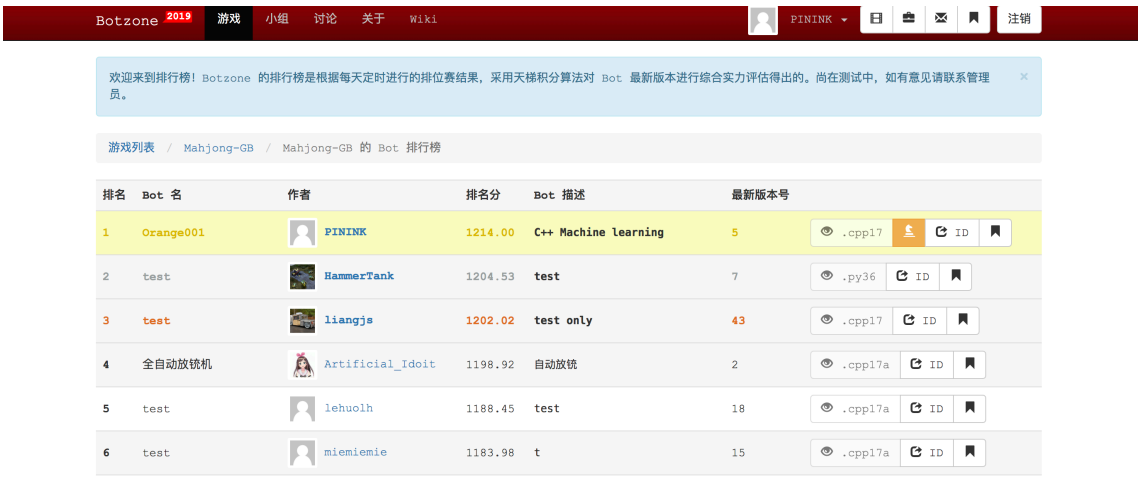
麻将期末报告

张迅 1600017704

倪临赟 1600017719

概述

在本次麻将期末大作业中，我们原计划先用C++写一个基于规则的AI作为baseline，再转用Python使用深度学习进一步优化。在C++基于规则的AI中，我们修改了ChineseOfficialMahjongHelper中basic_form_shanten计算上听数不考虑八番起和的问题，修改后的第一版向上听数贪心就进入了天梯前20。我们接着加入剩余牌记录，计算摸到有效牌的概率，并加入和绝张、抢杠和，改进后的版本排名稳定在天梯前十并获得模拟赛第七，甚至在一段时间内到达天梯第一。基于规则的C++程序的表现远超预期，而Python深度学习的训练模型融入C++代码的难度也比我们想象的大，因此我们调整计划将主要精力放在基于规则的C++程序上。



天梯第一截图

所用名词解释

- 顺子：数牌中，花色相同序数相连的3张牌。

- 刻子：三张相同的牌。碰出的为明刻，未碰出的为暗刻。俗称坎。杠也算刻子，明杠算明刻，暗杠算暗刻。

- 副露：吃牌、碰牌、杠牌的统称，即利用其他选手打出的牌完成自己手牌面子的行为 。

- 听牌：只差所需要的一张牌即能和牌的状态。

- 上听数：达到听牌状态需要牌的张数。
- 有效牌：能使上听数减少的牌 。
- 雀头：基本和牌形式中，单独组合的对子。

手牌、副露和剩余牌的记录

用vector<string>hand, fulu分别记录手牌和副露，用tile_table_t Table记录剩余牌，同时维护一个PrevPlayedCard变量记录上一张打出的牌。对于每个request，首先判断是否是自己摸牌（itmp==2），如果是则将摸到的牌加入hand中，该牌的剩余张数减一。如果不是自己摸牌的回合，分为对手回合和自己回合讨论。如果是对手回合只需更新剩余牌表，如果是自己回合需要更新手牌、副露和剩余牌表。此处踩过的坑有：1、PrevPlayedCard这张牌在之前打出时已经从剩余牌表中删去，因此碰的时候应该再减去两张而非三张，吃的时候要判断顺子的三张牌中哪张是PrevPlayedCard，不能重复减。2、对手回合杠时可能是明杠或暗杠，要判断上一回合是不是对手摸牌（DRAW），如果是对手摸牌则是暗杠，无法得知杠的是哪张牌，因而不需要更新剩余牌表，如果是明杠则需在剩余牌表中减去三张PrevPlayedCard。3、自己回合杠时更新副露时应指明是明杠还是暗杠，在ChineseOfficialMahjongHelper中[CCCC]是暗杠，[CCCC,1]则是明杠，若明杠时漏了“,1”则可能导致多计算双暗刻等番数。

决策算法

由于ChineseOfficialMahjongHelper提供了非常高效合理的数据结构和完善的算番库，因而我们的决策算法是基于ChineseOfficialMahjongHelper的改进版本。在本节我们会先简单介绍原始代码的不足，并介绍我们实现的改进方案。

原始ChineseOfficialMahjongHelper 的问题

在ChineseOfficialMahjongHelper的源码中有basic_form_shanten函数用于计算基本和型的上听数，但是该函数存在以下弊端：一是其仅仅是简单的一个递归计算，缺乏对叶子节点是否为满足八番起和番型的判断；二是在计算对应牌型的有效牌时，采用了模拟加入该牌之后重新递归计算上听数的方法来获得对应的有效牌表，效率较

为低下。因而在我们的决策算法中首先对basic_form_shanten进行了改进。

改进的basic_form_shanten

1、满足八番起和

在改进的basic_form_shanten中，我们会维护一个pack_t类型的数组来表示当前递归状态下已经枚举出的副露和面子，并且会记录如果是搭子的话相应的缺牌。之后对于递归函数的叶子节点进行如下处理：1、如果对于叶子节点的副露数、面子数和搭子数的总和小于4，会根据当前剩余的未成对牌型组成相应的可能面子或搭子，并且进行雀头补全。 2、传入Makeup_hu函数，对于搭子进行进一步处理，具体为根据当前剩余牌数来判断是否可以满足使搭子成为刻子或顺子，如若均可能，则返回完整的目标听牌牌型和期望和牌，并且将其传入calcluate_fan函数进行番型计算。 3. 返回所计算的番值，如若大于8，叶子节点正常返回当前上听数，否则返回极大值。

对于牌型：[B1 B1 B1 T4 T5 T6 T7 T8 T9 B3 B3 B3 J1 J1]，假设自摸且门清，门风和圈风均为东，实际番数为10番（不求人 4 双暗刻 2 连六 1 幺九刻1 缺一门 1 单钓将 1），满足和牌要求。而如果[B1 B1 B1]出现在副露中，番数变为5番（连六 1 幺九刻1 缺一门 1 单钓将 1 自摸 1），不满足和牌要求。原始的basic_form_shanten无法区分这两种牌型，而改进的basic_form_shanten可以成功计算出当前满足八番的上听数，而对于不满足八番的上听数时，由于上听数+1后可能会有满足的番型，因而返回的上听数比原始版本不考虑番数的basic_form_shanten中计算的上听数多1。

2、有效牌计算

原始的有效牌计算为重复调用basic_form_shanten 34次，如若加入新牌的上听数小于原牌，则将该新牌视为有效牌。 此计算十分冗杂并且费时。幸运的是，在我们所提到的满足八番起和的改进中，我们需要对叶子节点的枚举结果进行自动补全并且计算番型，所使用的自动补全的牌显然即为该番型的有效牌。因而我们在自动补全的同时，维护Useful_Table数组，记录当自动补全后所计算的番型大于等于八番时所使用的自动补全的牌。该Useful_Table即为有效牌表。

如表格所示，在进行改进之后，程序耗时明显下降，由于之后在计算概率时需要多次调用该函数，这样的效率提升是十分有帮助的。

牌型	原始耗时	改进后耗时
123456789s123pCC	0.0664394s	0.0021072s
[111p]456789s333pCC	0.0319034s	0.0014175s
112233445566sSS	0.0699876s	0.0017961s

决策函数

在我们的决策函数中，会先调用之前改进的basic_form_shanten函数获得当前上听数和有效牌表，之后根据有效牌表和当前剩余牌表计算概率，来获得减少上听数的概率，并根据该概率来进行决策。 在测试版中，我们还测试了计算两层减少上听数的概率，即利用摸到有效牌后再摸到一张有效牌的概率来进行判断，这样的好处是对于当一次上听数和有效牌概率相同时，可以更深一步的计算来进行更优的抉择，而不是随机选择一个。但不幸的是，该测试版会经常存在超时的现象，因而在我们参加比赛的版本中并未使用。

算两层的bot决策超时

吃碰杠函数

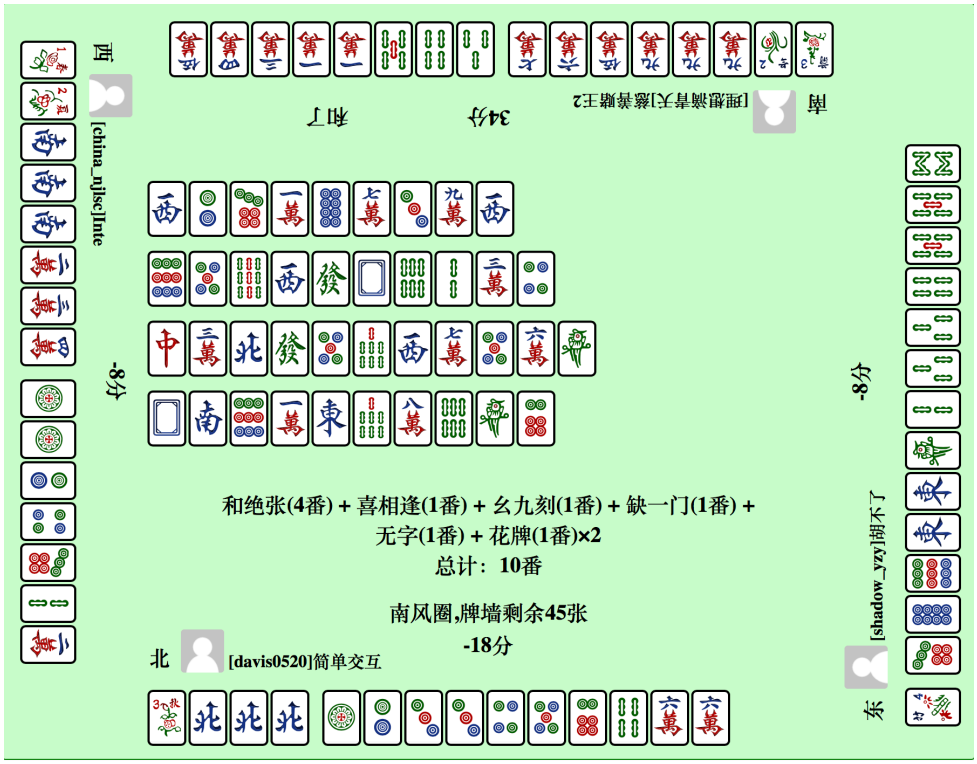
对于吃碰杠的判断，与决策函数基本类似。对于可能可以吃碰的牌，加入手牌后

调用basic_form_shanten来进行判断，如果上听数减少则说明应当执行该吃碰操作。对于杠，因为在执行杠操作后因为会再摸上一张牌，我们先枚举所有可能摸上来的牌，并计算相应的上听数。如果摸牌后使上听数减少或相等的概率大于使上听数增加的概率，就进行该杠操作。 因为杠往往可以增加番数，在上听数可以保持相等的情况下进行杠操作也是一种人为经验下的好策略。

和绝张与抢杠和

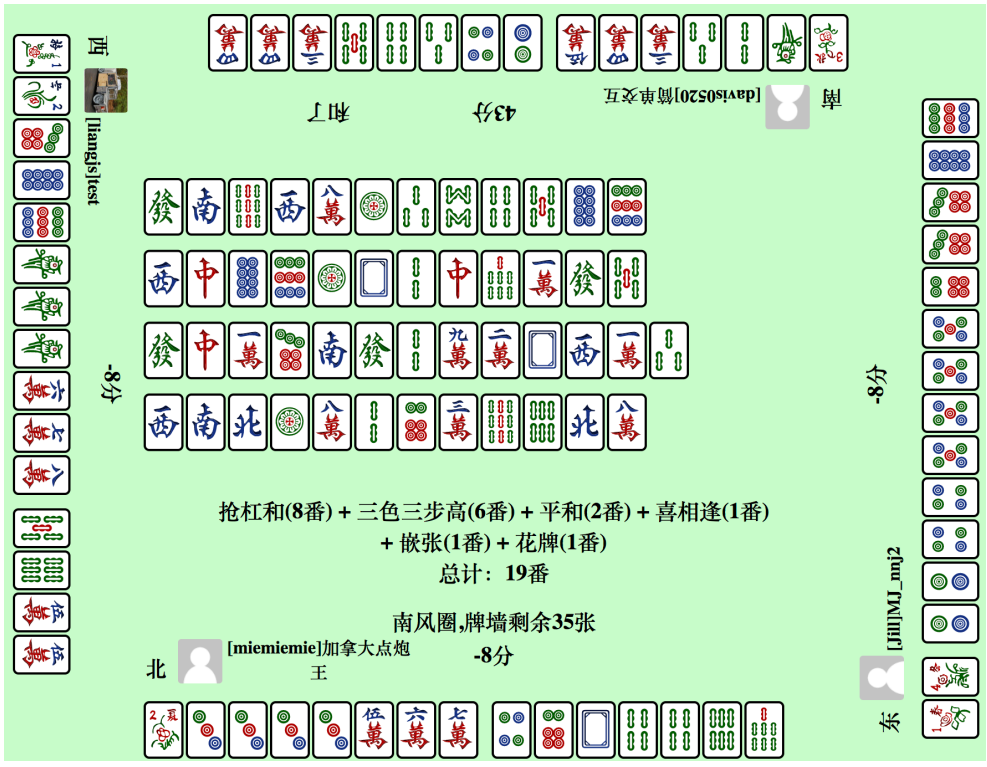
和绝张和抢杠和的判断代码量很小却十分有效。和绝张只需要在传入check_hu函数前判断剩余牌表中该牌张数是否为零，如果是绝张则加一句win_flag |= WIN_FLAG_4TH_TILE命令即可。抢杠和只需要在对手补杠时将win_flag改为WIN_FLAG_DISCARD | WIN_FLAG_ABOUT_KONG传入check_hu函数即可。在我们bot的历史对局中至少有超过十局依靠和绝张的四番凑满八番，还出现过多次牌面已经和牌但不够八番时故意打出一张牌造成绝张，在别人跟打相同牌时和牌的情况。这种情况的多次出现也是我们最终没有采取防守策略的原因之一。

依靠和绝张凑满八番



番数不够时故意造成和绝张局面（三万）

抢杠和在历史对局中也至少出现过五次。值得一提的是Botzone在出现抢杠和时，和牌方的分数结算正确，但其余三人分数都是-8，补杠的人没有按点炮处理导致四人总得分不为零，同时显示的手牌也会出现错误，建议Botzone修改此bug。



抢杠和，Botzone分数计算有bug

更多讨论

1、放弃防守策略的原因

由于国标麻将没有振听规则，同时和绝张有四番，因此别人打过的牌也不是安全的。再加上三色三步高、三色三同顺的普遍存在，相关牌防守的方法其实也是不科学的。我们又观察我们的bot点炮的对局，基本都发生在自己听牌的时候。而国标麻将对于未点炮的人也有-8分的惩罚，同时对自摸有较大的分数奖励，所以我们认为在听牌时弃和防守不是明智的选择。我们起初也考虑过参考GitHub（https://github.com/GanjinZero/Tenpai_prediction）上预测日本麻将立直听牌的代码，在国标麻将的人类数据集上使用双层LSTM网络连接34维的Softmax层对每局游戏中和牌者所听的牌进行预测。但基于以上原因，加上python训练的模型融入C++代码难度较大，我们最终放弃了防守策略。

2、决策函数的问题

由于我们的决策函数通过是计算摸到有效牌的概率来进行决策，而非最大化和牌番数的期望，因此我们的 bot 会尽可能往快速和牌的方向进行决策，而非获得总分数最多。虽然在大多数情况下这两种决策的结果相差无几，但是获得总分数最多的决策可能会在某些时候放弃容易和的简单番型，而去构造更高的番型从而获得更高的分数，这也就导致我们在天梯这种对分数奖惩不高而重视输赢的对局中可以获得较高的排名，而在正式比赛计算总分时排名相对较低。