Mini AlphaGo

1. Project Introduction

内容包括：（这部分内容不要太长，讲清楚即可）

（1） 开发环境及系统运行要求，包括所用的开发工具、开发包、开源库、系统运行要求等；

开发环境：Windows10

系统运行要求：Python 2.7

开发工具：VS2015

Pycharm

（2） 工作分配简介，即谁要做什么事情

陈凌昊：主要代码设计

来舒悦：测试+报告

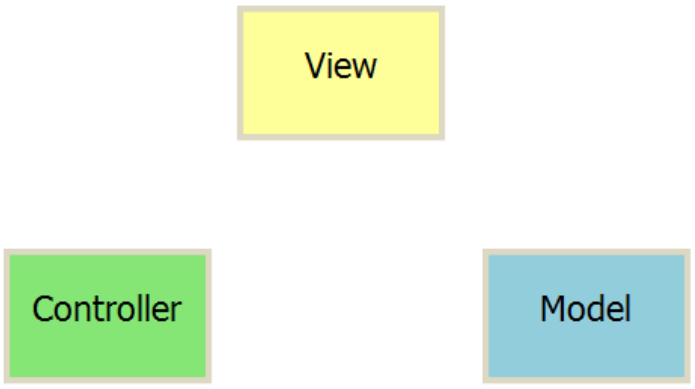
2. Technical Details

内容包括：

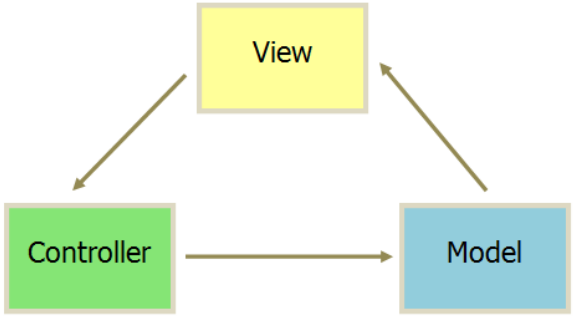
（1） 工程实践当中所用到的理论知识阐述

- **设计模式——MVC**

MVC 模式的意思是，软件可以分成三个部分。



各部分之间的通信方式如下。



- 中间适配器函数

```
#coding=utf-8
import Tkinter
def handler(event, a, b, c):
    """事件处理函数"""
    print event
    print "handler", a, b, c
def handlerAdaptor(fun, **kwds):
    return lambda event,fun=fun,kwds=kwds: fun(event, **kwds)
if __name__=='__main__':
    root = Tkinter.Tk()
    btn = Tkinter.Button(text=u'按钮')
# 通过中介函数 handlerAdaptor 进行事件绑定
    btn.bind("<Button-1>", handlerAdaptor(handler, a=1, b=2, c=3))
btn.pack()
    root.mainloop()
```

- 文件读入写出

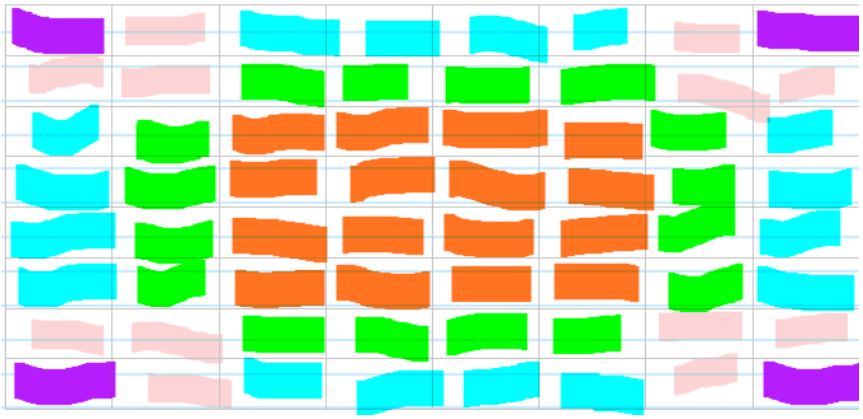
```
1 # dumps 功能
2 import pickle
3 data = ['aa', 'bb', 'cc']
4 # dumps 将数据通过特殊的形式转换为只有 python 语言认识的字符串
5 p_str = pickle.dumps(data)
6 print(p_str)


1 # loads 功能
2 # loads  将 pickle 数据转换为 python 的数据结构
3 mes = pickle.loads(p_str)
4 print(mes)
5 ['aa', 'bb', 'cc']
```

```
1 # dump 功能
2 # dump 将数据通过特殊的形式转换为只有 python 语言认识的字符串，并写入文件
3 with open('D:/tmp.pk', 'w') as f:
4     pickle.dump(data, f)

1 # load 功能
2 # load 从数据文件中读取数据，并转换为 python 的数据结构
3 with open('D:/tmp.pk', 'r') as f:
4     data = pickle.load(f)
```

- 优先级

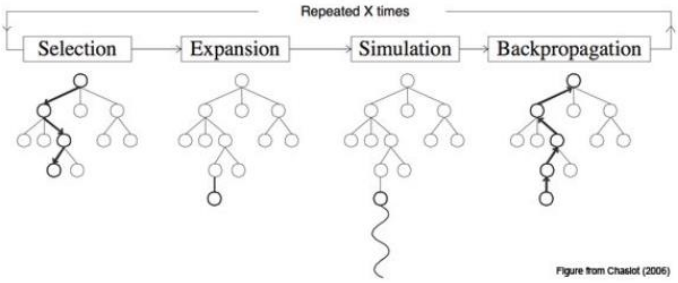


在不同的位置棋子具有不同的优先级。如上图，相同颜色的棋格具有相同的优先级。

- **蒙特卡洛树搜索 MCTS**

简介： 全称 Monte Carlo Tree Search，是一种人工智能问题中做出最优决策的方法，一般是在组合博弈中的行动（move）规划形式。它结合了随机模拟的一般性和树搜索的准确性。MCTS 受到快速关注主要是由计算机围棋程序的成功以及其潜在的在众多难题上的应用所致。超越博弈游戏本身，MCTS 理论上可以被用在以 {状态 state，行动 action} 对定义和用模拟进行预测输出结果的任何领域。

基本算法： 根据模拟的输出结果，按照节点构造搜索树。其过程可以分为下面的若干步。

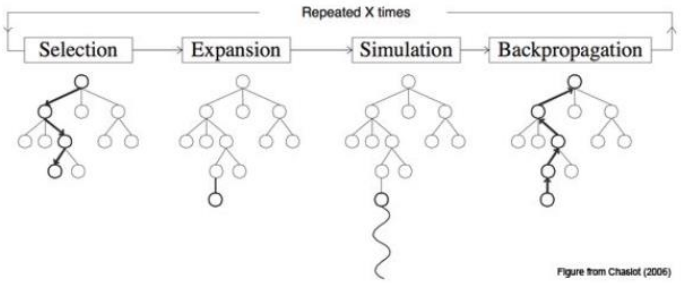


构建过程：

1. 选择 Selection：从根节点 R 开始，递归选择最优的子节点（后面会解释）直到达到叶子节点 L。
2. 扩展 Expansion：如果 L 不是一个终止节点（也就是，不会导致博弈游戏终止）那么就创建一个或者更多的字子节点，选择其中一个 C。
3. 模拟 Simulation：从 C 开始运行一个模拟的输出，直到博弈游戏结束。
4. 反向传播 Backpropagation：用模拟的结果输出更新当前行动序列。

（2） 具体的算法，请用文字、示意图或者是伪代码等形式进行描述（不要贴大段的代码）

- **蒙特卡洛树搜索 MCTS**



如上图所示，本次实验的主要算法有 MCTS 实现，共分为四个步骤。其中较为重要的是 simulation 和 backup 部分。

模拟 Simulation：从 C 开始运行一个模拟的输出，直到博弈游戏结束：

```
def do_simulations(self, node):
        """
        multiprocessing simulation.
```

```
        :param node: root node
        :return: count
        """
        count = 0
        self.time_limit = timedelta(seconds=min(single_time_limit, 62 - fabs(34 - self.moves) * 2))
        while datetime.utcnow() - self.start_time < self.time_limit:
            vl = self.tree_policy(node)
            delta = self.default_policy(vl)
            self.backup(vl, delta)
            count += 1
            if node.is_fully_expanded():
                break
        if len(node.children) == 0:
            return count
        # use multi-thread policy, refering github
        pool = ThreadPool(len(node.children))
        counts = pool.map(self.simulation, node.children)
        pool.close()
        pool.join()
        count += sum(counts)
        return count
```

反向传播 Backpropagation：用模拟的结果输出更新当前行动序列：

```
    def backup(self, node, reward):
        """
        back up.
        :param node: Node
        :param reward: reward:1 or 0
        :return:
        """
        while node is not None:
            node.n += 1
            if node.player == self.player:
                node.q += reward
            else:
```

```
                node.q += 1 - reward
            node = node.parent
```

• MVC 模式

在本次实验中，除了核心 MCTS 算法外，另一核心是采用了 MVC 的设计模式，使得整个实验更加清晰，代码结构更加优化。

Controller：

用来完成对整个实验的逻辑控制，其核心为两个下棋策略代码。

玩家下棋：

```
    def player_play(self, event):
        x, y = self.get_position(event)
        valid_list = valid.get_valid_list(self.board.matrix, globals.player_color)
        if len(valid_list) != 0:
            # 点的位置不在有效列表里,即点的位置不能下棋
            if (x, y) not in valid_list:
                return
            move(self.board, x, y, globals.player_color)
            # 删除提示
            self.board.valid_matrix = []
            self.notify()
        else:
            print("valid_list is empty! player pass")
            if not self.board.has_empty_box():
                self.finish_game()
        self.switch_player((x, y))
```

AI 自动下棋：

```
def AI_play(self):
        if globals.state != State.AI:
            return
        valid_list = valid.get_valid_list(self.board.matrix, globals.AI_color)
        (x, y) = (None, None)
        if len(valid_list) != 0:
            try:
                single_step_begin = time.time()
                (x, y) = self.tree.uct_search()
```

```
                    single_step_end = time.time()
                    self.total_time += single_step_end - single_step_begin
                    print('Time used: ', single_step_end - single_step_begin)
                    print('AI: ', y + 1, x + 1)
                except Exception:
                    print(self.tree.root)
                    # print()
                move(self.board, x, y, globals.AI_color)
            else:
                print("valid_list is empty! AI pass")
                if not self.board.has_empty_box():
                    self.finish_game()
            self.board.valid_matrix = valid.get_valid_list(self.board.matrix,
globals.player_color)
            self.notify()
            if len(self.board.valid_matrix) == 0:
                print("player pass!")
                self.tree.update(self.board, globals.AI_color, (x, y), force_add=True)
                self.AI_play()
            else:
                self.switch_player((x, y))
            pass
```

Model：

为数据模块，其主要功能是通过更新 matrix 即矩阵，来代表当前棋局状态。

```
    def __init__(self):
        self.chess_cnt = 4
        self.matrix = []
        self.valid_matrix = []
        for i in range(row):
            self.matrix.append([])
            for j in range(col):
                self.matrix[i].append(None)
        self.matrix[3][3] = white
        self.matrix[4][4] = white
```

```
            self.matrix[3][4] = black
            self.matrix[4][3] = black


    def has_empty_box(self):
            for i in range(row):
                for j in range(col):
                    if self.matrix[i][j] is None:
                        return True
            return False
```

View：

主要包含了两个模块，分别是 canvas 以及 game。其中 canvas 即画布，整个实验的 GUI 实现，而 game 模块则是实现用户交互，即点击反应。

canvas

```
def repaint(self, model):
        self.create_rectangle
        draw the intermediate lines
            Horizontal line
            Vertical line
        draw circles
        draw prompt
```

Game

```
def on_canvas_click(event, controller):
        controller.on_click(event)


        pass
```

（3） 程序开发中重要的技术细节，比如用到了哪些重要的函数？这些函数来自于哪些基本库？功能是什么？自己编写了哪些重要的功能函数？等等

- **GUI**

  在本次实验中，GUI 使用了 python 的标准库 tkinter 来实现。

- **文件信息的读入和写出**

  在本次实验中，文件信息的读入和写出使用了 pickle 来实现。

```
    @staticmethod
```

```
def read(filename):
    with open(filename, 'rb') as f:
        aa = pickle.load(f)
        return aa


def write(self, filename, t=None):
    with open(filename, 'wb') as f:
        if t is None:
            t = mcts.MCSearchTree(self.board, black)
        pickle.dump(t, f)
```

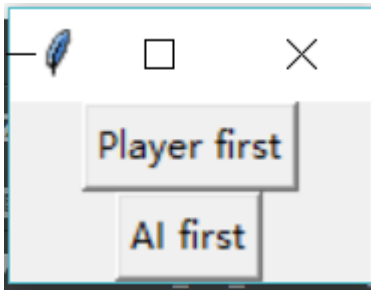
3. Experiment Results

用图文并茂的形式给出实验结果，如系统界面、操作说明、运行结果等，并对实验结果进行总结和说明。

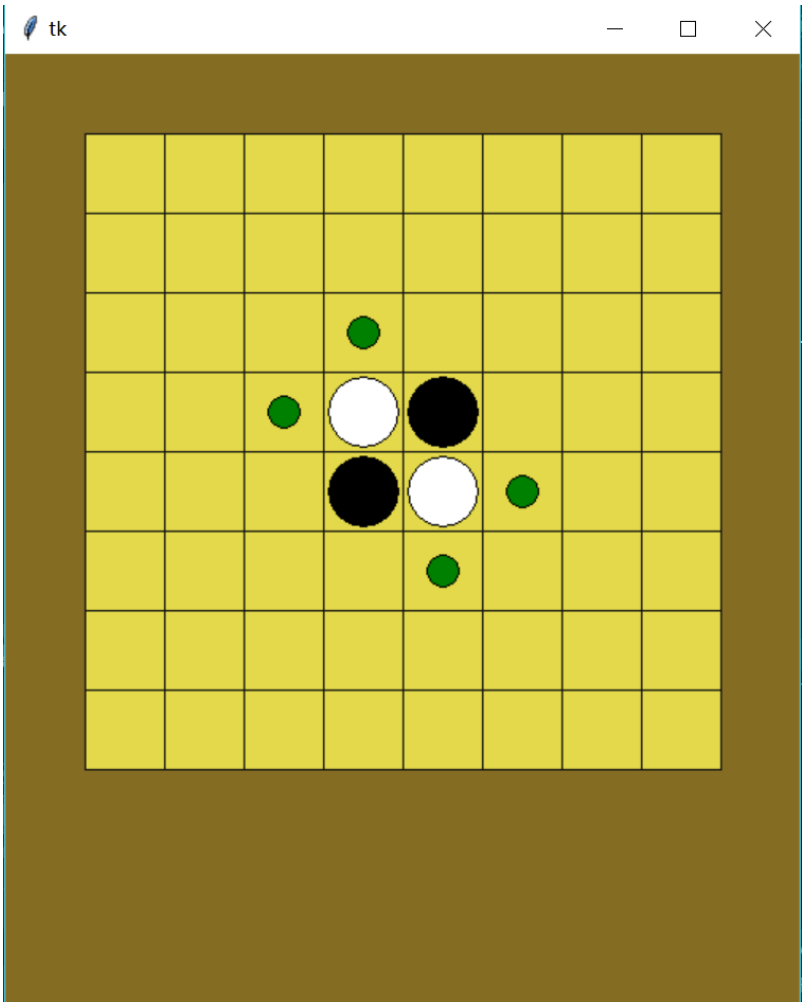
操作说明：本次实验可以直接在 windows10 下通过 pycharm 直接运行 main.py 进行运行。

系统界面：

开始界面：

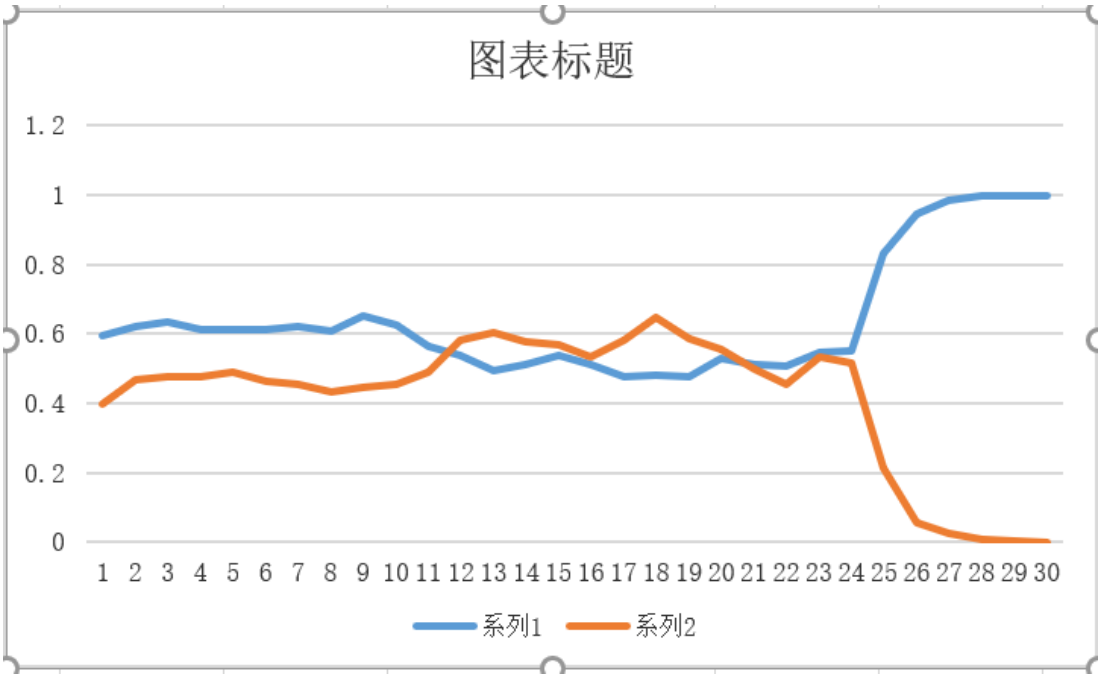


游戏界面：

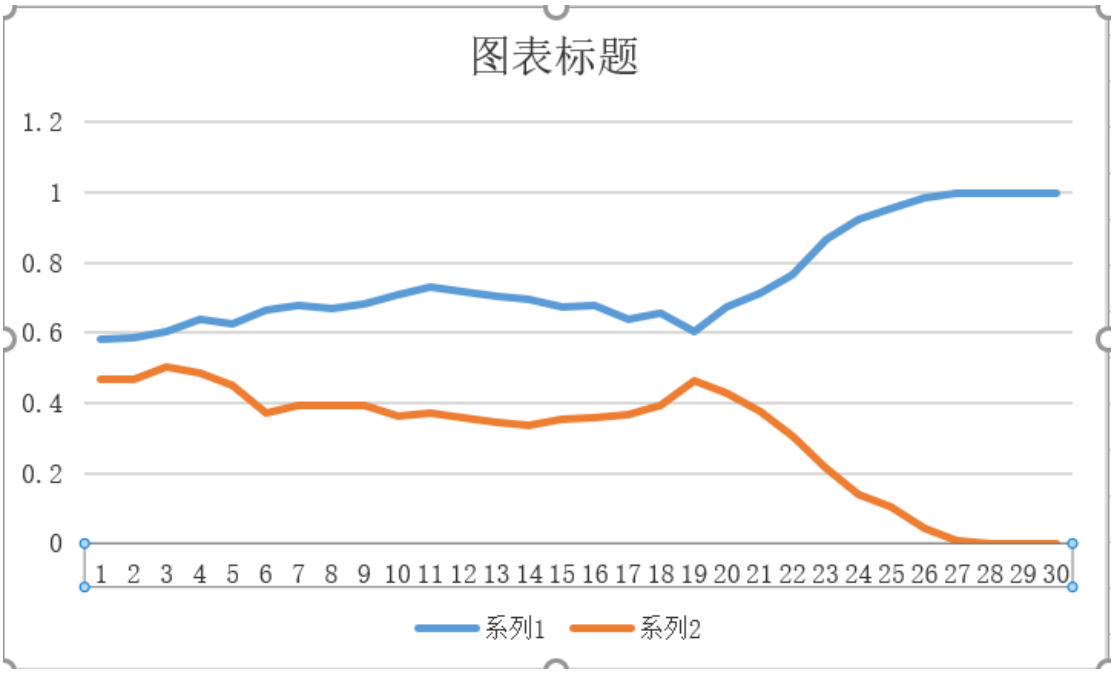


运行结果：（具体数据请查看 result.xls 文件）

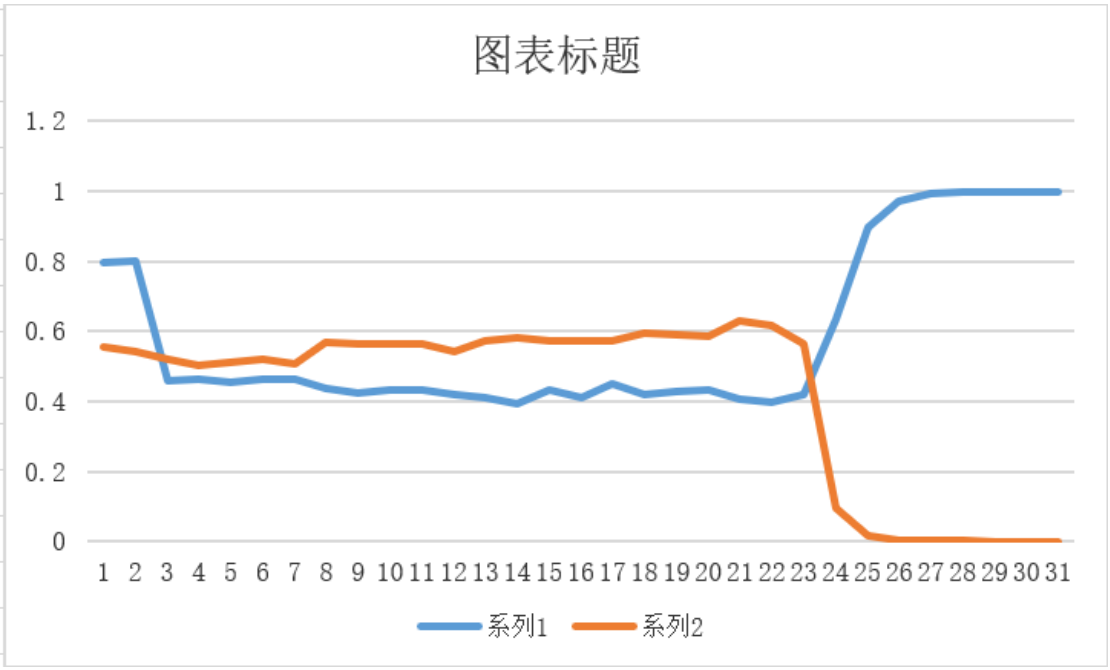
（1） 代码自己和自己对战结果：



（2） 本组 player_first vs 他人代码



（3） 本组 ai_first vs 他人代码



运行结果：在实际训练过程中，胜率一开始随着训练次数增加而有明显提升，后期提升效果较缓慢。

References:

给出主要的参考文献，可以是论文、网站、书籍、别人的技术报告等。

• 蒙特卡洛树搜索 MCTS:

https://zhuanlan.zhihu.com/p/25345778

https://zhuanlan.zhihu.com/p/26335999

https://zhuanlan.zhihu.com/p/34990220

・RKinter

https://www.cnblogs.com/collectionne/p/6885066.html

・MVC

https://blog.csdn.net/tinym87/article/details/6957438

备注：

代码中请给出较为详细的注释，此报告中切勿粘贴大量代码，否则扣分。