**国标麻将实验报告**

**一、数据处理方式：**

1. 初始化:

* 定义全局变量 obs, actions, 和 matchid 分别存储四个玩家的观察序列、动作序列和当前比赛的ID。
* 定义空列表 l 用于记录每场比赛处理后的观测-动作对数量。

2. 定义辅助函数:

* filterData: 过滤掉只有一个有效动作（ “Pass”）的观察状态。这些状态对于学习策略帮助不大。
* saveData: 确保每个玩家的观察和动作序列长度相匹配，然后将数据转换为NumPy数组并保存到一个以比赛ID命名的.npz文件中，同时清空观察和动作列表以准备下一轮数据。

3. 数据读取与处理循环:

* 逐行读取文本文件中的游戏记录。

遇到“Match”开始新的比赛，创建四个FeatureAgent实例，增加比赛ID。

* 忽略“Wind”行（因为会被单次且唯一地记录在Feature中）
* “Player”行根据不同的游戏事件（如Deal, Draw, Play, Chi, Peng, Gang, AnGang, BuGang, Hu）更新玩家的观察和动作。每个事件都会调用相应FeatureAgent的方法来处理游戏状态并更新玩家的观测或响应动作。对于某些事件还处理了其他玩家可能需要做出的响应（如Chi, Peng, Gang, Hu），确保所有相关动作都被记录。
* “Score”行触发数据过滤和保存过程。

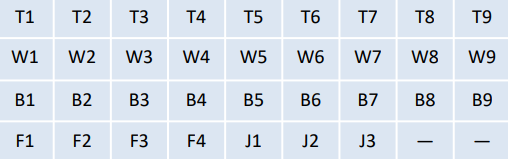
4. 数据后处理:

* 使用filterData去除单动作状态。
* 使用saveData将处理后的数据保存到文件，同时在data/count.json中记录每场比赛的数据量。

**二、特征表示方式：**

Observation为147\*4\*9的特征。

4\*9的特征图为按照如下方式对应麻将中的牌张，按照对应规则将每张牌对应维4\*9特征图中的单个元素。



转化代码如下：

*# 牌名列表*

    TILE\_LIST = [

        \*('W%d'%(i+1) for i in range(9)), *# 万*

        \*('T%d'%(i+1) for i in range(9)), *# 筒*

        \*('B%d'%(i+1) for i in range(9)), *# 饼*

        \*('F%d'%(i+1) for i in range(4)), *# 风*

        \*('J%d'%(i+1) for i in range(3))  *# 箭*

    ]

*# 牌名 -> 索引 的一个字典，这个索引的范围是 OBS 的第二个维度 36（实际上最后有空余）*

    OFFSET\_TILE = {c : i for i, c in enumerate(TILE\_LIST)}

147的通道数为门风1+场风1+己方手牌4+各方4\*(吃4+碰1+杠1)+己方暗杠1+各方4\*(弃牌历史28)+牌山4 ，具体说来：

座风/门风由本玩家的位次决定；

场风由数据中Wind 0..3一行决定；

接下来的一系列动作由Player P XX…决定，代表玩家P进行了XX行为，依照弃牌、吃、杠、碰、暗杠、补杠动作分别对对应通道进行更新，具体有：

己方手牌4：在进行发牌、弃牌等动作时进行更新，4代表麻将每种牌有4张。

吃4\*4：维护4位玩家的吃牌历史，在Player <N> Chi XX时更新（对应牌置为1）。由于每位玩家最多吃4次，因此通道数为4\*4。

碰4\*1：维护4位玩家的碰牌历史，在Player <N> Gang XX和Player <N> BuGang XX时更新，碰牌时增加碰牌特征（对应牌置为1）补杠时去掉碰牌特征（对应牌置为0）。由于每种牌最多碰1次，因此通道数为4\*1。

杠4\*1：维护4位玩家的明杠历史，在Player <N> Gang XX和Player <N> BuGang XX时更新，杠和补杠时增加杠牌特征（对应牌置为1）。由于每种牌最多杠1次，因此通道数为4\*1。

己方暗杠1：维护玩家自己的暗杠（抽到四张同样的牌）情况，在Player <N> AnGang XX且N为本玩家时更新。由于暗杠的牌只有自己知道，且每种牌只能杠一次，因此通道数为1。

各方弃牌历史4\*28：维护4位玩家的弃牌历史，在Player <N> Play XX时更新（打出XX牌，对应牌置为1）。

牌山4：维护当前不可知的牌特征，在各玩家弃牌和本玩家摸牌和吃碰杠时更新，尚未出现在场上的牌置为1，出现的置为0；由于每种牌有4张，因此通道数为4。

**三、动作空间表示方式：**

使用一个235维的action\_mask表示动作空间，具体为：

235 = 过1 + 胡1 + 弃牌34 + 明杠34 + 暗杠34 + 补杠34 + 碰牌34 + 吃牌63，其中吃牌 63 = 花色万条饼3 \* 中心牌二到八7 \* 吃三张中的第几张3。

通过一定的映射规则将动作映射为动作空间中的相应索引。代码如下：

def action2response(*self*, *action*):

        """

        将动作索引转换为对应的动作字符串。

        """

        if *action* < *self*.OFFSET\_ACT['Hu']:

            return 'Pass'

        if *action* < *self*.OFFSET\_ACT['Play']:

            return 'Hu'

        if *action* < *self*.OFFSET\_ACT['Chi']:

            return 'Play ' + *self*.TILE\_LIST[*action* - *self*.OFFSET\_ACT['Play']]

        if *action* < *self*.OFFSET\_ACT['Peng']:

            t = (*action* - *self*.OFFSET\_ACT['Chi']) // 3

            return 'Chi ' + 'WTB'[t // 7] + str(t % 7 + 2)

        if *action* < *self*.OFFSET\_ACT['Gang']:

            return 'Peng'

        if *action* < *self*.OFFSET\_ACT['AnGang']:

            return 'Gang'

        if *action* < *self*.OFFSET\_ACT['BuGang']:

            return 'Gang ' + *self*.TILE\_LIST[*action* - *self*.OFFSET\_ACT['AnGang']]

        return 'BuGang ' + *self*.TILE\_LIST[*action* - *self*.OFFSET\_ACT['BuGang']]

    def response2action(*self*, *response*):

        """

        将动作字符串转换为对应的动作索引。

        """

        t = *response*.split()

        if t[0] == 'Pass': return *self*.OFFSET\_ACT['Pass']

        if t[0] == 'Hu': return *self*.OFFSET\_ACT['Hu']

        if t[0] == 'Play': return *self*.OFFSET\_ACT['Play'] + *self*.OFFSET\_TILE[t[1]]

        if t[0] == 'Chi': return *self*.OFFSET\_ACT['Chi'] + 'WTB'.index(t[1][0]) \* 7 \* 3 + (int(t[2][1]) - 2) \* 3 + int(t[1][1]) - int(t[2][1]) + 1

        if t[0] == 'Peng': return *self*.OFFSET\_ACT['Peng'] + *self*.OFFSET\_TILE[t[1]]

        if t[0] == 'Gang': return *self*.OFFSET\_ACT['Gang'] + *self*.OFFSET\_TILE[t[1]]

        if t[0] == 'AnGang': return *self*.OFFSET\_ACT['AnGang'] + *self*.OFFSET\_TILE[t[1]]

        if t[0] == 'BuGang': return *self*.OFFSET\_ACT['BuGang'] + *self*.OFFSET\_TILE[t[1]]

        return *self*.OFFSET\_ACT['Pass']

**四、神经网络结构：**

Bot所用神经网络结构主要参照ResNet，由5个类Residual Block（含两个3\*3卷积层）、3层全连接层和一个Dropout层组成，概略如下：

1. 输入经过第一个Residual Block通道数变为128，再经过一个Residual Block提取特征。

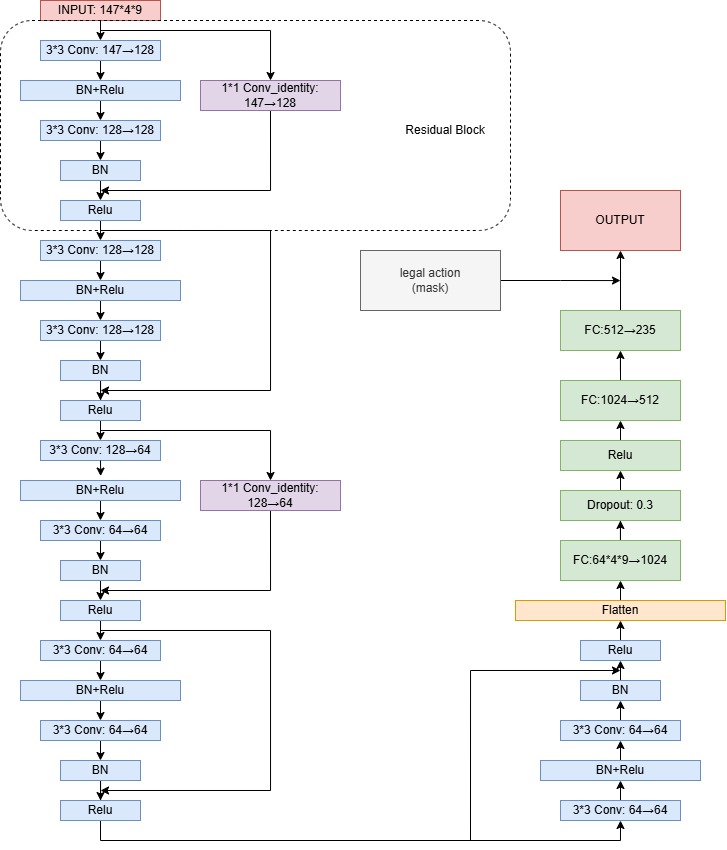
2. 经过第三个Residual Block通道数变为64，再经过两个Residual Block提取特征。

3. 通过Flatten层展平张量，便于全连接层处理。

4. 依次通过一层全连接层、一层Dropout层、Relu层、两层全连接层得到与动作空间相对应的235维张量。

5. 将得到的张量与原输入的action\_mask经过取对数并限制大小后得到的inf\_mask相加得到输出。

具体的神经网络结构如下图：



**五、训练超参数配置：**

数据集划分splitRatio：设置为0.9，意为有90%的数据将被用于训练集，另外10%将被用于测试集。

批量大小 batchSize：设置为1024，意为每次送入1024个样本进行训练。

学习率lr：使用Adam优化器，学习率设置为5e-4。

**六、训练技巧（重采样、接续训练等）：**

* 接续训练：体现在supervised.py中：识别目标目录中是否已存在指定训练若干epoch后的模型文件，如果存在则读取该模型参数进行接续训练。
* 可视化训练过程：体现在supervised.py中，使用tensorboard实时记录训练情况，便于调整网络结构和超参数。

**七、是否结合其他算法（强化学习、搜索等），以及结合的方法：**

无

**八、训练中发现的现象以及可能的解释：**

1. 训练的准确度上升，不代表Bot的有更好的对战能力。

可能原因：1) 训练时的准确度与对战能力无关，而与Bot的行为能否更好地拟合给定训练数据有关。如果给定的训练数据本身在某些情况并非最优解，可能导致Bot的行为也非最优；2) Feature提取是手工的，可能存在某些内在特征没有被提取到；3) 麻将相较于其他竞技游戏，可观测的信息更少，随机性更高，使得Bot训练出的行为并不一定能得到最优Payoff，从而表现得更差。

2. 当未改写Feature提取方式时，无论网络结构如何，训练出来的Bot总缺乏吃碰杠等复杂行为。

可能原因：作业默认提供的Feature提取只有六个通道：门风场风和自己的手牌，而缺乏他人的信息。在这种情况下，Bot训练出的行为可能更倾向于如何达成自摸，而不是其他更具有攻击性或防守性的行为（或可认为训练倾向如何找到纯何切问题的最优解）。

**九、训练中发现的问题：**

1. 采用作业默认模型训练时，模型欠拟合程度高，Bot表现差。

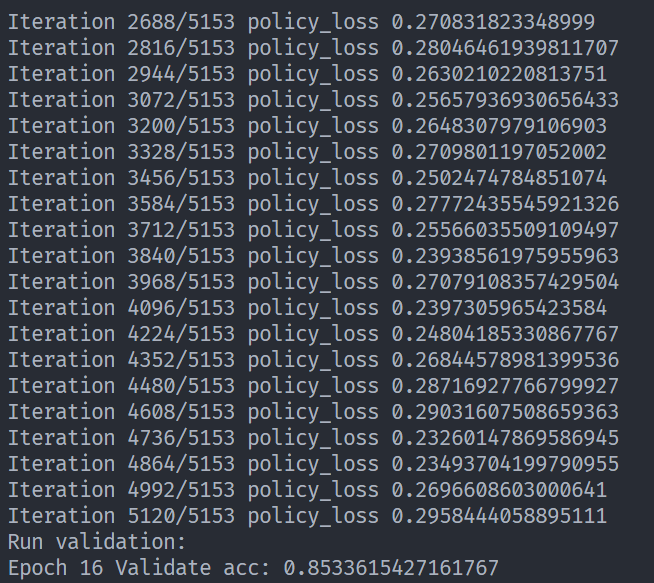
解决方案：使用残差块和多层MLP修改神经网络模型，增加模型拟合能力。

2. 模型缺乏吃碰杠等复杂行为。

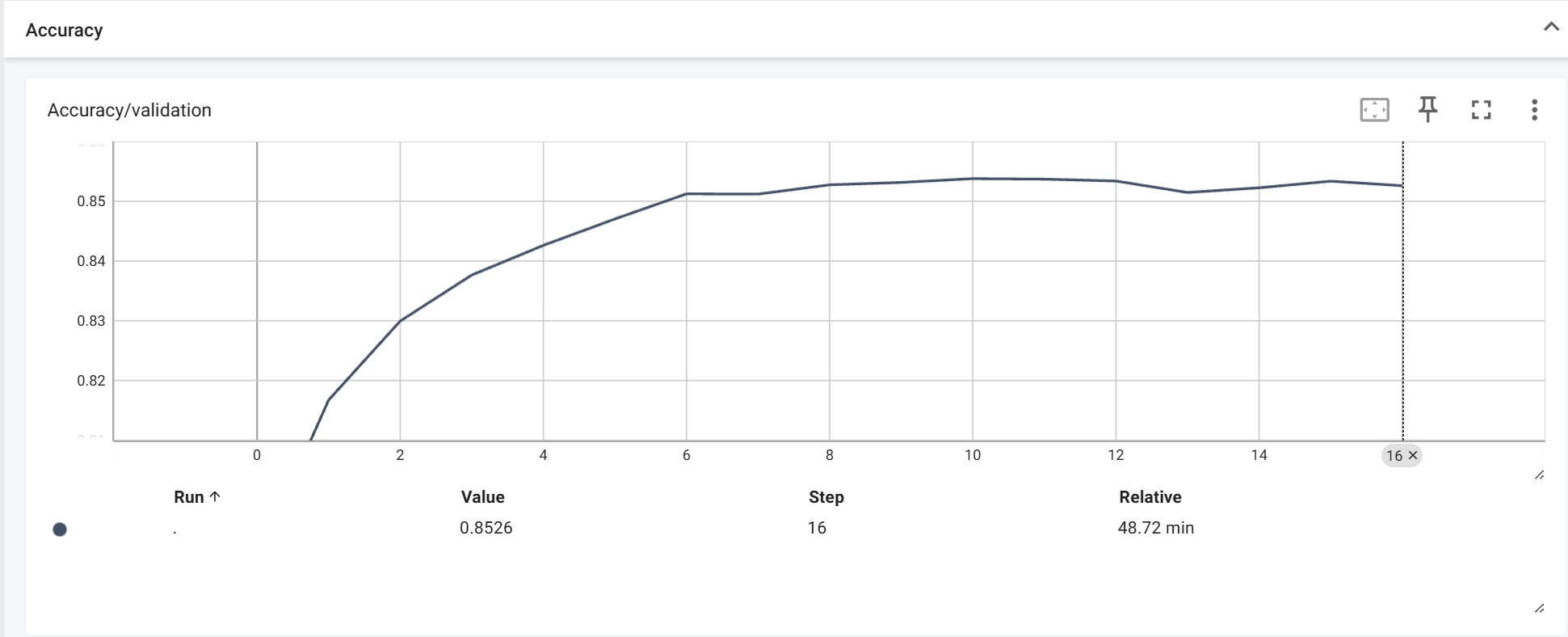
解决方案：修改特征提取，增加吃碰杠、弃牌历史和牌山等信息，供模型学习复杂攻防行为。

**十、训练效果展示（准确率等指标）：**

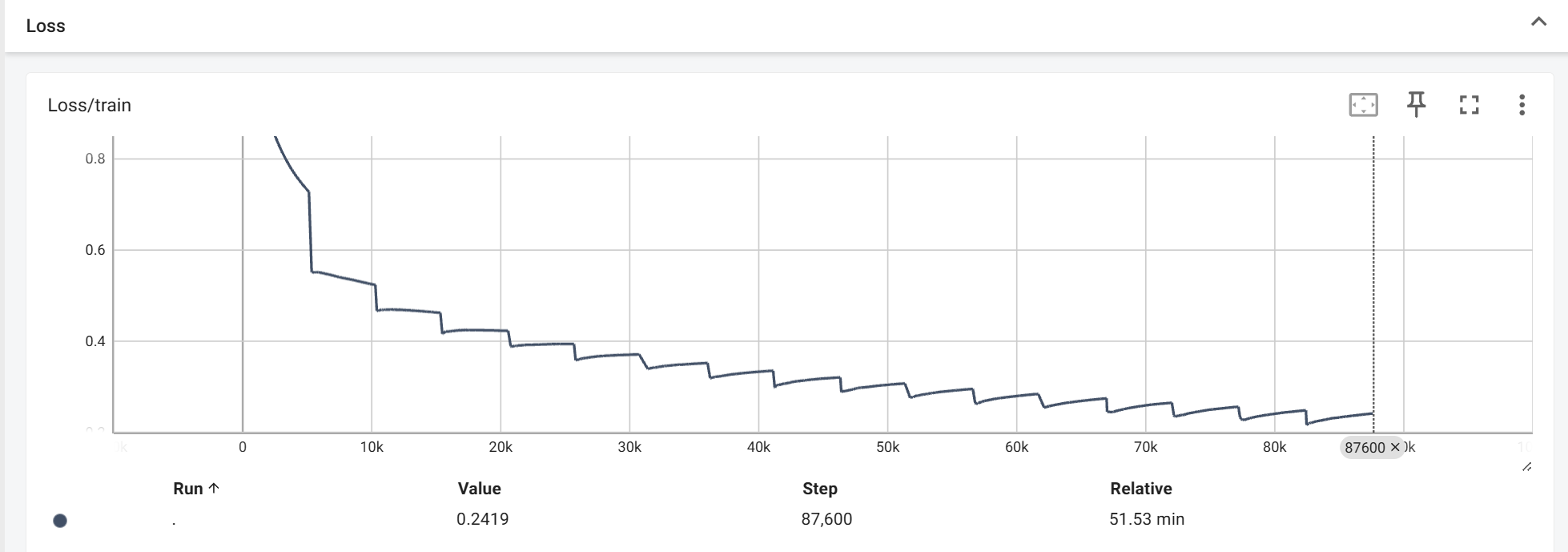
选取模型的训练效果：



训练测试集Accuracy变化曲线：



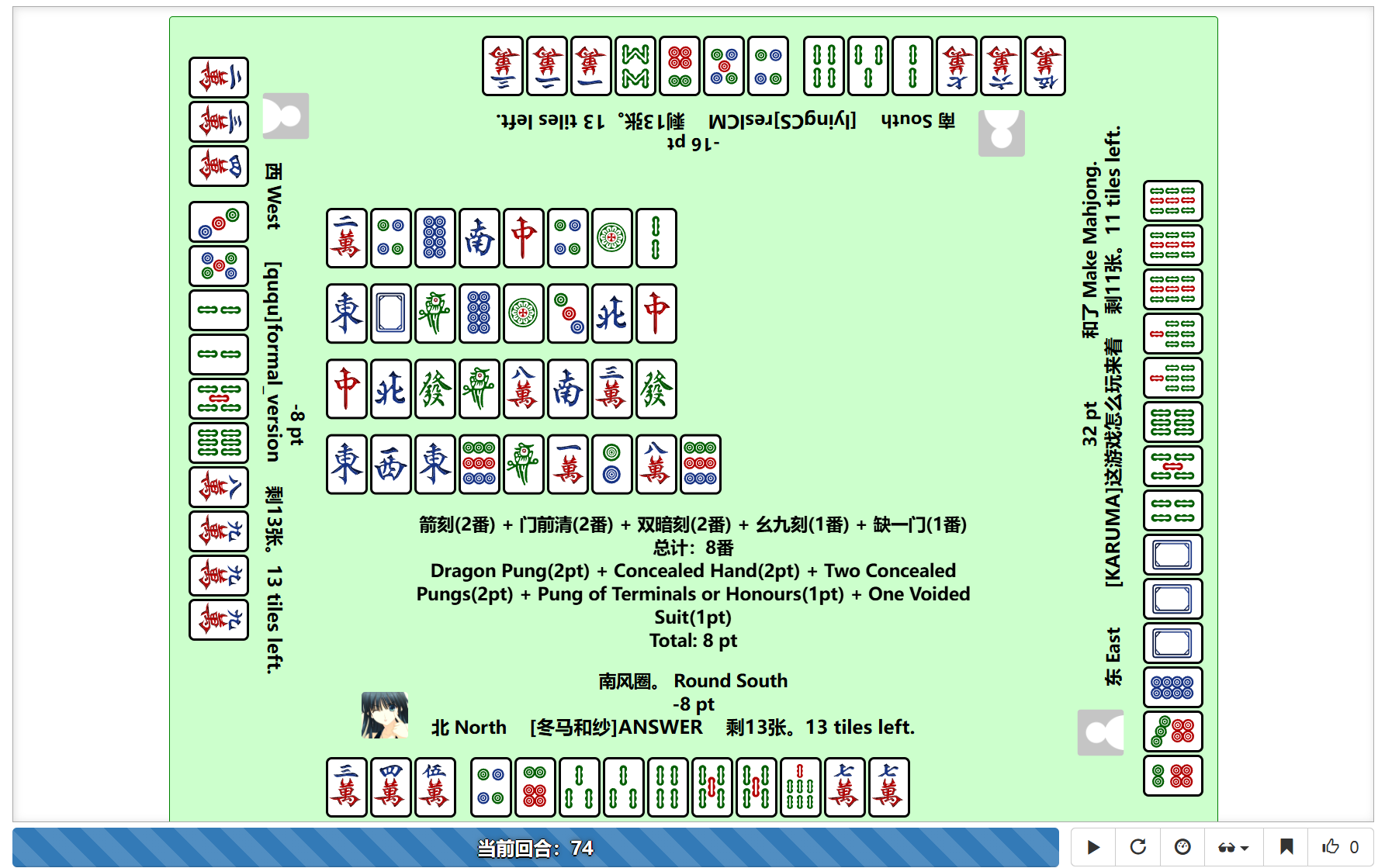
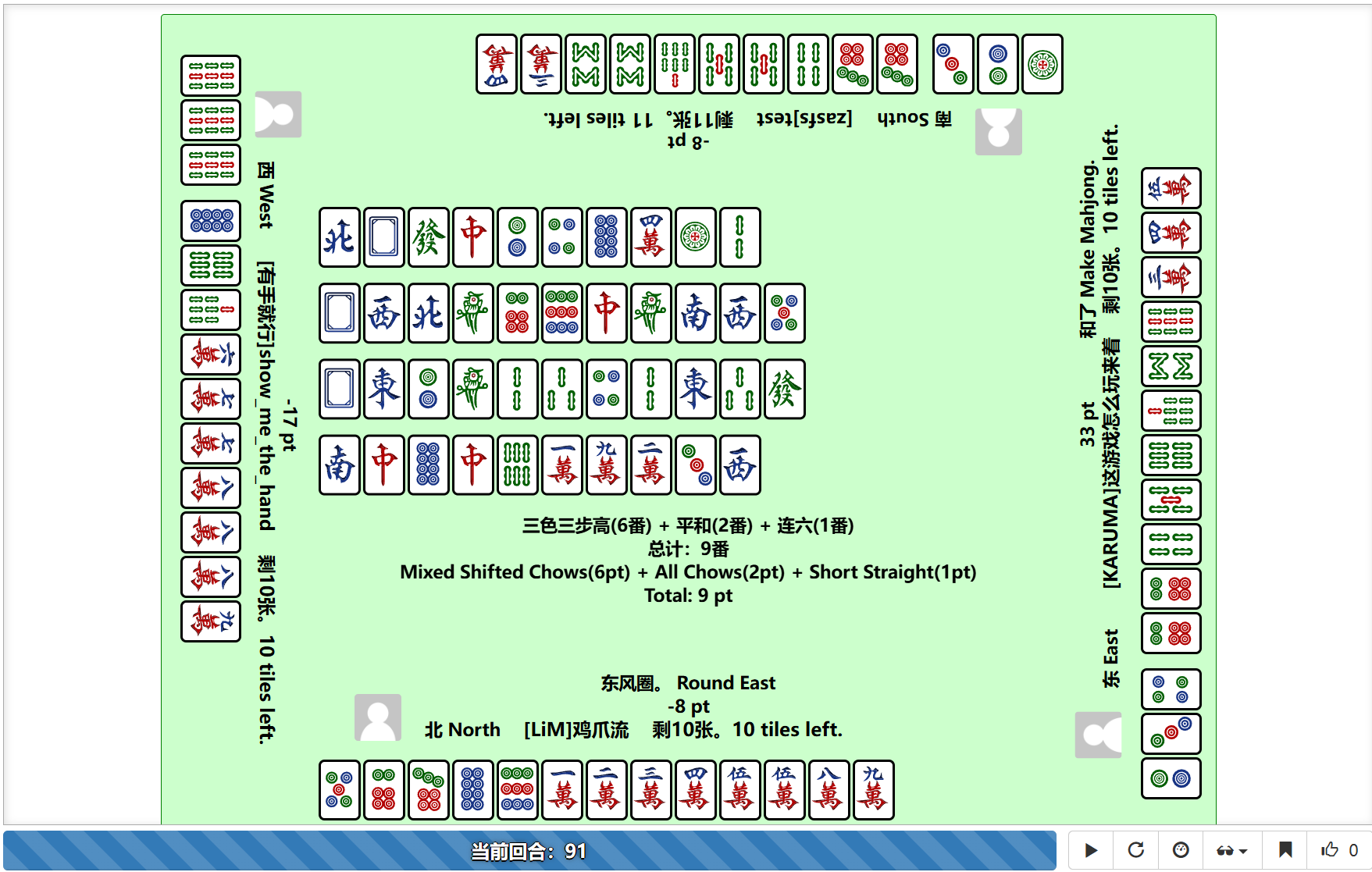
训练Loss变化曲线：

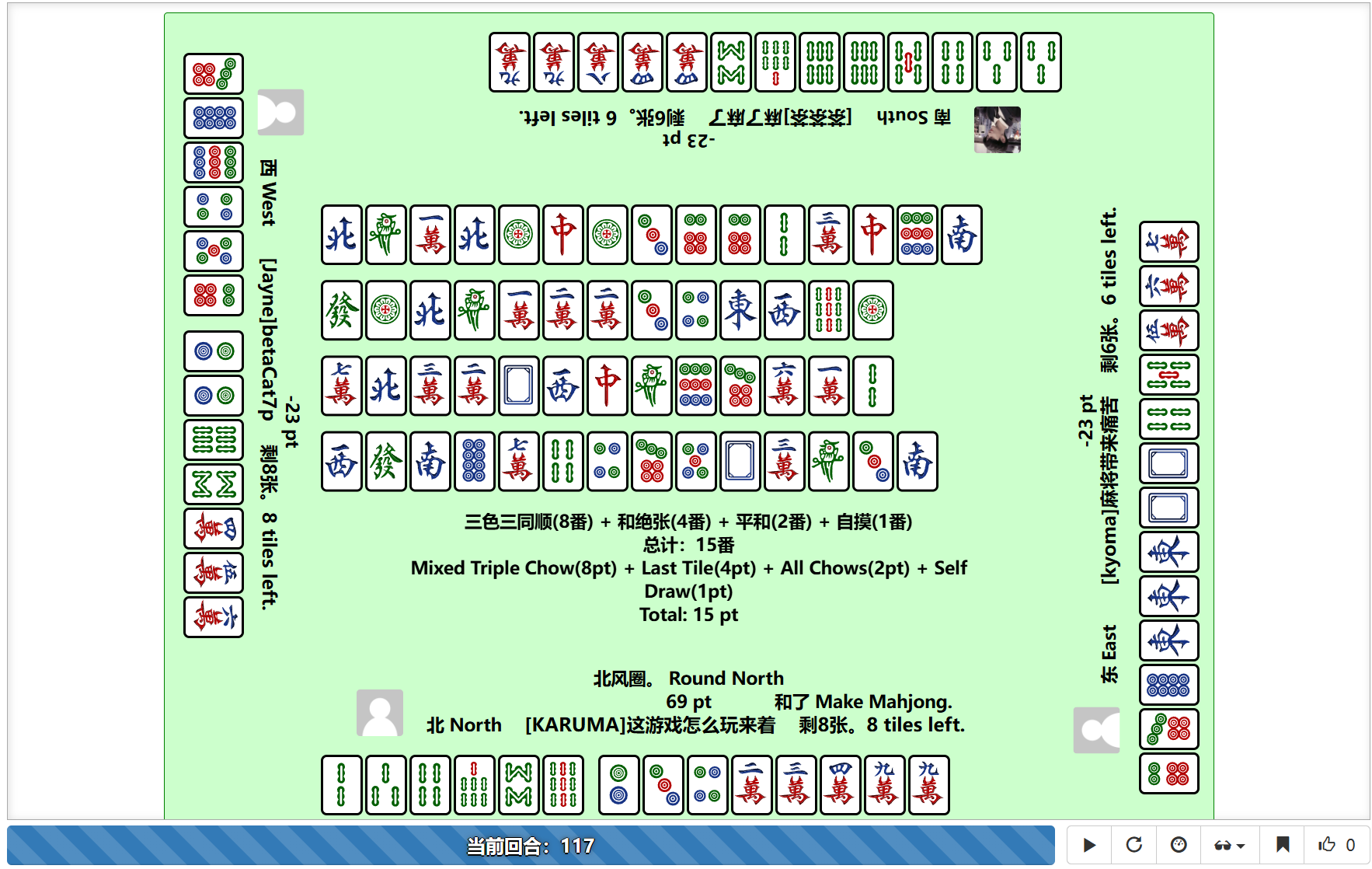


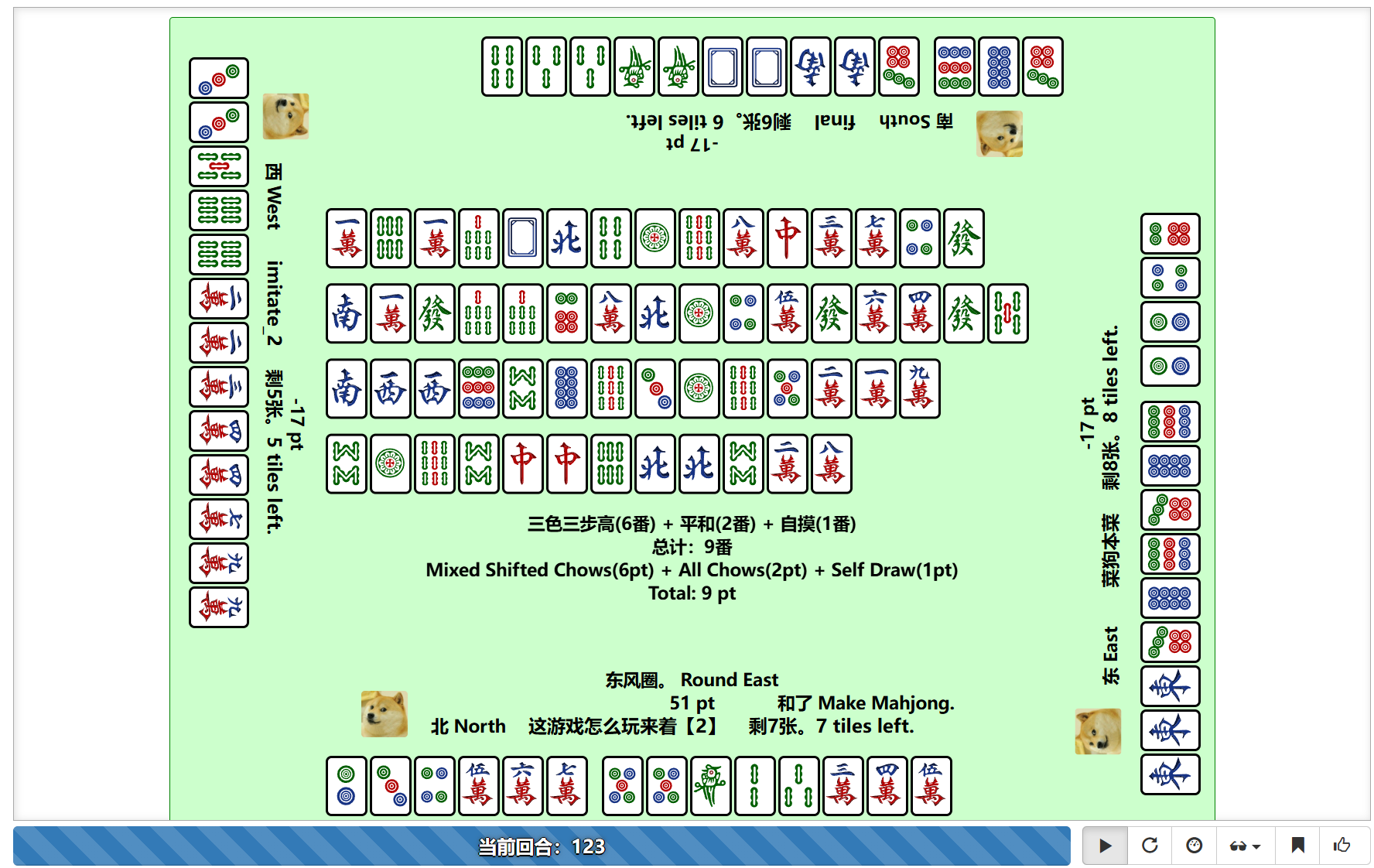
**十一、对战效果展示（上传Botzone对战效果）：**

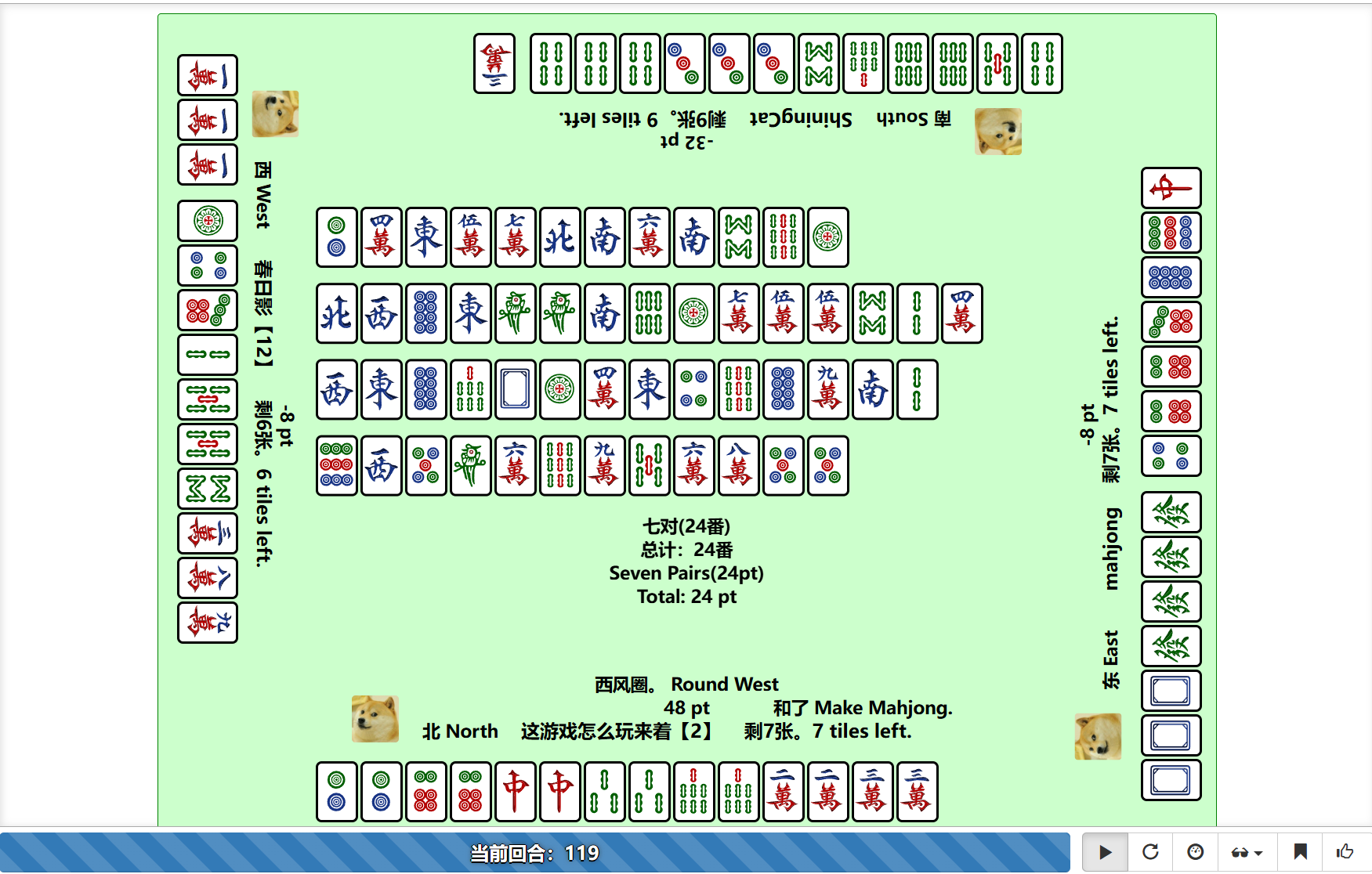
1. 对局效果展示（选取的均为天梯赛对局或排行榜中积分与本人相似的Bot对局）

（下列对局和积分赛排名的Bot中“[KARUMA]这游戏怎么玩来着”和“这游戏怎么玩来着”为本人Bot，存在两个版本的Bot）









2. 班内练习赛（第6次）排名：（本人为KARUMA）38/71



**十二、启示与收获：**

由于在本次作业前并未接触过任何形式的麻将游戏，本人因此在开始完成Bot代码时遇到了不小的阻力，包括几乎看不懂除了model.py和监督模型训练过程以外的所有代码，因此初期完善Bot性能时只能从修改神经网络模型入手，复原并改进样例模型。直到发现模型能力过于有限，才进一步学习麻将基础规则以修改特征提取更多全场信息。

另外，麻将是一种看似规则简单，实际游玩情况、策略都十分复杂的游戏，因此单纯的学习拟合已有对局数据实际上并不一定能找到最优解，这暗示了可能使用纯神经网络和监督学习方法不能获取AI行为最优解，需要其他搜索或对抗算法来获取更优行为。再者，麻将的玩家优劣势很难直接看出，必须要求玩家对麻将有着一定的了解才能明白某些情况下的某些行为才是更有优势的。除此之外，由于麻将是典型的不完全信息博弈游戏，且含有较大的随机性。在各种因素影响下，模型能力很难直接通过训练数据得到即时反馈，并且在缺乏麻将游戏的训练的情况下判断Bot的决策是否合理也相对困难。

这启示我在往后完成游戏类AI时，至少要对游戏规则有着充分的了解并对游戏策略有一定的理解，在这种情况下才能认识到Bot决策是否合理，并且根据已有的游戏理解进一步改进模型，从而获得更有效的AI。

*后记：本人Bot在班内赛中最终成绩为10/76。*