python本身速度较慢, 查阅资料得知一般MCTS的过程是使用C++写的加速, 再使用python与C++对接. 但考虑到C++的速度约是python的20~30倍, 即C++的MCTS算法可能在python允许的时间下可以迭代模拟10000到20000次. 在数据统计上相对棋力会更高一些, 但对于初局和中局的棋盘局面来说,

从理论分析效果也非常一般.

但是从个人多次下黑白棋来进行测试得到的规律来说，我认为可以实际上黑白棋的一些落子位置与别的位置其实是有区别的。比如当棋子占据四个角落时，由于这四个点能够更有效的翻转更多对手棋子并且永远不会被翻转，所以导致了占据四个角落后的胜率会更高。

因此可以考虑一个1/4棋盘的每个位置其实是有不同的启发信息的，讲这些启发信息引入AI模型中，应该能够使得MCTS得到更好的搜索结果