一、 引言

本作业设计并实现了一个基于深度强化学习的五子棋 AI，并对其性能进行了相应的测试。深度强化学习是深度学习与强化学习相结合的产物，本作业选用 DQN（Deep Q Network）作为学习算法，卷积神经网络作为神经网络模型，使用 PyTorch 进行实现，并利用 Tensorboard 对实验结果进行可视化。

五子棋博弈问题是一个人工智能经典问题，本作业使用深度强化学习解决该问题能够验证在复杂环境下深度强化学习的效果和算法的有效性。强化学习通过计算机在与环境交互的过程中通过不断尝试来学习并改进策略，而深度强化学习则利用深度学习方法进行值函数的近似。本作业的目的在于探寻深度强化学习在五子棋上的有效性和效果，并对其各项指标进行评测。

与传统的基于搜索的棋类游戏博弈 AI 相比，使用的深度强化学习的博弈 AI 可以通过不断的对局中进行自我更新与自我学习，而不会固定于算法给出的定式和限制。研究基于深度强化学习的五子棋博弈 AI 能够为之后更复杂、更高级的 AI 智能体的研究打下基础。

本作业实现了基于深度强化学习的五子棋人工智能体及用于强化学习的五子棋环境，并采用基于 Minimax 搜索算法的智能体来对网络进行训练。同时，本作业也实现了一个 GUI 人机交互界面，以及一系列评价测试程序。

二、设计思路

2.1 功能设计

为了实现基于深度强化学习的五子棋博弈 AI，本作业分为以下几部分功能模块：智能体、DQN 网络实现、五子棋环境、神经网络模型、GUI 图形界面、超参数配置、训练模块和一些辅助模块等。

2.1.1 智能体模块设计

智能体模块通过权衡当前棋局来采取最佳动作，并通过记忆和学习方法来进行自我学习。所有智能体都继承自抽象基类 BaseAgent 以实现接口的统一。本作业实现了基于 DQN 网络的深度强化学习智能体 DQNAgent 作为主要学习和评估对象、基于 MiniMax 搜索算法的传统智能体作为训练及评估的对手以及基于 GUI 棋盘的人机交互代理，以及一些基于贪婪和随机策略的智能体用以评估 DQN 智能体的评估。

对 DQN 智能体的训练及评估时使用的智能体并非本作业的重点在此不再赘述。

本作业的重点在于实现 DQN 智能体的训练及对落子价值的计算。该部分功能大部分位于 DQN 网络实现模块内，而 DQN 智能体模块仅为 DQN 网络实现的一个代理，向上提供一个智能体通用的接口。具体方法如下：

初始化方法：初始化方法接受一个 DQN 网络实例和可选的 TensorBoardX 日志记录器及网络模型保存文件夹，并将网络实例的超参数保存与日志记录中。

动作选择方法、记忆方法、学习方法、保存方法：为 DQN 网络动作方法的一个简单包装；学习方法内有记录损失函数的值于日志记录中的功能。

结束方法：在训练过程中于一盘棋局结束后被调用，用以动态更新探索率（采用指数衰减方法，公式为：$\epsilon = \epsilon_{\min} + (\epsilon_{\max} - \epsilon_{\min}) * \exp\left(-1 * \frac{\text{episode}}{\text{decay\_rate}}\right)$）。

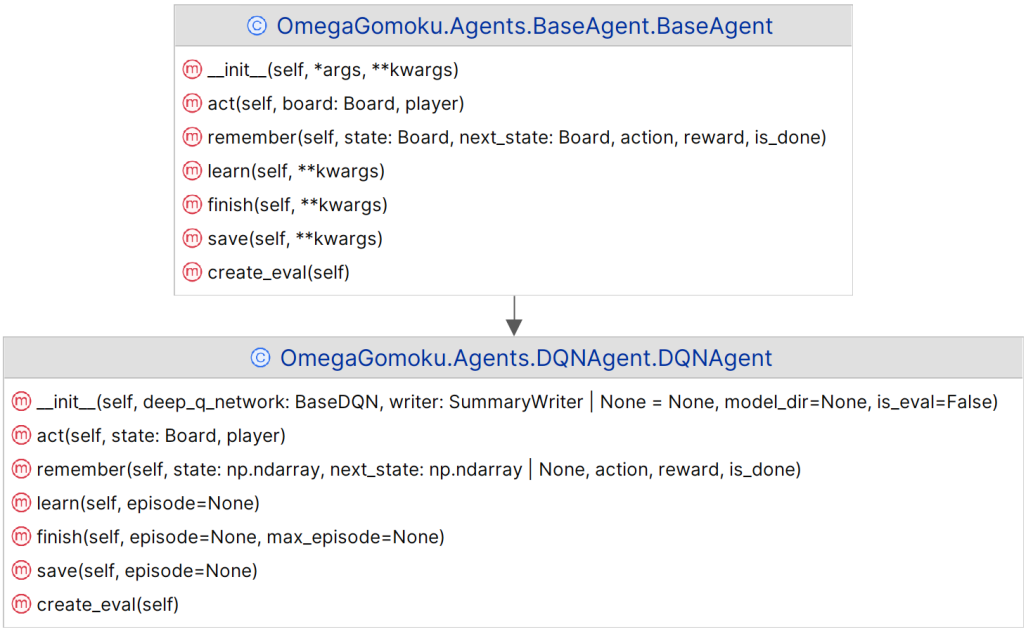

图 1 DQNAgent UML 类图

2.1.2 DQN 网络实现模块设计

DQN 网络实现是本作业最核心也是最重要的模块，本模块基于 PyTorch 框架实现了深度强化学习中的 DQN（Deep Q Network）算法，具有动作选择、记忆、学习、检查点的保存与加载等功能。算法的具体流程和详细步骤于 2.2 核心算法及详细步骤中详述，本小节仅介绍模块的逻辑实现。

本模块包含初始化方法、动作选择方法、学习方法、记忆方法、探索率衰减方法等，下面将对这些方法进行介绍：

初始化方法：该方法接收一系列参数，包括棋盘大小、训练超参数、是否使用 CUDA 加速及是否处于训练模式；根据接收到的参数，进行相应的初始化，并实例化损失函数和优化器。

动作选择方法：基于 ϵ-贪婪策略满足智能体在学习初期时的随机动作训练样本及学习后期中需要更进一步针对自己的网络计算结果进行训练这两种“探索（Exploration）”和“利用（Exploitation）”的平衡。本作业使用指数方法衰减探索率 ϵ，能够较好地平衡学习初期的“探索”和学习后期的“利用”。在智能体需要“利用”时，动作选择即为将棋盘状态输入神经网络，选择能获得最大 Q 值的落子，即在当前状态下采取收益期望最大的动作。而在智能体需要“探索”的时，为了解决五子棋盘中动作空间过大，纯随机落子的话产生的无意义落子过多而导致的神经网络收敛效果不佳、训练效率过低等问题，本作业引入了一种基于贪心策略的落子方法——即对棋盘进行攻防角度进行搜索产生具有优先级的动作空间，并限制智能体能够落子的区域来减少无意义的探索。具体算法请参见 2.2 核心算法及详细步骤。

学习方法：学习方法是整个网络模块的核心，简单来说就是在经验池中随机取样一批经验对 Q 值神经网络进行训练。并每隔一段时间更新目标网络的参数。具体算法请参见 2.2 核心算法及详细步骤。

记忆方法：记忆方法将当前状态、后续状态、动作、奖励、是否结束这个五元组存入经验池中。

探索率衰减方法：探索率衰减方法负责接受线性衰减率或衰减函数对探索率 ϵ 进行衰减处理，以提高智能体后期对“利用”方面的“偏爱”，调整探索率衰减率是平衡智能体训练时对“探索”和“利用”权衡的关键，很大程度上影响模型的训练效果。

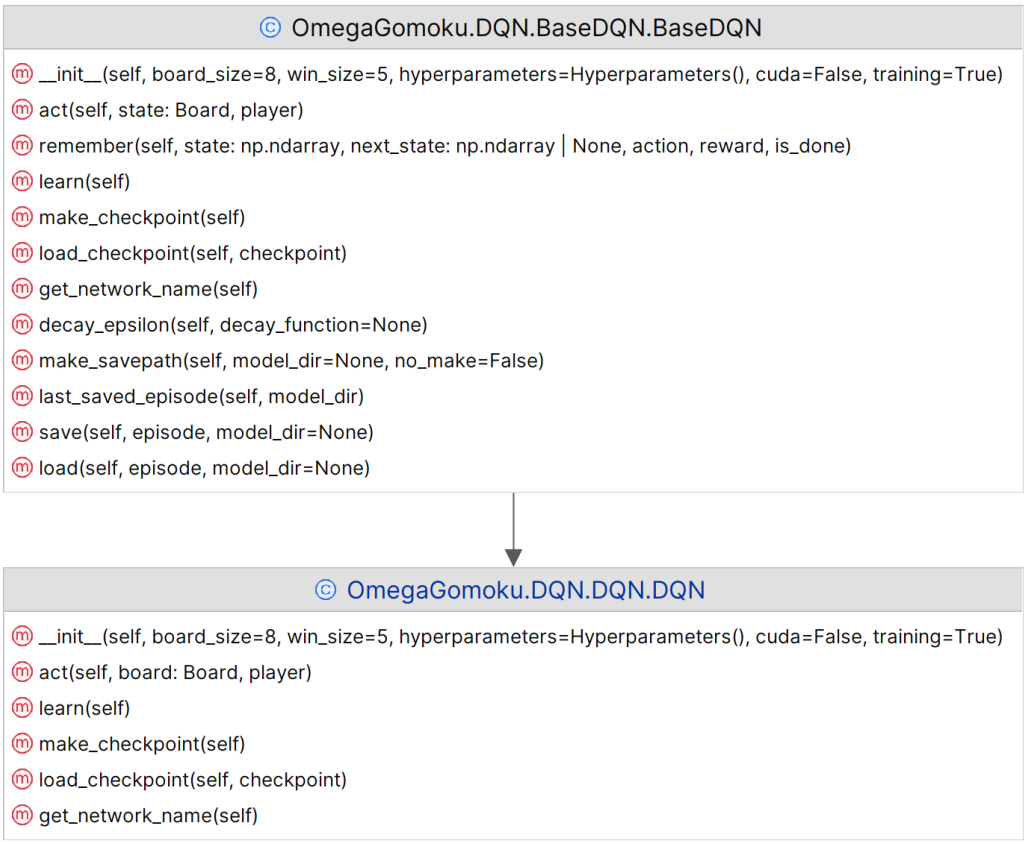

图 2 DQN UML 类图

2.1.3 五子棋环境模块设计

在强化学习中，环境作为智能体与之互动并取得反馈的唯一途径，是强化学习的基础。而深度强化学习也是通过与环境的不断交互来进行信息的获取，从而学习到最优的决策，所以一个良好的环境设计对智能体训练的效果起到了至关重要的作用。

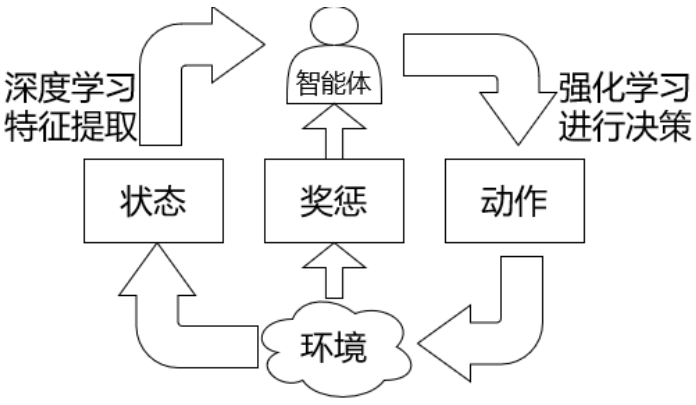

图 3 深度强化学习框架

针对本作业的五子棋博弈 AI，需要针对五子棋的不同棋形和落子产生的胜负情况等各种因素进行综合考虑计算过程性奖励，以对智能体实时反馈，引导智能体学习到能让落子产生最大价值的最优策略。本作业起初力图避免于环境中引入先验知识进行训练，但效果极差，作为状态空间极大的对弈类游戏的五子棋一旦不引入先验知识对

棋局进行过程性判定，则会导致奖励过于稀疏，神经网络无法学习到有效策略。

本模块包含初始化方法、重置方法和着子方法等交互方法，内部包含棋盘定义、奖励计算等其他子模块。下面针对这些方法和子模块进行介绍：

初始化方法：接收对手智能体、棋盘大小、GUI 棋盘等参数，并将其保存于模块实例内。对手智能体是进行深度强化学习训练的基准，本作业尝试过智能体自我对弈，但效果不佳，故使用改编后的 MiniMax 智能体进行训练。同时，由于五子棋的二人博弈特性，故对手的落子也属于环境的一部分。（本作业初期将对手的着子过程于训练模块中进行，但容易产生误解。）

重置方法：重置环境状态。

着子方法：接收着子位置，并给出此步着子后的棋盘、奖励分数和棋局结束状态。

奖励计算方法于 2.2 核心算法及详细步骤中详述。

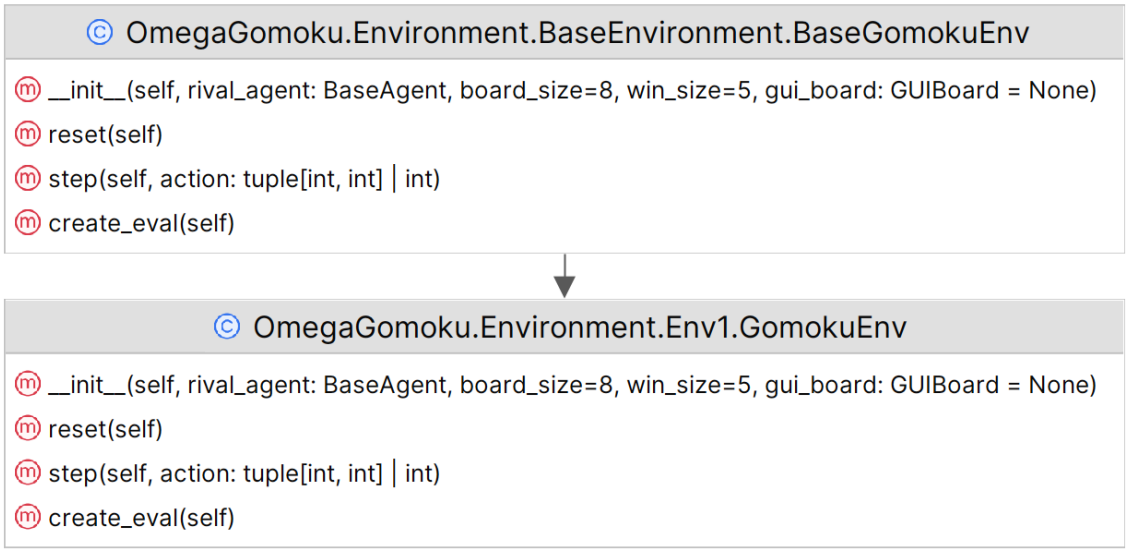

图 4 GomokuEnv UML 类图

2.1.4 神经网络模型模块设计

神经网络模型模块是在本作业中实现起来最简单但是却起到关键作用的模块，利用 PyTorch 提供的各类工具，可以很方便的构建一个神经网络，并能够通过手动编写的前向传播函数自动生成反向传播函数。

本作业尝试了各种神经网络模型，包括全连接模型和卷积神经网络，以及对卷积神经网络的池化层的使用效果进行了评估。最后选用了使用三层卷积核分别为 64、128、128 的不使用池化层的卷积神经网络及一层作为输出用的全连接层的模型。

神经网络的输入为两通道的棋盘状态，即白子棋盘状态（1 代表着子，0 代表空位）和黑子棋盘状态（1 代表着子，0 代表空位），本作业尝试使用单通道棋盘棋盘状态（即 1 代表白子、2 代表黑子、0 代表空位）进行实验，但无法正常收敛。

神经网络的输出为棋盘大小的 Q 值张量，代表动作选择到棋盘上的对应位置会取得的预期收益，通常取预期收益最大值所在的位置为动作选择结果。

2.1.5 GUI 图形界面模块设计

GUI 图形界面使用 Tkinter 开发，运行于子线程中。本模块对外提供绘制棋盘、等待玩家输入、显示信息等功能。能够提高人机交互的使用体验，并可更方便地评估训练好的模型。

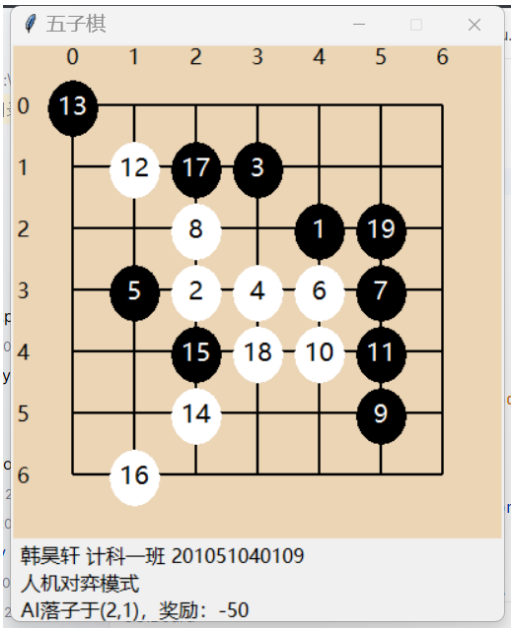

图 5 GUI 图形界面实例

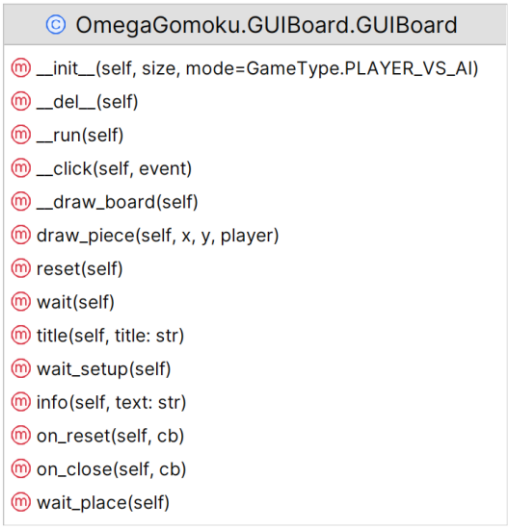

图 6 GUIBoard UML 类图

2.1.6 其他辅助模块设计

本作业还设计实现了训练模块、超参数模块和枚举常量模块，用以辅助智能体的训练。其中超参数模块用以快速调整神经网络训练参数，对于本作业具体选取的超参数请参阅 2.3 神经网络训练参数。

2.2 核心算法及详细步骤

本节选择了本作业中重要的算法及过程，并针对其进行详细说明。

2.2.1 DQN 算法及详细步骤

DQN 算法是 Q-Learning 算法与深度学习的结合。对于规模过大的问题，使用传统的 Q-Learning 中的 Q 表存储状态不切实际，以五子棋盘为例，一个 10*10 大小的棋盘，每个格子有 3 各种状态，那么一共会存在3^{100}个 Q 表项，这显然是不可能做到的。而 DQN 算法利用深度学习拟合 Q 值函数，能够高效地进行智能体的训练和估值。

DQN 算法要求智能体从环境中记忆状态，并通过自身的记忆不断调整网络，从而达到逼近真实的 Q 值函数的目的。

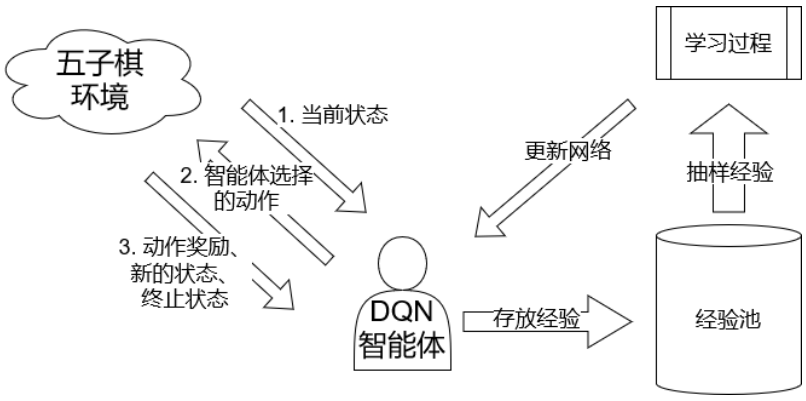

图 7 DQN 算法框架

DQN 算法要求智能体在与环境的交互中记忆下每一步产生的新状态和该布的奖励值等数据，用以在算法内抽样并进行学习。具体的算法详细步骤如下：

Algorithm 1 DQN 算法

输入: 五子棋环境 E

初始化经验池 D，动作网络 Q，目标网络 $\hat{Q}$
repeat 训练未结束
    重置五子棋环境 E
    repeat 棋局未结束
        观察环境，获得当前状态 S
        使用 ϵ-**贪婪**策略选择动作 $a = \begin{cases} \textbf{贪心落子算法}, & \text{若 } \epsilon\text{-贪婪策略选择探索} \\ \text{argmax}_a Q(S,a), & \text{若 } \epsilon\text{-贪婪策略选择利用} \end{cases}$
        执行动作 a，观察环境，获得奖励 r、新的状态 S'
        记忆经验：$D \Leftarrow (S, S', a, r)$
        if 经验池 D 容量过小 **then**
            跳过训练，继续进行棋局
        end if
        于经验池 D 中抽取一批经验 (SS, SS', aa, rr)
        计算真实 Q 值 $y = \begin{cases} rr, & \text{若 SS' 为终止状态} \\ rr + \gamma \max_{a'} \hat{Q}(SS', a'), & \text{否则} \end{cases}$
        根据设定的损失函数计算真实 Q 值 y 和计算 Q 值 $Q(SS, aa)$
        通过反向传播进行梯度下降
        每间隔 C 步更新 $\hat{Q} \leftarrow Q$

2.2.2 贪心落子算法及详细步骤

作为没有初始训练样本的深度强化学习算法，智能体学习初期所使用的训练样本都是自己根据ϵ-贪婪策略生成的。而对于五子棋这种动作空间过大，采用完全随机的探索策略产生的训练样本通常是没有意义的，智能体训练的效率也会大打折扣。为了提高智能体训练的效率和效果，本作业采用了一种基于贪心策略的落子算法以供智能体执行探索策略时使用。

贪心落子算法基于传统搜索算法，通过遍历棋盘上所有己方和对方的连线，根据其能够转移到的进攻和防守价值的点位对进攻和防守操作添加相应的动作，并按照防守优先、进攻其次（存在可能连五的空位时进攻优先）的优先级选择接下来落子。

贪心落子算法还可以用于对智能体训练结果的评估。具体的算法详细步骤如下：

Algorithm 2 贪婪落子算法

输入: 五子棋棋盘 **Board**

输出: 贪婪落子算法采取的动作

初始化进攻价值为 $5\dots2$ 的层 $\textbf{Attack}_{5\dots2} \leftarrow \{\}$
for **Line** $\in$ 棋盘空间 $\textbf{Board}_{\text{player}}$ **do**
    if **Line** 下一步着子选择 **Actions** 可转移至具备进攻价值 N 的状态 **then**
        $\textbf{Attack}_N \Leftarrow$ **Actions**
    end if
end for
初始化防守为 $4\dots3$ 的层 $\textbf{Defense}_{4\dots3} \leftarrow \{\}$
for **Line** $\in$ 棋盘空间 $\textbf{Board}_{\text{rival}}$ **do**
    if **Line** 下一步存在 **Actions** 可转移当前棋形的进攻价值 N 到 $N+1$ 的状态 **then**
        $\textbf{Defense}_N \Leftarrow$ **Actions**
    end if
end for
动作空间 **Actions** $\leftarrow \{\}$
if $\textbf{Attack}_5 \neq \{\}$ **then** **Actions** $\leftarrow \textbf{Attack}_5$
else if $\textbf{Defense}_4 \neq \{\}$ **then** **Actions** $\leftarrow \textbf{Defense}_4$
else if $\textbf{Attack}_4 \neq \{\}$ **then** **Actions** $\leftarrow \textbf{Attack}_4$
else if $\textbf{Defense}_3 \neq \{\}$ **then** **Actions** $\leftarrow \textbf{Defense}_3$
else if $\textbf{Attack}_3 \neq \{\}$ **then** **Actions** $\leftarrow \textbf{Attack}_3$
else if $\textbf{Attack}_2 \neq \{\}$ **then** **Actions** $\leftarrow \textbf{Attack}_2$
else **Actions** $\leftarrow$ 所有有效的动作
end if
return **Actions** 中随机选取的一个动作

2.2.3 奖励模型与奖励计算详细步骤

在本作业中，有以下几种主要五子棋的棋形参与了奖励的评估（白点代表可用的动作空间，不包括互相对称的棋形）：

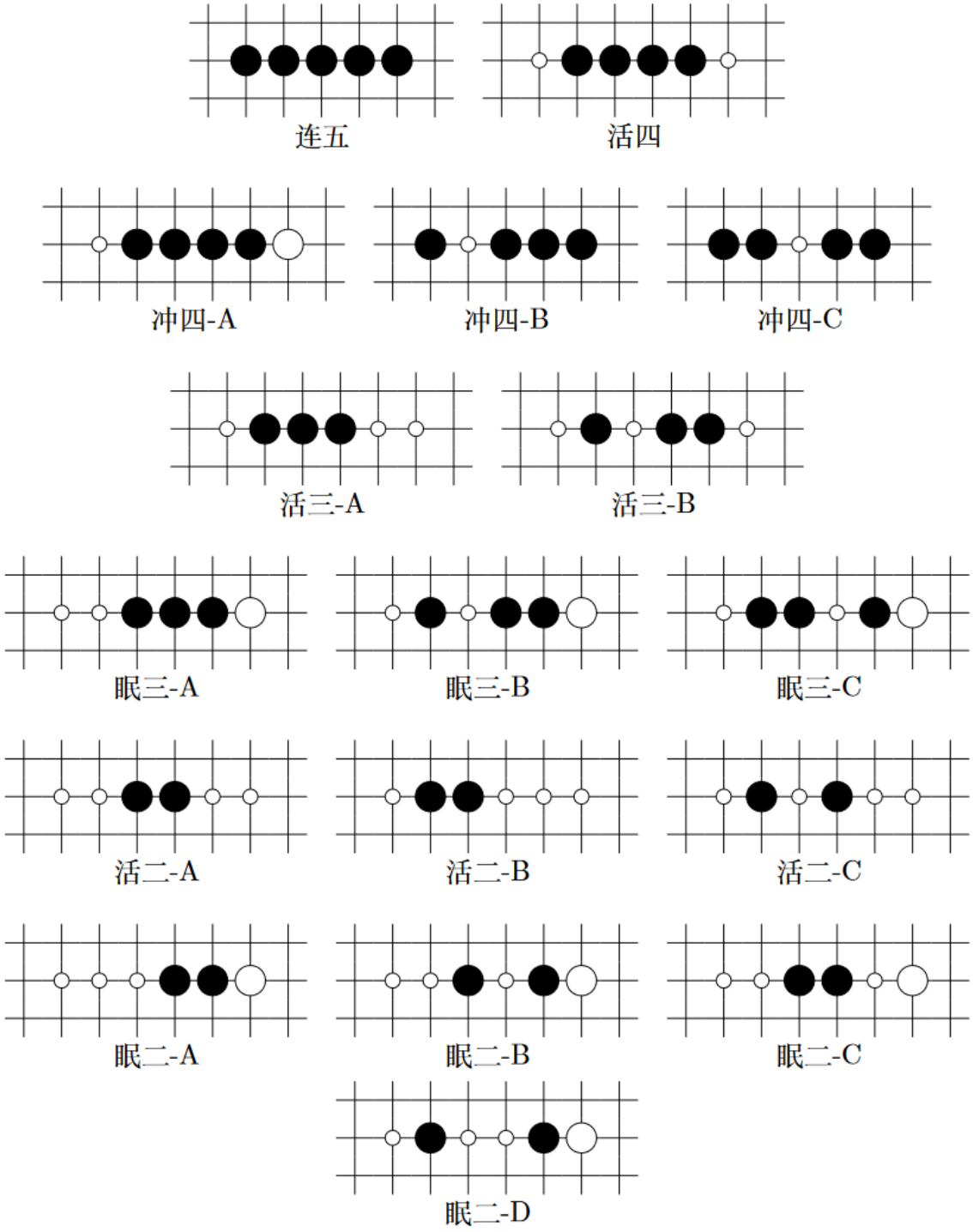

图 8 本作业中使用的五子棋棋形

针对上图中的每个棋形，本作业设计了相对应的固定奖励表，用以量化各个棋形的价值以及参与后续的奖励计算步骤。固定奖励表如下：

棋形	奖励值（价值）
连五	200
活四	100
冲四	60
活三	30
眠三	20
活二	10
眠二	5

表 1 棋形奖励表

本作业为了更好的引导智能体的训练，采用动态计算奖励值的方法。通过将智能体落子产生的棋形、落子打破对手的棋形、对手落子产生的棋形和终止状态综合考虑，计算出一个动态的奖励值，具体算法如下（奖励系数表、终局奖励表于下节中给出）：

Algorithm 2 奖励计算算法

输入: 智能体落子形成的最佳棋形 $\mathbf{Pattern}_{\text{self}}$，智能体落子打破对手可能形成的最佳棋形 $\mathbf{Pattern}_{\text{break}}$，对手落子形成的最佳棋形 $\mathbf{Pattern}_{\text{rival}}$，终止状态，棋形奖励表 R，奖励系数表 Γ，终局奖励表 T

输出: 奖励数值

if $\mathbf{Pattern}_{\text{self}}, \mathbf{Pattern}_{\text{break}}, \mathbf{Pattern}_{\text{rival}}$均为空 **then return** 0
end if
if 终止状态为平局 **then return** T_{draw}
else if 终止状态为胜局 **then return** T_{win}
else if 终止状态为负局 **then return** T_{loss}
end if
初始化奖励数值 $r = 0$
if $\mathbf{Pattern}_{\text{self}}$不为空 **then** $r \leftarrow r + R[\mathbf{Pattern}_{\text{self}}]$
end if
if $\mathbf{Pattern}_{\text{rival}}$不为空 **then**
    初始化 $r_{\text{rival}} \leftarrow R[\mathbf{Pattern}_{\text{rival}}]$，对手奖励折扣系数 $\gamma = \Gamma_{\text{rival}}[3]$
    if $r_{\text{rival}} \leq r$ **then** $\gamma \leftarrow \Gamma_{\text{rival}}[1]$
    else if $r_{\text{rival}} \leq 60$ **then** $\gamma \leftarrow \Gamma_{\text{rival}}[2]$
    end if
    $r \leftarrow r - \gamma * r_{\text{rival}}$
end if
if $\mathbf{Pattern}_{\text{break}}$不为空 **then**
    初始化 $r_{\text{break}} \leftarrow R[\mathbf{Pattern}_{\text{break}}]$，破坏棋形奖励折扣系数 $\gamma = \Gamma_{\text{break}}[3]$
    if $r_{\text{break}} \leq 20$ **then** $\gamma \leftarrow \Gamma_{\text{break}}[2]$
    else if $r_{\text{break}} \leq 60$ **then** $\gamma \leftarrow \Gamma_{\text{break}}[1]$
    end if
    $r \leftarrow r + \gamma * r_{\text{break}}$
end if
return r

2.3 神经网络训练参数

本作业尝试了多种训练参数、网络模型、损失函数和优化器的组合，并得出了一个学习效果最佳、性能最好的组合。具体参数等设置及解释如下：

参数名	取值	解释
batch_size	256	一次训练中抽样经验的数量。
memory_size	100000	记忆空间大小。
learning_rate	1e-5	学习率。
gamma	0.70	奖励折扣因子，越高代表智能体倾向长期价值。
epsilon_min	0.10	最小探索率。探索率越高智能倾向选择“探索”。
epsilon_max	1.00	最大探索率。
epsilon_decay_rate_exp	1500	探索率指数衰减参数，参见 2.1.1。
update_target_model_each_iter	200	每间隔几次学习更新目标网络，参见 2.2.1。
train_epochs	2	每次训练时进行几次梯度下降。
loss	SmoothL1Loss	使用何种损失函数。
optimizer	AdamW	使用何种优化器。

表 2 超参数表

对于计算动态奖励值时使用的奖励系数表 Γ，同样属于超参数的一部分，影响智能体的训练决策。目前选择的奖励系数表如下表所示：

Γ_{rival}	**索引**	**1**	**2**	**3**
	取值	0.3	0.5	0.8
Γ_{break}	**索引**	**1**	**2**	**3**
	取值	0.3	0.5	1

表 3 奖励系数表

对于达到终局的奖励值计算，我们只取固定的值以稳定终局状态的价值。

胜局	败局	平局
800	-200	-20

表 4 终局奖励表

三、详细设计及结果分析

3.1 代码实现

本节节选了本作业代码中的核心部分，完整代码请参阅项目文件。

3.1.1 DQN 算法核心代码实现

```
from . import BaseDQN, Memory, MemoryEntry
from ..Hyperparameters import Hyperparameters
from ..Models import GomokuAI
from ..Environment import Board, Utils

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np


class DQN(BaseDQN):
    def __init__(self, board_size=8, win_size=5,
hyperparameters=Hyperparameters(), cuda=False, training=True):
        super().__init__(board_size, win_size, hyperparameters, cuda)
        self.memory: Memory = self.memory  # For auto-complete
        self.action_size = board_size ** 2
        self.eval_model =
GomokuAI(board_size).train(training).to(self.device)
        self.target_model = GomokuAI(board_size).eval().to(self.device)

        self.learn_count = 0
        self.training = training
        loss_class = getattr(nn, hyperparameters.loss)
        optimizer_class = getattr(optim, hyperparameters.optimizer)
        self.loss = loss_class()
        self.optimizer = optimizer_class(self.eval_model.parameters(),
lr=hyperparameters.learning_rate)

    def act(self, board: Board, player):
        valid_moves = board.get_valid_moves().reshape(self.action_size)
        suggested_moves = board.get_suggested_moves(player,
extend=3).reshape(self.action_size)
        if self.training:
            valid_moves *= suggested_moves  # Let valid moves only in
suggested moves
        if board.is_empty():
            # 如果棋盘为空，在棋盘中心的 5x5 区域随机选取一个点位下棋
            round_size = 5
            board_size = board.board_size
            round_arr = np.full((board_size, board_size), -np.inf)
            s = (board_size - round_size) // 2
            round_arr[s:s + round_size, s:s + round_size] = 1
            round_arr = round_arr.reshape(self.action_size)
            valid_moves *= round_arr
        state = board.get_state()
        if board.is_empty() or self.training and np.random.rand() <=
self.hyperparameters.epsilon:
            # Do epsilon choose
            atk2, atk3, atk4, atk5 = Utils.gen_dfs_atk_moves(state, True)
            def3, def4 = Utils.gen_dfs_atk_moves(state, False)
            pool = None
```

```
            if atk5.any():
                pool = atk5
            elif def4.any():
                pool = def4
            elif atk4.any():
                pool = atk4
            elif def3.any():
                pool = def3
            elif atk3.any():
                pool = atk3
            elif atk2.any():
                pool = atk2
            else:
                # pool = valid_moves * suggested_moves
                pool = valid_moves
            pool = pool.reshape(self.action_size)
            action_values = np.random.uniform(-1, 1, size=self.action_size)
            valid_values = np.where(pool == 1, action_values, -np.inf)
            return np.argmax(valid_values)

        # Do Think
        state = state.reshape([2, self.board_size, self.board_size])
        state = state[np.newaxis, :, :]
        state = torch.from_numpy(state).float().to(self.device)
        # print(state, state.size())
        with torch.no_grad():
            # 不需要梯度
            action_values = self.eval_model(state)
        valid_values = np.where(valid_moves == 1,
action_values.cpu().detach().numpy()[0], -np.inf)
        # return np.argmax(action_values.cpu().numpy()[0] * valid_moves)
        return np.argmax(valid_values)

    def learn(self):
        # if self.memory_counter < self.hyperparameters.batch_size:
        if len(self.memory) < 16:
            return None
        if self.learn_count %
self.hyperparameters.update_target_model_each_iter == 0:
            self.target_model.load_state_dict(self.eval_model.state_dict())
        batch_size = min(self.hyperparameters.batch_size, len(self.memory))
        batch = self.memory.sample(batch_size)
        # print(f"Sample: {batch[0]}")
        batch = MemoryEntry(*zip(*batch))
        states = torch.from_numpy(np.array(batch.state,
dtype=np.float32)).to(self.device)
        actions = torch.from_numpy(np.array(batch.action,
dtype=np.int64)).to(self.device)
        rewards = torch.from_numpy(np.array(batch.reward,
dtype=np.float32)).to(self.device)

        non_final_mask = torch.tensor(tuple(1 if not i else 0 for i in
batch.is_done)).to(self.device)
        non_final_next_states = torch.from_numpy(
            np.array(tuple(i for i in batch.next_state if i is not None),
dtype=np.float32)
        ).to(self.device)

        q_eval = self.eval_model(states)
        q_target = q_eval.clone()
```

```
        batch_index = np.arange(batch_size, dtype=np.int32)
        next_state_values = torch.zeros(batch_size, device=self.device)
        with torch.no_grad():
            next_state_values[non_final_mask == 1] =
self.target_model(non_final_next_states).max(1)[0]
        q_target[batch_index, actions] = rewards +
self.hyperparameters.gamma * next_state_values * non_final_mask

        # fit
        q_target = q_target.detach()
        loss = 0
        for _ in range(self.hyperparameters.train_epochs):
            pred = self.eval_model(states)
            pred_loss = self.loss(pred, q_target)
            self.optimizer.zero_grad()
            pred_loss.backward()
            self.optimizer.step()
            loss += pred_loss
        self.learn_count += 1
        avg_loss = loss / self.hyperparameters.train_epochs
        return avg_loss

    def make_checkpoint(self):
        return {
            'model': self.eval_model.state_dict(),
            'memory': self.memory,
            'hyperparameters': self.hyperparameters,
            'optimizer': self.optimizer.state_dict()
        }

    def load_checkpoint(self, checkpoint):
        self.eval_model.load_state_dict(checkpoint['model'])
        self.target_model.load_state_dict(checkpoint['model'])
        self.optimizer.load_state_dict(checkpoint['optimizer'])
        self.memory = checkpoint['memory']
        self.hyperparameters = checkpoint['hyperparameters']

    def get_network_name(self):
        return "DQN2-" + self.eval_model.model_name
```

3.1.2 卷积神经网络模型代码实现

```
class GomokuAI3(nn.Module):
    def __init__(self, board_size=8):
        super(GomokuAI3, self).__init__()
        conv_setup = [(2, 0, 0, 0), (64, 5, 1, 2), (128, 3, 1, 1), (128, 3,
1, 1)]
        for i in range(len(conv_setup) - 1):
            in_channel = conv_setup[i][0]
            out_channel, kernel_size, stride, padding = conv_setup[i + 1]
            self.__setattr__(f"conv{i + 1}", nn.Conv2d(in_channel,
out_channel, kernel_size, stride, padding))
            self.__setattr__(f"bn{i + 1}", nn.BatchNorm2d(out_channel))
        self.conv_size = conv_setup

        def conv2dSizeOut(size, layer=1):
            size = size if layer == len(conv_setup) - 1 else
conv2dSizeOut(size, layer + 1)
            _, kernel_size, stride, padding = conv_setup[layer]
```

```
            return (size - kernel_size + 2 * padding) // stride + 1

        out_ = conv2dSizeOut(board_size)

        self.fc1 = nn.Linear(out_ ** 2 * conv_setup[-1][0], board_size **
2)
        self.model_name = "GomokuAI-3"

    def forward(self, input):
        for i in range(len(self.conv_size) - 1):
            input = nn.functional.relu(
                self.__getattr__(f"bn{i + 1}")(
                    self.__getattr__(f"conv{i + 1}")(input)))
        return self.fc1(input.view(input.size(0), -1))
```

3.1.3 训练过程代码实现

```
class Trainer:
    def __init__(self,
                 env: GomokuEnv,
                 agent: DQNAgent,
                 writer: SummaryWriter | None = None,
                 max_save_checkpoints=5):
        if not isinstance(agent, DQNAgent):
            raise Exception("Only DQNAgent Could be Trained")
        self.env = env
        self.agent = agent
        self.checkpoint_savepaths = deque()
        self.max_save_checkpoints = max_save_checkpoints
        self.writer = writer
        if self.writer is None:
            w = self.agent.__getattribute__('writer')
            if w is not None:
                self.writer: SummaryWriter = w

    def train(self, episodes, start_episode=0):
        wins = 0
        round_ = 0
        for e in tqdm(range(start_episode, episodes + 1)):
            round_ += 1
            try:
                board = self.env.reset()
                done = False
                steps = 0
                total_reward = 0
                while not done:
                    steps += 1
                    # 先手，待训练智能体
                    state = board.get_state().copy()
                    player = self.env.current_player
                    action = self.agent.act(board, player)
                    board, reward, terminal_status = self.env.step(action)
                    done = terminal_status is not None
                    next_state = None if done else board.get_state().copy()

                    total_reward += reward
                    memory = MemoryEntry(state, next_state, action, reward,
done)
                    if False and done and terminal_status != 0:
```

```
                    tqdm.write(
                        f"Final states: {memory}\nTerminal Status:
{['draw', 'win', 'loss'][terminal_status]}"
                    )
                # noinspection all
                self.agent.remember(*memory)

                # 学习
                self.agent.learn(episode=e)

            if e % 500 == 0 and e != 0:
                savepath = self.agent.save(episode=e)
                self.checkpoint_savepaths.append(savepath)
                if len(self.checkpoint_savepaths) >
self.max_save_checkpoints:
                    to_delete = self.checkpoint_savepaths.popleft()
                    try:
                        os.remove(to_delete)
                    except Exception as _:
                        tqdm.write(f"Failed to delete old checkpoint
\"{to_delete}\"")
            if e % 100 == 0 and e != 0:
                if self.writer is not None:
                    start_time = time.time()
                    win_rate, draw_rate, avg_steps, avg_rewards =
evaluate_win_rate(
                        self.env.create_eval(),
                        self.agent.create_eval(),
                        rounds=20  # 神经网络对于MiniMax这种没有随机性的算法总会
产生一样的输出，但我们起手是随机的
                    )
                    """
                    tqdm.write(
                        f"Evaluation at Episodes {e} Got: " +
                        f"Win Rate={win_rate}; Draw Rate={draw_rate}; " +
                        f"Avg Steps={avg_steps}; Avg
Rewards={avg_rewards}")
                        """
                    end_time = time.time()
                    # self.writer.add_scalar('Train/WinRate', win_rate, e)
                    self.writer.add_scalars('Evaluations/Rate', {
                        'Win Rate': win_rate, 'Draw Rate': draw_rate, 'Loss
Rate': 1.0 - draw_rate - win_rate,
                        'Non-Loss Rate': win_rate + draw_rate
                    }, e)

                    self.writer.add_scalar('Train/Evaluations/Avg Steps',
avg_steps, e)
                    self.writer.add_scalar('Train/Evaluations/Avg
Rewards', avg_rewards, e)
                    # tqdm.write(f"Evaluation at Episode {e} spent
{end_time - start_time} seconds.")

            self.agent.finish(episode=e, max_episode=episodes)
            avg_reward = total_reward / steps
            if self.writer is not None:
                self.writer.add_scalar('Train/Reward', avg_reward, e)
                self.writer.add_scalar('Train/Steps', steps, e)
        except KeyboardInterrupt:
            # self.agent.save(f"models/gomoku_ai_{e + 1}.pt")
            break
```

3.1.4 棋盘评估过程代码实现

```
def get_lines_to_check(board: np.ndarray, x, y, pattern_len) ->
np.ndarray:
    board_size = board.shape[0]
    lines = []
    # 横向
    start_x = max(0, x - pattern_len + 1)
    end_x = min(board_size - pattern_len + 1, x + 1)
    for i in range(start_x, end_x):
        lines.append(board[i:i + pattern_len, y])
    # 纵向
    start_y = max(0, y - pattern_len + 1)
    end_y = min(board_size - pattern_len + 1, y + 1)
    for i in range(start_y, end_y):
        lines.append(board[x, i:i + pattern_len])
    # 对角线
    diag_k = y - x
    diag_s = x if diag_k >= 0 else y
    diag = board.diagonal(diag_k)
    start_i = max(0, diag_s - pattern_len + 1)
    end_i = min(len(diag) - pattern_len + 1, diag_s + 1)
    for i in range(start_i, end_i):
        lines.append(diag[i:i + pattern_len])
    # 斜对角线
    anti_diag = np.diag(np.fliplr(board), board_size - y - x - 1)
    start_i = max(0, diag_s - pattern_len + 1)
    end_i = min(len(anti_diag) - pattern_len + 1, diag_s + 1)
    for i in range(start_i, end_i):
        lines.append(anti_diag[i:i + pattern_len])
    lines = np.unique(np.asarray(lines), axis=0)
    return lines


class Board:
    PATTERNS = {
        'L5': [[1, 1, 1, 1, 1]],  # 连五，即获胜局面
        'H4': [[0, 1, 1, 1, 1, 0]],  # 活四，即两个点可以成连五
        'C4': [[0, 1, 1, 1, 1], [1, 0, 1, 1, 1], [1, 1, 0, 1, 1]],  # 冲四，
仅有一个点可以连五
        'H3': [[0, 1, 1, 1, 0, 0], [0, 1, 0, 1, 1, 0]],  # 活三，能形成活四的三
        'M3': [[1, 1, 1, 0, 0], [0, 1, 0, 1, 1], [0, 1, 1, 0, 1]],  # 眠三，
只能形成冲四的三
        'H2': [[0, 0, 1, 1, 0, 0], [0, 1, 1, 0, 0, 0], [0, 1, 0, 1, 0, 0]],
# 活二，能形成活三的二
        'M2': [[0, 0, 0, 1, 1], [0, 0, 1, 0, 1], [0, 0, 1, 1, 0], [0, 1, 0,
0, 1]]  # 眠二，能形成眠三的二
    }
    PATTERNS_FLATTEN = [item for sub in PATTERNS.values() for item in sub]
    PATTERN_LENS = set(map(len, PATTERNS_FLATTEN))
    PATTERN_MAX_LEN = max(PATTERN_LENS)
    PATTERN_MIN_LEN = min(PATTERN_LENS)

    def __init__(self, board_size=8, win_size=5):
        self.board = np.zeros((2, board_size, board_size), dtype=int)
        self.board_size = board_size
        self.win_size = win_size
        self.steps = 0
```

```
    def get_state(self) -> np.ndarray:
        return self.board

    def reset(self):
        self.board = np.zeros(self.board.shape, dtype=int)
        self.steps = 0

    def put(self, x, y, player) -> str | None:
        """
        :return: 返回当前着子产生的棋形
        """
        if (self.board[0, x, y] + self.board[1, x, y]) != 0:
            raise Exception(f"Player {player} 试图下在有子的位置 {x},{y}")
        self.board[player - 1, x, y] = 1
        pattern = self.find_pattern(x, y, player)
        self.steps += 1
        # is_draw = self.steps == self.board_size ** 2 and pattern != 'L5'
        # return (self.PATTERNS_REWARD[pattern] if pattern is not None else
-150 if is_draw else 0,
        #         is_draw or pattern == 'L5')
        return pattern

    def is_done(self):
        return self.steps == self.board_size ** 2

    def is_empty(self):
        return self.steps == 0

    def get_valid_moves(self) -> np.ndarray:
        """
        获取当前有效的落子区域，一个 Mask，-inf 代表不可落子，1 代表可以落子
        :return: 返回一个-inf 和 1 组成的二维数组
        """
        valid_moves = (self.board == 0).astype(int)
        valid_moves[self.board != 0] = -1
        valid_moves = valid_moves * np.inf
        valid_moves[valid_moves == np.inf] = 1
        return valid_moves[0] * valid_moves[1]

    def get_suggested_moves(self, player, extend=1) -> np.ndarray:
        """
        获取当前建议的落子区域，一个 Mask，-inf 代表不建议落子，1 代表建议落子。
        建议的落子区域为当前棋盘上有子存在的矩形外围一格
        :return: 返回一个-inf 和 1 组成的二维数组
        """
        if self.steps == 0:
            return np.full((self.board_size, self.board_size), 1)
        pieces = np.argwhere(self.get_valid_moves() == -np.inf)
        min_x, min_y = pieces.min(axis=0)
        max_x, max_y = pieces.max(axis=0)
        min_x = max(0, min_x - extend)
        min_y = max(0, min_y - extend)
        max_x = min(self.board_size, max_x + extend + 1)
        max_y = min(self.board_size, max_y + extend + 1)
        suggested_moves = np.zeros((self.board_size, self.board_size))
        suggested_moves[min_x:max_x, min_y:max_y] = 1
        suggested_moves[suggested_moves == 0] = -np.inf
        return suggested_moves

    def find_pattern(self, x, y, player) -> str | None:
```

```
        """
        落子在 x, y 时产生的最佳棋形
        """
        board = self.board[player - 1]
        is_a_attempt = board[x, y] == 0
        if is_a_attempt:
            board[x, y] = 1
        to_check_lines = {}
        for pattern_len in self.PATTERN_LENS:
            lines = get_lines_to_check(board, x, y, pattern_len)
            to_check_lines[pattern_len] = lines
        for pattern_key, pattern_list in self.PATTERNS.items():
            for pattern in pattern_list:
                pattern_len = len(pattern)
                lines = to_check_lines[pattern_len]
                for line in lines:
                    if (line == pattern).all() or (line ==
list(reversed(pattern))).all():
                        if is_a_attempt:
                            board[x, y] = 0
                        return pattern_key
        if is_a_attempt:
            board[x, y] = 0
        return None

    @staticmethod
    def beautify_board(board: np.ndarray, mask_mode=False,
number_mode=False, space=' ') -> str:
        result = '    ' + Fore.LIGHTBLUE_EX + space.join(map(str,
range(board.shape[0]))) + Style.RESET_ALL + '\n'
        i = 0
        def get_symbol(i):
            non = Fore.WHITE + Style.BRIGHT + '.' + Style.RESET_ALL
            cross = TermColor.BOLD + Fore.RED + Style.BRIGHT + 'X' +
Style.RESET_ALL
            circle = TermColor.BOLD + Fore.GREEN + Style.BRIGHT + 'O' +
Style.RESET_ALL
            if mask_mode:
                return cross if i == -np.inf else circle if y == 1 else non
            elif number_mode:
                return str(i)
            else:
                return {'.': non, 'X': cross, 'O':
circle}[Player.PlayerSymbol(i)]
        for x in board.transpose((1, 0)):
            result += Fore.LIGHTBLUE_EX + '{:<4d}'.format(i) +
Style.RESET_ALL
            i += 1
            for y in x:
                result += get_symbol(y) + space
            result += "\n"
        return result

    def __str__(self):
        return Board.beautify_board(self.board[0] + self.board[1] * 2)

Board.PATTERNS_BY_LEN = {
    pattern_len: list(filter(lambda p: len(p) == pattern_len,
Board.PATTERNS_FLATTEN))
    for pattern_len in Board.PATTERN_LENS
}
```

3.1.5 奖励计算过程代码实现

```
def calculate_reward(pattern: str | None,
                     break_pattern: str | None,
                     rival_pattern: str | None,
                     terminal_status: int | None) -> float:
    if pattern is None and break_pattern is None and rival_pattern is None:
        return 0
    if terminal_status is not None:
        if terminal_status == 0:
            return FINAL_STATUS['draw']
        elif terminal_status == 1:
            return FINAL_STATUS['win']
        else:
            return FINAL_STATUS['loss']
    reward = 0
    if pattern is not None:
        # 构成了特定的棋形
        reward += PATTERNS_REWARD[pattern]
    if rival_pattern is not None:
        # 对手形成了棋形
        rival_reward = PATTERNS_REWARD[rival_pattern]
        reward -= rival_reward * (
            PATTERNS_GAMMA['rival'][0] if rival_reward <= reward
            else PATTERNS_GAMMA['rival'][1] if rival_reward <= 60
            else PATTERNS_GAMMA['rival'][2])
    if break_pattern is not None:
        # 打破了对手的棋形
        assume_reward = PATTERNS_REWARD[break_pattern]
        reward += assume_reward * (
            PATTERNS_GAMMA['break'][0] if assume_reward <= 20
            else PATTERNS_GAMMA['break'][1] if assume_reward <= 60
            else PATTERNS_GAMMA['break'][2])
    return reward
```

3.1.6 五子棋环境过程核心代码实现

```
# 五子棋环境
class GomokuEnv(BaseGomokuEnv):
    def __init__(self, rival_agent: BaseAgent, board_size=8, win_size=5,
gui_board: GUIBoard = None):
        super().__init__(rival_agent, board_size, win_size, gui_board)
        self.board = Board(board_size=board_size, win_size=win_size)
        self.current_player = Player.BLACK
        self.winner = None
        self.done = False

    def reset(self) -> Board:
        """
        重置环境状态
        """
        self.board.reset()
        self.current_player = Player.BLACK
        self.winner = None
        self.done = False
        if self.gui_board is not None:
            self.gui_board.reset()
        return self.board
```

```
    def step(self, action: tuple[int, int] | int) -> tuple[Board, float,
int | None]:
        """
        :param action: 一个由着子位置 X 和 Y 组成的元组或 int:X*size + Y
        :returns:
            - board: 着子之后的棋盘状态
            - reward: 奖励分数
            - terminal_status: 棋局结束状态，None 代表未结束，1 代表胜利，-1 代表失
败，0 代表平局
        """

        # 先手 action
        x, y = action if isinstance(action, tuple) else divmod(action,
self.board_size)
        pattern = self.board.put(x, y, self.current_player)
        is_done = self.board.is_done()
        if self.gui_board is not None:
            self.gui_board.draw_piece(x, y, self.current_player)
        if is_done or pattern == 'L5':
            # 如果棋局结束，判断是否此着形成连五，否则平局
            terminal_status = 1 if pattern == 'L5' else 0
            reward = Reward.calculate_reward(pattern, None, None,
terminal_status)
            if self.gui_board is not None:
                self.gui_board.info(f"AI 落子于({x},{y})，奖励：{reward}")
            return self.board, reward, terminal_status
        # 切换玩家
        self.current_player = Player.SwitchPlayer(self.current_player)
        # 先手的 action 可以打断的后手玩家的棋形
        break_pattern = self.board.find_pattern(x, y, self.current_player)
        # 后手 action
        rival_action = self.rival_agent.act(self.board,
self.current_player)
        rx, ry = rival_action if isinstance(rival_action, tuple) else
divmod(rival_action, self.board_size)
        rival_pattern = self.board.put(rx, ry, self.current_player)
        is_done = self.board.is_done()
        terminal_status = None
        if is_done or rival_pattern == 'L5':
            terminal_status = -1 if rival_pattern == 'L5' else 0
        reward = Reward.calculate_reward(
            pattern, break_pattern, rival_pattern, terminal_status)
        if self.gui_board is not None:
            self.gui_board.draw_piece(rx, ry, self.current_player)
            self.gui_board.info(f"AI 落子于({x},{y})，奖励：{reward}")
        # 切换玩家
        self.current_player = Player.SwitchPlayer(self.current_player)
        return self.board, reward, terminal_status

    def create_eval(self) -> 'GomokuEnv':
        """
        :return: 以当前环境参数创建的评估环境
        """
        return GomokuEnv(
            self.rival_agent.create_eval(),
            self.board_size,
            self.win_size,
            self.gui_board
        )
```

3.2 实验结果

由于硬件设备和作业时间限制，本作业仅对 7x7 大小的五子棋棋盘进行了十万局与 MiniMax 智能体（搜索深度为二层）对弈的训练。训练中的数据如下：

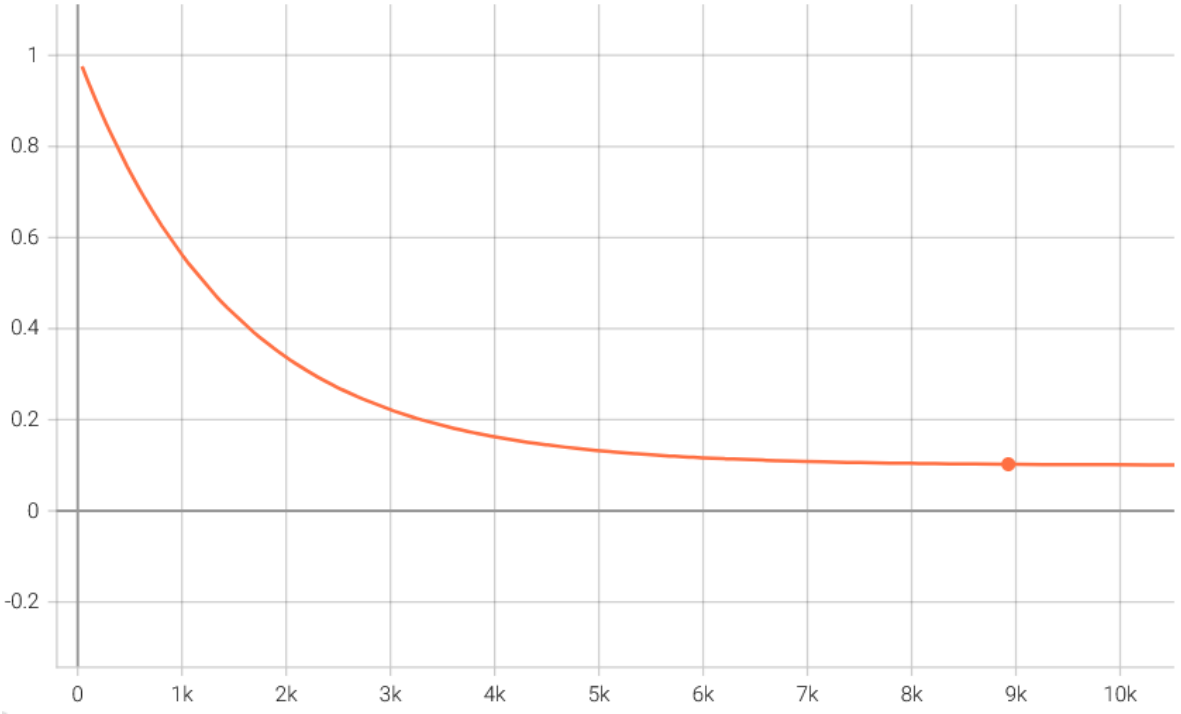

图 9 一万步以内探索率衰减曲线

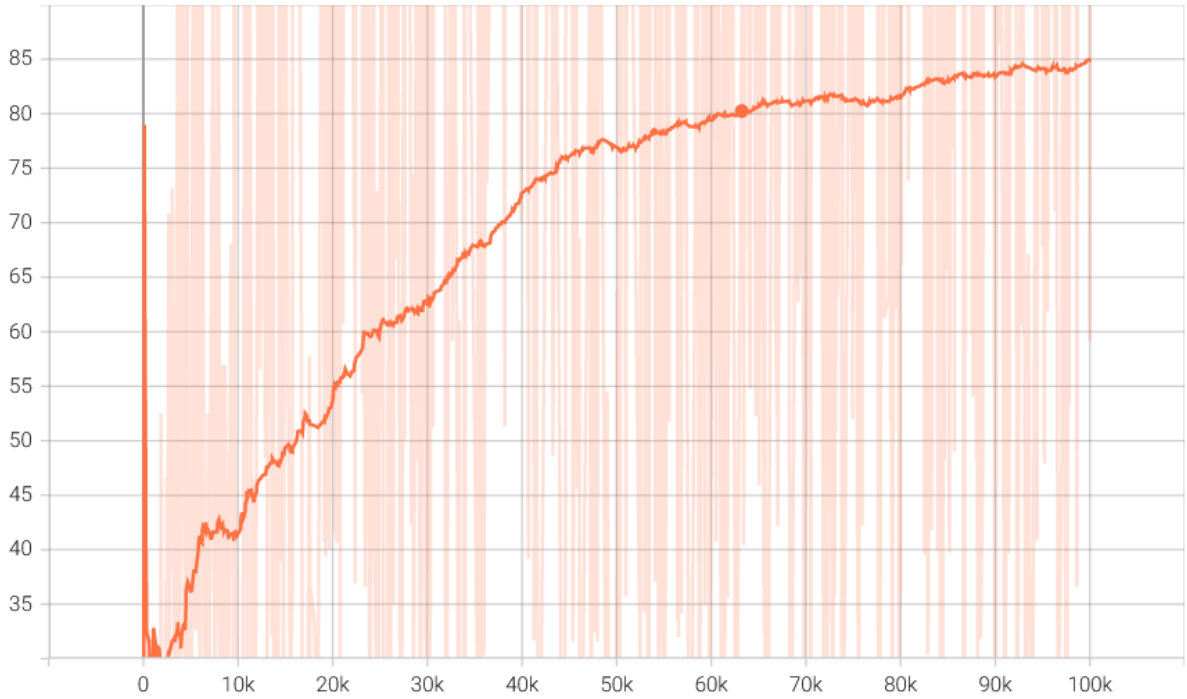

图 10 训练中获得的奖励平滑曲线

图 11 训练中的每局步数平滑曲线

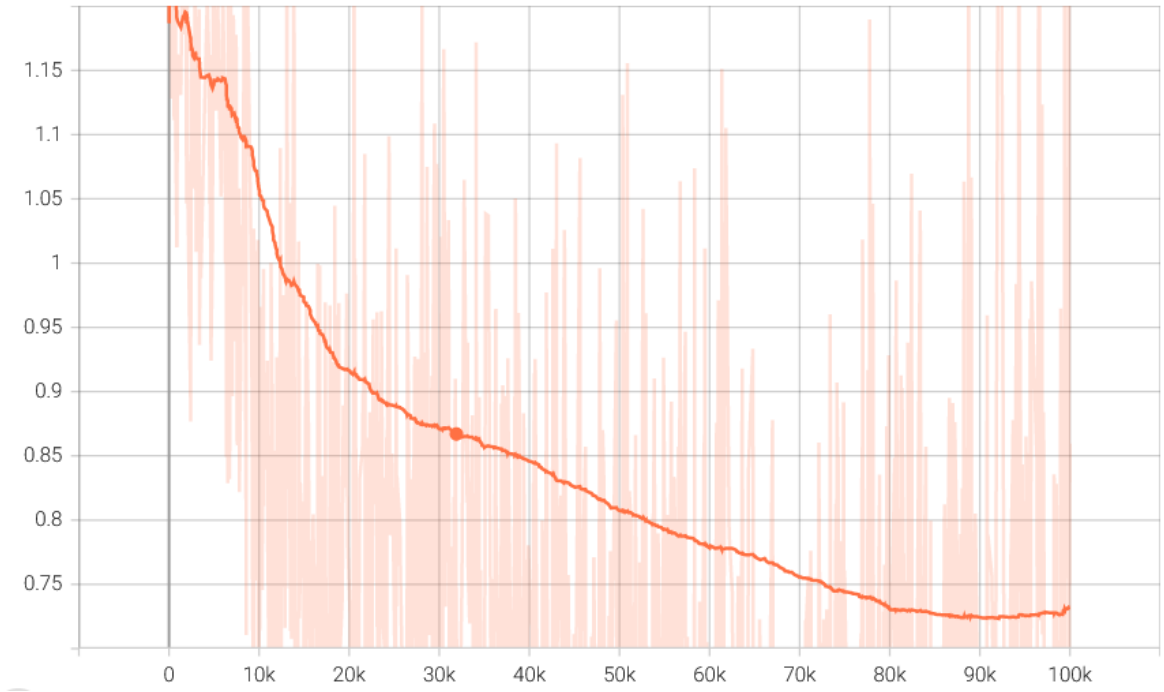

图 12 训练中的 Loss 值的平滑曲线

值得一提的是，在深度强化学习中，由于并不存在传统的“训练集”，而损失函数的计算也是通过变动的经验池中的随机抽样，所以 Loss 值并不一定会收敛，也有文献证明了 Q-Learning 的值函数近似是不可能收敛的。本作业的试验结果以训练中进行的评估为准。

在训练中，每间隔 100 次棋局，便对训练的神经网络采用不使用ϵ-贪婪策略而完全使用值网络进行评估（为保证随机性，智能体第一次落子采用随机落子，否则智能体在棋盘为空的情况下会固定下在某一位置）的情况如下：

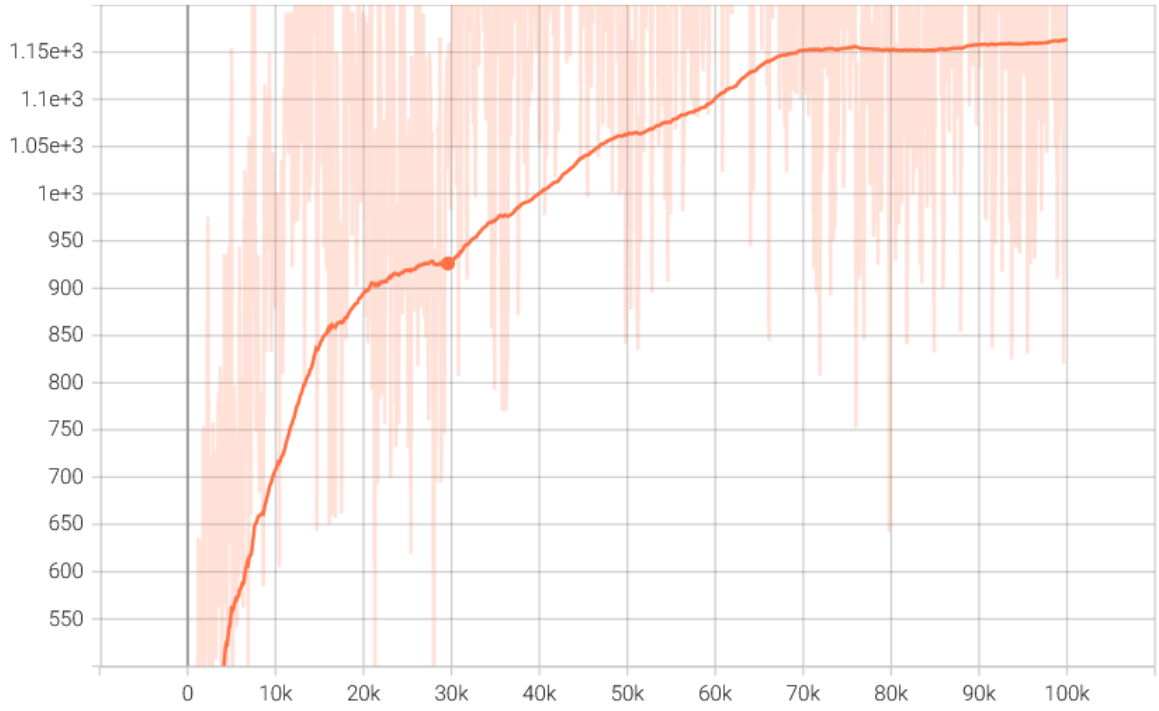

图 13 每局获得的总计奖励的平滑曲线

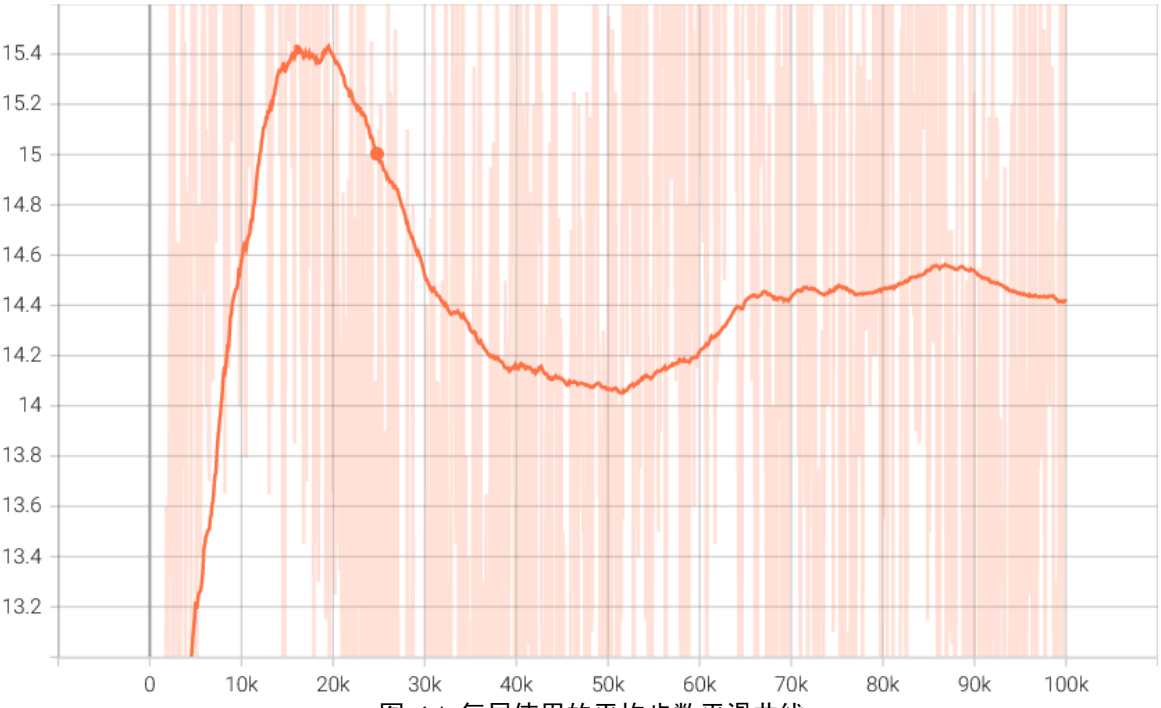

图 14 每局使用的平均步数平滑曲线

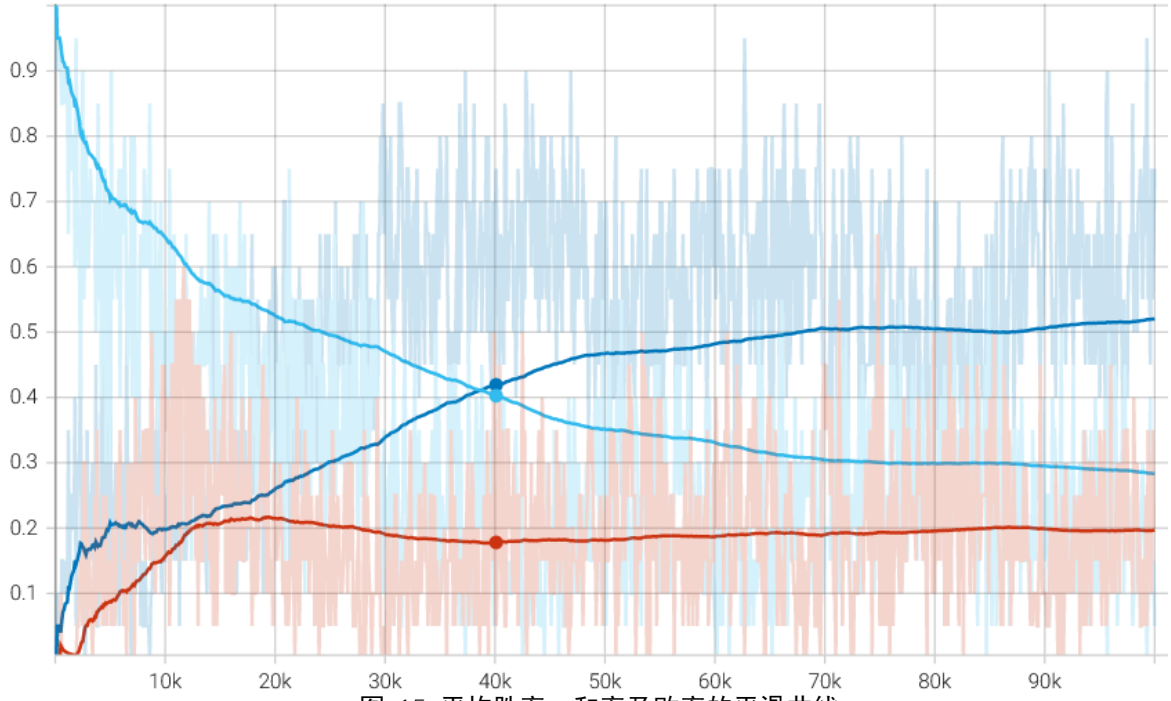

图 15 平均胜率、和率及败率的平滑曲线

（上图中深蓝色曲线代表智能体在评估中对战 MiniMax 智能体的胜率，淡蓝色曲线代表败率而棕色曲线代表和率。）

可以看到在十万次的训练中，DQN 智能体表现出了一定的智能，并呈现明显的上升趋势，与二层搜索深度的 MiniMax 进行对战，最高可以取得 95%的胜率。

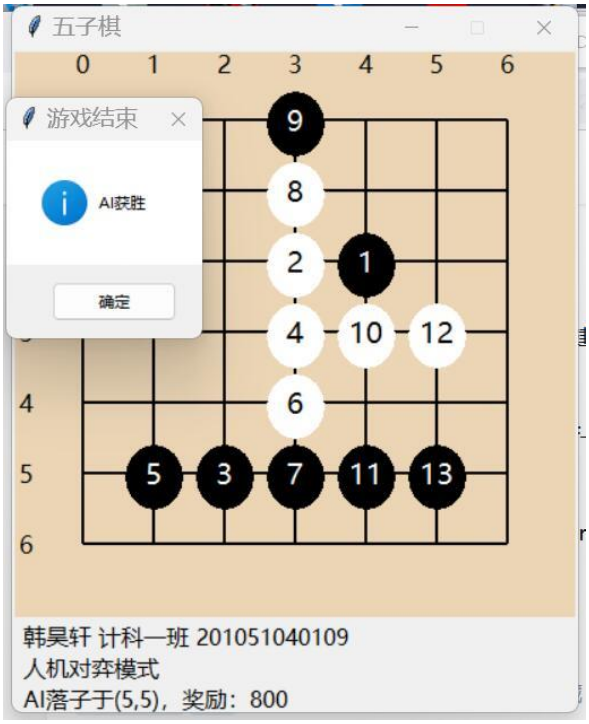

图 16 人机对战中的智能体步子分布

通过人机对战检验学习成果，发现智能体已经形成防活三、堵冲四，阻挠对手连五，并在对手出现活三而自己有连五机会的时候果断选择连五。可以说，在经过十万次的训练之后，智能体已经有着较为高等级的五子棋智能水平了。

四、总结

4.1 总结

本作业基于深度强化学习中的 DQN（Deep Q Network）算法实现了一个五子棋人工智能体，并实现了相应的五子棋游戏环境和训练代码，以及建立相应的奖励数值模型。并采取多种手段优化了智能体训练的效率和效果。最终得到了一个效果良好的基于神经网络模型的五子棋智能体，并对其性能进行了相应的测试。

4.2 设计过程中遇到的问题及解决方案

在实现初期对输入棋盘状态未进行分离，导致神经网络无法收敛。解决方案：将棋盘状态分割成己方落子和对方落子两个通道。

智能体陷入局部最优。解决方案：探索率折损算法导致探索率降低过快，通过修正探索率折损算法，放缓探索率的降低速度解决。

神经网络出现过拟合现象，解决方案：学习率过高导致的，降低学习率解决了过拟合现象。

训练过程中 Loss 下降顺利但是评估效果差，解决方案：网络结构太小，导致无法拟合值函数，通过改进网络结构解决。

在调参时奖励系数设置过高会导致智能体训练时过于期待未来价值，不适合本作业中无禁手的小规模五子棋短平快的特征，降低奖励系数可以解决。在测试初期智能体可以在小规模的 4x4 棋盘（三子连线胜利）上表现良好，但在 8x8 的棋盘上表现不佳，通过改进奖励机制，引入更多先验知识后解决。

同时，DQN 算法在本作业实现中多次遇到问题，通过查询资料和阅读论文后解决。

4.3 未来展望

由于硬件设备限制，本作业未能实现在标准 15x15 棋盘上的智能体训练，且目前训练的智能体表现尚未达到最优。

本作业采取的各种评估和优化手段可能会对智能体产生一些负面的局限性。

本作业使用的 DQN 算法为 2015 年原作者基于 DQN 原始论文引入目标网络后的版本，后续应尝试 DuelDQN、DoubleDQN、RainbowDQN 等最近提出的 DQN 算法。

本作业所构建的数值网络可能在棋类游戏中表现不佳，后续可以尝试实现策略网络相关算法。

本作业起初尝试过智能体的自我对弈，但在大棋盘上表现不佳。

本作业为了解决奖励稀疏的问题在训练阶段引入了先验知识（例如动态奖励计算、MiniMax 智能体辅助训练等），后续可以开发新的经验回放算法以保证在全程无先验知识的情况下解决 DQN 对弈问题。