深圳大学考试答题纸

(以论文、报告等形式考核专用)  
二○二 四 ～二○二 五 学年度第 1 学期

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 课程编号 | 15019900006 | | 课程名称 | | 人工智能导论 | | 主讲教师 | | 高灿 | 评分 |  |
| 学 号 |  | | 姓名 |  | | 专业年级 | |  | | | |
|  | | | | | | | | | | | |
| 教师评语： | | | | | | | | | | | |
| 题目： | | 基于DQN算法的俄罗斯方块游戏AI智能体 | | | | | | | |  | |

**基于DQN算法的俄罗斯方块游戏AI智能体**

**【摘要】**

本报告研究并实现了一个基于深度Q网络（DQN）算法的俄罗斯方块游戏AI智能体。通过使用DQN来学习游戏策略，我实现了一个可以自动玩俄罗斯方块游戏的AI模型。报告详细介绍了DQN算法的原理和应用、实验中模型架构、实验的过程以及实验结果与分析。

【**正文**】

**第一章 简介及相关工作**

**1.1** 简介

俄罗斯方块（Tetris）是一款经典游戏，其主要玩法是控制下落的方块形状和位置，使之在底部形成完整的行，并消除这一行以取得分数。在游戏中，玩家通过旋转、移动方块来尽可能多地消除行数以获得分数，并且还要防止方块堆积到屏幕顶部，否则游戏结束。

**1.2** 相关工作

在人工智能领域，强化学习（Reinforcement Learning, RL）方法已被广泛应用于游戏AI的研究。特别是Deep Q-Network（DQN）算法，通过结合卷积神经网络（CNN）来逼近Q值函数，在经典的Atari 2600游戏中取得了显著成果。该算法使智能体能够从游戏状态中自动学习策略，并有效完成游戏目标。

在俄罗斯方块游戏AI的研究中，也有学者们探索了多种算法，包括基于规则的系统、传统的机器学习方法和强化学习技术。具体而言，Q学习（Q-learning）、蒙特卡洛树搜索（MCTS）和深度Q网络（DQN）等算法都被应用于自动游戏AI的构建。其中，DQN因其简单易懂且能够取得良好性能，成为初学者的理想选择。通过将深度学习的特征提取能力与Q学习的决策过程相结合，DQN能够有效应对俄罗斯方块游戏中的高维状态空间。

**第二章 DQN算法原理及方法分析**

**2.1** 深度Q网络（DQN）原理

DQN算法是一种结合了深度学习和Q\_learning的强化学习算法。它的核心思想是使用深度神经网络（CNN）来近似Q函数，通过最大化累积奖励来学习最优策略。DQN算法能够处理高维、连续的状态空间，这使得它非常适合于复杂的游戏环境，如俄罗斯方块。

**2.1.1** Q\_Learning算法

Q\_Learning是一种无模型的强化学习算法，它通过迭代更新Q函数来学习最优策略。Q函数，也称为动作价值函数（Action-Value Function），它用于评估在特定的状态下采取某个特定动作的长期期望回报（即效用）。通常情况下，算法会将当前环境状态输入Q函数，然后选择输出效用最高的动作反馈给环境，如此循环。Q\_Learning 的核心更新规则：

其中：

* 表示当前的环境状态；
* 表示智能体当前采取的动作；
* 表示当前状态下采取动作的Q值（效用）；
* 表示学习率；
* 表示在状态下采取动作后智能体获得的即时奖励；
* 表示折扣因子，较大的值表示未来奖励的重要性更高，较小的值则表示更关注当前的即时奖励；
* 表示在下一状态下，智能体根据策略所能选择的动作中，Q值的最大值。

可以看到，Q\_Learning是基于经验来优化未来的决策。该公式将当前的Q值与实际获得的即时奖励和未来可能获得的最大累积奖励进行比较，利用这一差值来修正当前的 Q值估计。通过反复迭代和学习，Q值函数逐渐收敛，最终能够准确地表示每个状态-动作对的长期价值，从而指导智能体选择能获得最高累积奖励的最优动作。

**2.1.2** DQN算法

DQN算法则是基于Q\_Learning算法，使用深度神经网络来近似Q函数，从而克服了传统 Q-learning在复杂环境中的局限性。传统Q-learning的Q函数需要以表格形式存储所有状态-动作对的Q值，这种用表格存储动作价值的做法只在环境的状态和动作都是离散的，并且空间都比较小的情况下适用，而对于状态空间大的复杂环境，以俄罗斯方块为例，光表示当前布局就有种状态，每一个状态用表格来存储是不现实的。而DQN用Q神经网络代替表格，通过学习状态到Q值的映射，能够处理高维、连续的状态空间。

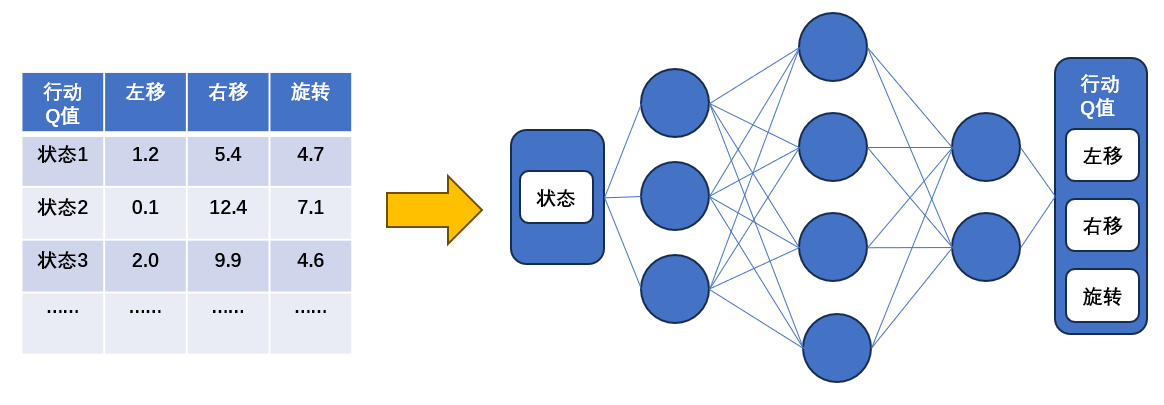


图 1 Q\_Learning到DQN的Q函数变化

**2.1.3** Experience replay经验回放

在Q\_learning算法中，每一个数据只会用来更新一次值。为了更好地将Q\_learning和深度神经网络结合，DQN算法采用了经验回放（experience replay）方法，具体做法为维护一个回放池，将每次与环境交互产生的经验（状态、动作、奖励、下一状态）存储到回放池中，训练Q网络的时候再从回放池中随机采样若干数据来进行训练。

采用经验回放有两个优势：

* 传统 Q-learning 会按照时间顺序在线更新 Q 值，而连续的样本通常高度相关，容易导致模型训练的不稳定性和效率低下。经验回访通过打乱样本的时间顺序，使训练数据更接近于独立同分布，从而减小样本间的相关性，提升模型的泛化能力。
* 可以重复利用历史经验，降低样本收集的频率，提升数据使用效率和训练稳定性。

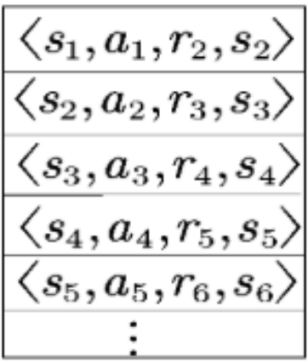


图 2 经验回放池

**2.1.4** Target Network目标网络

目标网络是DQN算法通过引入一个与主网络结构相同但参数固定的网络，其用于计算目标Q值，并定期将主网络的参数复制到目标网络中。如此可以让目标Q值在一定时间内保持不变，减少了训练时目标值波动带来的不稳定性，从而加速收敛并提高模型的稳定性和训练效果。

**2.2** 方法分析  
 下面将DQN算法应用到俄罗斯方块游戏中。

**2.2.1** Q网络

一开始，我将方块在下落过程中的每一帧的游戏状态（当前游戏区域布局+当前下落方块的形状和下落位置）作为Q网络的输入，输出是那一帧方块左移一格、右移一格、顺时针旋转90度的Q值作为输出。但经过实践发现，由于游戏区域布局和下落方块状态的组合状态非常多，导致状态空间巨大，从而使 Q 网络难以充分探索和学习这些状态。

接下来，我将新方块出现时的第一帧游戏状态（当前游戏区域布局+当前下落方块的形状）作为Q网络的输入。在俄罗斯方块游戏中，我们只需要决定方块左移或右移多少格，以及顺时针旋转90度多少次，即可确定方块最终下落至底部的位置和状态。因此，对于输出，与其在每一帧都输出方块当前可能的动作（例如左移一格、右移一格、顺时针旋转90度等）的Q值，不如直接输出整个下落过程的最终决策（例如总共左移几格、右移几格以及顺时针旋转90度几次）的Q值。但实践发现，对于每一个方格，其左右移动的范围大致为[-5,5]（方块初始位置在网格顶部中央，且游戏区域宽为10格），旋转次数为[0,3]（旋转4次及以上的状态相当于旋转该次数对4取模后状态），输出的维度为11×4，维度过高，这种高维输出可能使得网络难以有效地学习所有动作的 Q 值，同时还导致了训练效率的大幅下降。

最后，我将当前下落方块下落到底的游戏状态（当前游戏区域布局+当前下落方块的形状+堆叠方块的最大高度+方块布局的凹凸程度+堆叠方块下的空洞数+当前方块下落后消除的层数）作为Q网络的输入，将该状态的Q值作为输出的值，这不仅解决了状态空间过大的问题，还解决了输出维度过大的问题。每次需要决策时，只需要生成当前下落方块所有合法动作（左移或右移多少格、顺时针旋转90度多少次）后的最终下落游戏状态，将这些状态输入Q网络，选择Q值最大的游戏状态对应的动作作为本回合的决策。

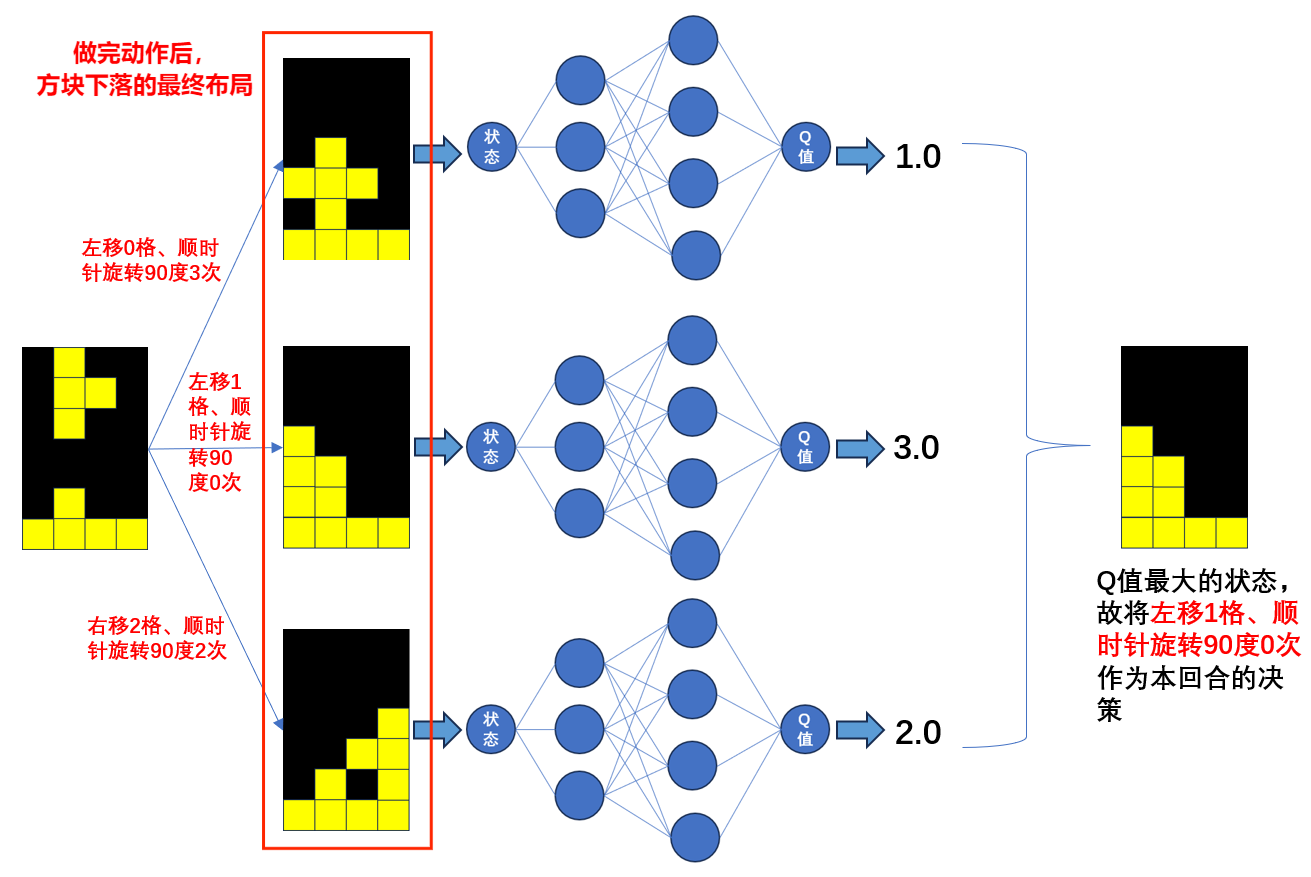


图 3 决策架构图

**2.2.2** 状态表示

根据上面Q网络的设计，游戏的状态由以下三个部分组成：

* 游戏区域布局：在我们设计的俄罗斯方块，其游戏区域宽为10格，长为20格，所以我们可以将游戏区域布局转换为一个10×20的二维数组，其中数组中的每个元素只能为0或1，0表示在该区域内不存在方块，1表示在该区域内存在方块。

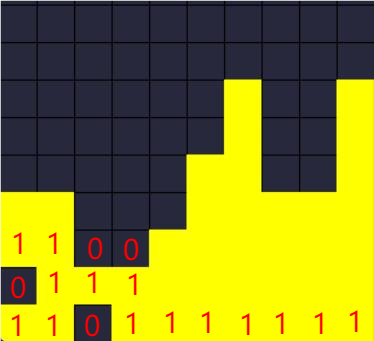


图 4 游戏区域布局状态部分展示

* 当前下落方块：在俄罗斯方块中一共有7种不同形状的方块，每种方块均由4个小方块组成，我们同样也可以用一个4×4的二维数组来表示当前下落方块，根据不同形状的方块以及其当前旋转状态，在二维数组中用0表示不存在小方块，1表示存在小方块。

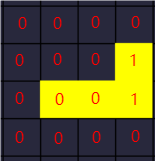


图 5 当前下落方块状态

* 游戏区域布局特征：除了游戏区域布局作为状态外，我还将游戏区域布局中的几个重要特征（堆叠方块的最大高度、方块布局的凹凸程度、堆叠方块下的空洞数、当前方块下落后消除的层数）提取出来作为状态。这是因为堆叠方块的高度决定了游戏失败的风险，空洞数会影响未来方块的堆叠效率，凹凸程度则影响整体布局的平整性，当前方块下落后消除的层数决定了本回合获得的分数。这些特征在很大程度上决定了游戏的最终得分，将它们提取出来并作为状态有助于强化智能体对关键因素的关注，优化其决策能力。

综上，经维度压缩后，状态表示为一个220×1的矩阵（游戏区域布局10×20=200，当前下落方块4×4=16，游戏区域布局特征4，200+16+4=220）。

**2.2.3** 动作空间

俄罗斯方块游戏中的动作空间包括旋转方块、向左移动和向右移动。在我设计的算法中，动作由一个元组组成：

* R表示一个方块顺时针旋转90度次数的变量，其取值在[0,4)之间，因为一个方块顺时针旋转90度4次后相当于没有旋转。
* M表示一个方块左右移动的格数，其取值根据每个方块形状和旋转状态的不同而不同，但大多取值都集中在[-5,5]之间（方块初始位置在网格顶部中央），取值为负表示向左移，取值为正表示向右移。

**2.2.4** 奖励

每次智能体做完决策并更新游戏布局后，都可以获得奖励。设消除的行数为n，那么获得的奖励为

除了奖励外，还需要进行惩罚，设更新后的游戏布局相对于原本的游戏布局的空洞数增加了a、方块布局的凹凸程度增加了b，则获得的惩罚为

设置惩罚的原因是为了引导智能体在追求即时奖励（如消除方块得分）的同时，更关注游戏布局的长期优化。空洞数和凹凸程度的增加会导致布局质量下降，使后续方块难以合理堆叠，增加游戏失败的风险。通过对这些不良布局行为施加惩罚，智能体可以学会减少空洞、保持布局平整，从而延长游戏时间、提高整体得分策略的可持续性。

最终，智能体每做完一次决策后得到的最终奖励为

值得注意的是，虽然设置惩罚可以有效降低游戏结束的风险，提高持续得分的可能性，但惩罚机制也会削减整体奖励值。在Q网络训练尚不充分、决策能力较弱的初期，智能体可能难以通过消除行获得正向奖励，导致总体奖励为负。这种负奖励可能对训练过程产生不利影响：首先，智能体可能过度惧怕探索某些潜在有利的状态和动作，从而导致策略陷入局部最优；其次，负奖励的累积会干扰Q值的更新，导致网络难以准确估计未来回报。这些影响均会降低训练效率，使模型难以快速优化。

**2.2.5** 规则决策

在我设计的算法中，智能体除了利用Q值最大的状态来做决策（即贪婪策略）外，还引入了一种基于规则的决策方式。智能体会先遍历所有候选动作对应的状态，检查每个状态中清除的行数。如果有动作可以清除更多的行（清除行数至少大于等于1），智能体会直接选择该动作，而不再依赖Q值计算结果。这种规则决策的优先级高于Q值决策。

根据下面不使用规则决策和使用规则决策下训练10000轮的训练结果，不难发现，规则可以帮助智能体更快学习到清除行的重要性，从而在前期获得更高的奖励，同时整个训练过程获得奖励也高于不使用规则的训练过程。

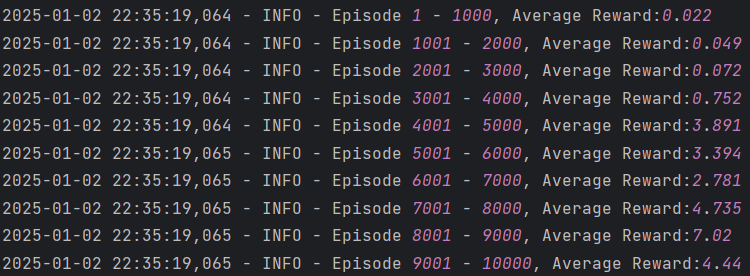


图 6 不使用规则决策训练10000轮的结果

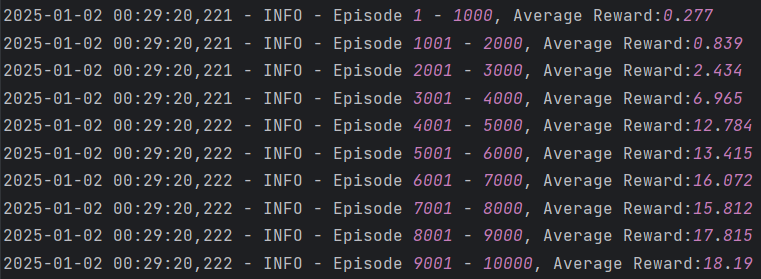


图 7 使用规则决策训练10000轮的结果

**第三章 实验论证**

**3.1** 实验过程

**3.1.1** 创建游戏环境

在Tetris.py中使用Pygame创建了俄罗斯方块游戏环境。Tetris为游戏的核心类，负责整体游戏的初始化、方块的生成、分数计算、方块的自动下落、操作的响应等。它还包含了显示游戏界面的功能，并支持分数、时间和消除行数的展示。



图 8 俄罗斯方块游戏运行环境

**3.1.2** 初始化训练环境

创建主Q网络（用于预测Q值）、目标Q网络（用于计算目标Q值，且稳定训练，避免Q值震荡）和游戏环境，初始化优化器和经验回放池以及训练所需的各参数。



图 9 初始化训练环境

Q网络由4层全连接层组成，其中前三层作为隐藏层，每层的输出分别为1024、2048和1024个神经元，隐藏层之间使用ReLU激活函数以增加网络的非线性表达能力。最后一层为输出层，输出一个标量值，表示输入状态的Q值估计。

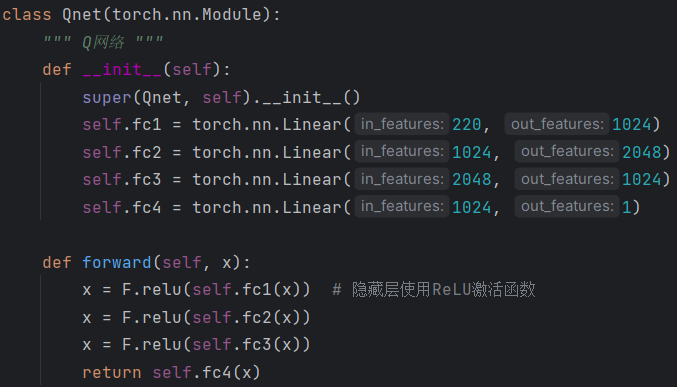


图 10 Q网络架构

**3.1.3** 智能体与环境交互

初始化游戏环境env.reset(False)并获得初始状态state，开始游戏循环直至游戏结束，每轮循环让智能体进行决策和交互，并存储经验：

* 调用env.get\_next\_states()获取所有可能的下一步状态。
* 使用agent.take\_action(next\_states)选择一个动作（根据规则和epsilon-贪婪策略）。



图 11 智能体决策动作

* 通过env.step(action)与环境交互，执行动作并获得奖励reward和游戏是否结束标志done。奖励会额外扣除因空洞数和凹凸程度增加带来的惩罚，用于引导智能体优化游戏布局。
* 将当前经验（状态、动作、奖励、下一状态、是否结束）存入经验回放池。



图 12 智能体与环境交互

**3.1.4** 训练模型

* 随机从经验回放池中采样一个批次的数据：batch = random.sample(self.replay\_buffer, batch\_size)。每个批次包括：当前状态 state、执行动作 action、收到的即时奖励 reward、下一状态 next\_state、当前状态是否为终止状态 done。

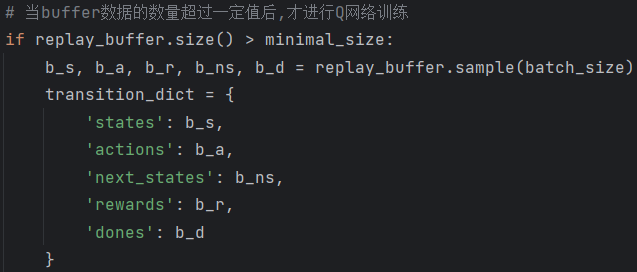


图 13 采样经验

* 将采样的数据拆分为对应的张量，使用目标网络计算下一状态的最大Q值，同时计算出更新Q网络所需的目标Q值，然后使用主Q网络计算当前状态的Q值。



图 14 计算最大Q值和目标Q值

* 通过均方误差（MSE）计算当前Q值与目标Q值的差异，使用反向传播和优化器更新主Q网络的参数。



图 15 更新网络

* 每隔固定步数，将主Q网络的参数同步到目标Q网络，以稳定目标Q值的计算。

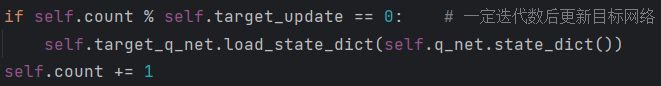


图 16 同步目标Q网络

* 随着训练的进行，逐步减少ε（探索率），从而减少随机动作的概率，更多依赖网络的预测值进行决策。

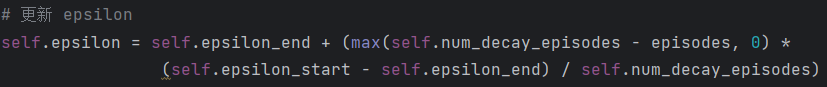


图 17 减少探索率

**3.1.5** 人工产生经验数据

为了让智能体快速学习有效策略并加速训练过程，我还在TetrisAI.py中实现了人工操作生成经验的代码。具体来说，generate\_manual\_experience 函数让用户通过手动操作游戏，并将每次决策的经验数据存储到列表中，在游戏结束时将列表中的经验数据统一添加到experience\_data.json文件。在每次开始训练模型前，可以读取该文件中的经验数据，导入到初始化的经验池中。



图 18 人工产生经验数据



图 19 导入人工经验数据

**3.2** 实验分析  
 在训练了31000轮后，得到了最终训练结果模型参数optimal\_model.pth文件。在Test.py中编写测试代码，导入最终训练的Q网络模型参数，使用智能体游玩1000局俄罗斯方块，结果如下。



图 20 智能体在1000局下的平均结果

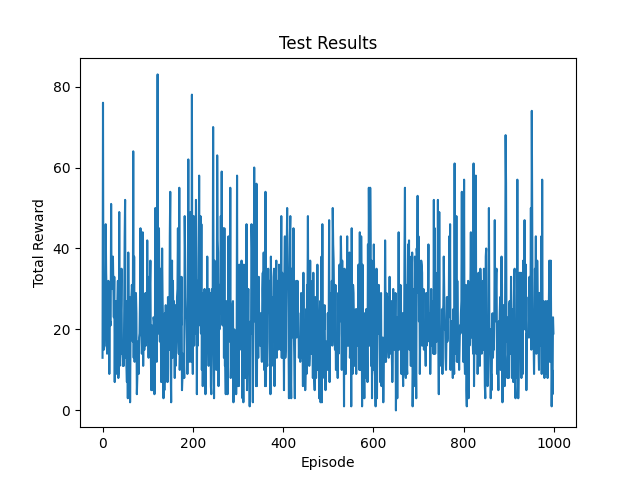


图 21 智能体在1000局中每局的得分

经过1000局训练测试，智能体的表现如下：

* 平均得分为22.57，平均消除行数为21.42，每局平均操作方块数为87.99。这些数据显示基于规则决策和我设计的Q网络架构的DQN算法训练出的AI智能体在游戏中具有一定的稳定性和有效性，能够持续获得分数并消除行数，从而延长游戏时间。
* 智能体偶尔能够获得40分以上的高分，且最高得分突破了80分。这种高分表现很可能是智能体通过惩罚机制在游戏布局上做出了较好的优化，提高整体得分策略的可持续性。

综合来看，该智能体能在俄罗斯方块游戏中取得不错的成绩。

【结论】

在本项目中，我成功实现了一个基于DQN算法的俄罗斯方块游戏AI智能体，并在原始DQN算法的基础上进行了多项优化，包括：

* 根据俄罗斯方块游戏特点设计Q网络模型，使网络更适配游戏状态。
* 添加规则提升得分效率，帮助智能体在特定场景中选择更优动作。
* 引入游戏布局特征和惩罚机制，有效降低游戏提前结束的风险。
* 人工生成优质经验，显著提升模型初期的训练效果。

尽管整个项目代码量不大，仅约五六百行，但将DQN模型部署到俄罗斯方块游戏上耗费了大量精力。改动最多的部分是Q网络模型架构设计。在方法分析部分可以看到，我尝试了三种不同的网络架构，最终证明只有最后一种架构既可行又能带来良好的训练效果。与此同时，还对俄罗斯方块的游戏代码进行了多次调整，以适配网络架构对状态表示的需求。

此外，惩罚机制的设置也是一大挑战。起初在模型训练初期引入惩罚，结果导致模型难以取得理想的训练效果。后来调整为在训练后期加入惩罚，用于微调模型性能。同时，对惩罚权重的配置进行了反复实验，最终才找到一个较为合适的值。

最后，相比于现有的网上案例，我设计的智能体仍有较大提升空间。例如，可以引入更先进的强化学习算法（如Actor-Critic或PPO），以提高学习效率和决策质量；或者设计更多辅助规则，帮助智能体在复杂场景中做出更优决策。

【参考文献】

1. 动手学强化学习第7章DQN算法<https://hrl.boyuai.com/chapter/2/dqn%E7%AE%97%E6%B3%95>
2. 强化学习DQN之俄罗斯方块 <https://blog.csdn.net/qq128252/article/details/129145534>
3. 不到 300 行代码实现俄罗斯方块 <https://www.cnblogs.com/gl1573/p/10213812.html>
4. Playing Atari with Deep Reinforcement Learning <https://arxiv.org/abs/1312.5602>