智能体（Agent）设计与实现详解

CS181-Project

June 2, 2025

目录

智能体体系结构

- **Player**：抽象基类，定义所有玩家的接口
- **HumanPlayer**：人类玩家，动作由外部输入
- **RandomPlayer**：随机选择动作，作为基线
- **GreedyPlayer**：贪心选择当前最优动作
- **MinimaxPlayer**：极大极小搜索，考虑对手反应
- **MCTSPlayer**：蒙特卡洛树搜索，平衡探索与利用
- **ApproximateQLearningPlayer**：特征线性Q学习
- **Neural_ApproximateQLearningPlayer**：深度Q学习（神经网络）

评估函数 evaluation

用于**Greedy/Minimax**智能体，衡量棋盘状态优劣。

- 计算每个己方棋子到目标区所有空位的最大欧氏距离
- 所有棋子都到达目标区时有惩罚（-20）
- 返回值乘以-1，使得距离越小分数越高

$$\text{val} = -\sum_{p \in P_{\text{self}}} \begin{cases} \max\limits_{g \in G_{\text{empty}}} \sqrt{(x_p - x_g)^2 + (y_p - y_g)^2}, & \text{if } G_{\text{empty}} \neq \emptyset \\ -20, & \text{if } G_{\text{empty}} = \emptyset \end{cases}$$

$$\text{dist}(p, g) = \sqrt{(x_p - x_g)^2 + (y_p - y_g)^2}$$

评估函数 evaluation_MCTS

用于**MCTS**和强化学习智能体，综合多项指标。

- 目标区棋子数量奖励（40%）
- 距离评分（30%）
- 阶段性奖励（递进式）
- 家中棋子惩罚

$$
\begin{aligned}
\text{goal\_score} &= 0.4 \times 1000 \times \frac{\text{pieces\_in\_goal}}{4} \\
\text{distance\_score} &= 0.3 \times 300 \times \left(1 - \frac{\text{normalized\_dist\_sum}}{N}\right) \\
\text{stage\_bonus} &= \text{分阶段奖励} \\
\text{home\_penalty} &= -100 \times \frac{\text{pieces\_at\_home}}{N} \\
\text{final\_score} &= \text{goal\_score} + \text{distance\_score} + \text{stage\_bonus} + \text{home\_penalty}
\end{aligned}
$$

RandomPlayer & GreedyPlayer

RandomPlayer

- 完全随机选择动作
- 用于基线对比

GreedyPlayer

- 遍历所有动作，模拟后用evaluation评估
- 选择分数最高的动作
- 只考虑当前一步

MinimaxPlayer（极大极小搜索）

- 递归搜索到指定深度或超时
- 己方回合最大化分数，对手回合最小化分数
- 使用Alpha-Beta剪枝优化搜索效率
- 叶节点用evaluation评估
- 支持**Local Search**（局部搜索）技术，进一步提升效率

$$\text{Minimax}(s, d) = \begin{cases} \max_{a \in A(s)} \text{Minimax}(s', d-1), & \text{if maximizer} \\ \min_{a \in A(s)} \text{Minimax}(s', d-1), & \text{if minimizer} \end{cases}$$

Local Search（局部搜索）是一种在博弈树搜索中减少分支、提升效率的启发式方法。

- 标准Minimax每层枚举所有己方棋子的所有动作，分支数极大。
- Local Search只为每个己方棋子挑选最优（或前k优）动作，大幅减少分支。
- 实现方式：对每个己方棋子，模拟所有可行动作，用evaluation函数评估，仅保留分数最高的动作。
- 这样能让搜索更深，提升决策速度，同时保证整体推进。

伪代码：

$$A_{\text{local}}(s) = \bigcup_{p \in P_{\text{self}}} \left\{ \arg\max_{a \in A_p(s)} \text{evaluation}(s, a) \right\}$$

其中$A_p(s)$为棋子p所有可行动作。

Local Search的优缺点

优点：

- 极大减少分支数，提升搜索深度和效率
- 保证每个己方棋子都被推进，避免只关注部分棋子
- 实现简单，易于集成到Minimax框架

缺点：

- 可能丢失全局最优解（只考虑局部最优）
- 对evaluation函数的准确性依赖较强

MCTSPlayer（蒙特卡洛树搜索）

四阶段流程

1. **Selection**：用UCB选择最优子节点
2. **Expansion**：扩展新子节点，优先朝目标方向
3. **Simulation**：90%概率用启发式，10%随机，最多30步
4. **Backpropagation**：回传模拟结果，更新访问次数和累计价值

MCTS的UCB公式

UCB（Upper Confidence Bound）公式：

$$UCB = \frac{Q}{N} + c\sqrt{\frac{\ln N_{parent}}{N}} + \text{strategy\_score}$$

- Q：累计价值
- N：访问次数
- c：探索系数（前期大，后期小）
- strategy_score：方向性、跳跃奖励等

MCTS动作优先级与模拟

- 优先减少到目标距离的动作
- 奖励跳跃，惩罚后退
- 模拟阶段90%概率选择朝目标方向动作
- 最终动作选择综合胜率、访问率、方向性

$$\text{score} = 0.4 \times \text{win\_ratio} + 0.2 \times \text{visit\_ratio} + 0.4 \times \text{direction\_score}$$

MinimaxPlayer (Minimax Search)

- Recursively searches to a specified depth or until timeout.
- Maximizes the score on the player's turn, minimizes on the opponent's turn.
- Uses Alpha-Beta pruning to improve search efficiency.
- Uses the evaluation function at leaf nodes.
- Supports **Local Search** to further improve efficiency.

$$\text{Minimax}(s, d) = \begin{cases} \max_{a \in A(s)} \text{Minimax}(s', d-1), & \text{if maximizer} \\ \min_{a \in A(s)} \text{Minimax}(s', d-1), & \text{if minimizer} \end{cases}$$

Local Search in Minimax

Local Search is a heuristic method to reduce the branching factor and improve efficiency in game tree search.

- Standard Minimax enumerates all possible moves for all pawns at each layer, resulting in a huge branching factor.
- Local Search selects only the best (or top-k) move(s) for each pawn, greatly reducing the branching factor.
- Implementation: For each pawn, simulate all possible moves, evaluate them using the evaluation function, and keep only the move with the highest score.
- This allows deeper search and faster decision-making, while ensuring all pawns are advanced.

Pseudocode:

$$A_{\text{local}}(s) = \bigcup_{p \in P_{\text{self}}} \left\{ \arg \max_{a \in A_p(s)} \text{evaluation}(s, a) \right\}$$

where $A_p(s)$ is the set of all possible moves for pawn p.

Advantages and Disadvantages of Local Search

Advantages:

- Greatly reduces the branching factor, allowing deeper and more efficient search.
- Ensures all pawns are advanced, avoiding focus on only a few pawns.
- Simple to implement and easy to integrate into the Minimax framework.

Disadvantages:

- May miss the global optimum (only considers local optima).
- Relies heavily on the accuracy of the evaluation function.

ApproximateQLearningPlayer（特征Q学习）

特征设计

- 状态特征：目标区棋子比例、平均距离
- 动作特征：距离改善、方向性、跳跃、到达目标、后退、离家

$$Q(s, a) = \sum_i w_i \cdot f_i(s, a)$$

权重示例：

- pieces_in_goal: 2000
- avg_distance: -800
- distance_improvement: 500
- direction: 300
- is_jump: 100
- reaches_goal: 1500
- is_backwards: -1000
- leaves_home: 200

Q值更新与奖励设计

TD(0)更新公式：

$$w_i \leftarrow w_i + \alpha \cdot (r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \cdot f_i(s, a)$$

奖励设计

- 胜利奖励：$3000 + 500 \times \text{pieces\_in\_goal}$
- 目标进展奖励：$300 \times 2^{\text{当前目标棋子数}}$
- 距离改善奖励：前期100，后期200
- 跳跃奖励：前期200，中期50，后期0
- 后退惩罚：前期-200，后期-500

ApproximateQLearningPlayer: Feature Overview

Feature-based Q-Learning:

- Uses a linear combination of hand-crafted features to estimate $Q(s, a)$.
- Features are divided into state features and action features.

$$Q(s, a) = \sum_i w_i \cdot f_i(s, a)$$

Neural_ApproximateQLearningPlayer 总结

核心思想: 使用深度神经网络 (DNN) 来近似Q值函数 $Q(s, a)$，结合经验回放和目标网络进行训练。

网络结构 (NeuralFeatureExtractor):

- **输入:**
  - 棋盘状态 (Board State): $4 \times N \times N$ 张量 (己方棋子, 对方棋子, 目标区, 初始区)
- 动作 (Action): 4维向量 (start_x, start_y, end_x, end_y)
- **棋盘编码:** 3层卷积网络 (Conv2d) $\rightarrow$ Flatten
- **动作编码:** 1层全连接网络 (Linear)
- **特征融合:** 拼接棋盘和动作特征 $\rightarrow$ 3层全连接网络 (Linear)
- **输出:** 单个Q值 $Q(s, a)$

Q值函数:

$$Q(s, a; \theta) = \text{DNN}(\text{encode\_board}(s), \text{encode\_action}(a); \theta)$$

其中 θ 是神经网络的参数。

训练与更新 (DQN思想):

State Features (Examples)

State features:

$$\text{pieces\_in\_goal} = \frac{\text{Number of player's pieces in goal area}}{4}$$

$$\text{avg\_distance} = \frac{1}{4D_{\text{max}}} \sum_{p \notin G} |x_p - x_c| + |y_p - y_c|$$

- G: goal area, (x_c, y_c): goal center, D_{max}: max possible distance

Action Features (Examples)

Action features:

$$\text{distance\_improvement} = \frac{d_{\text{start}} - d_{\text{end}}}{D_{\text{max}}}$$

$$\text{direction} = \begin{cases} \frac{(x_{\text{end}} - x_{\text{start}}) + (y_{\text{end}} - y_{\text{start}})}{2B}, & \text{if RED} \\ \frac{(x_{\text{start}} - x_{\text{end}}) + (y_{\text{start}} - y_{\text{end}})}{2B}, & \text{otherwise} \end{cases}$$

$$\text{is\_jump} = \begin{cases} 1, & \text{if jump move} \\ 0, & \text{otherwise} \end{cases}$$

$$\text{reaches\_goal} = \begin{cases} 1, & (x_{\text{end}}, y_{\text{end}}) \in G \\ 0, & \text{otherwise} \end{cases}$$

More Action Features

Additional action features:

$$\text{is\_backwards} = \begin{cases} 1, & \text{if move is backwards} \\ 0, & \text{otherwise} \end{cases}$$

$$\text{leaves\_home} = \begin{cases} 1, & (x_{\text{start}}, y_{\text{start}}) \in H \text{ and } (x_{\text{end}}, y_{\text{end}}) \notin H \\ 0, & \text{otherwise} \end{cases}$$

- H: home area

Example Feature Weights

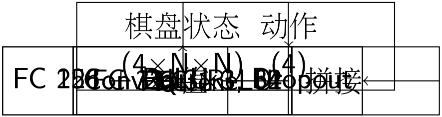
Example weights:

- pieces_in_goal: 2000
- avg_distance: -800
- distance_improvement: 500
- direction: 300
- is_jump: 100
- reaches_goal: 1500
- is_backwards: -1000
- leaves_home: 200

$$Q(s,a) = \sum_{i=1}^{n} w_i \cdot f_i(s,a)$$

$$\begin{aligned} Q(s,a) = {} & w_1 \cdot \text{pieces\_in\_goal} \\ & + w_2 \cdot \text{avg\_distance} \\ & + w_3 \cdot \text{distance\_improvement} \\ & + w_4 \cdot \text{direction} \\ & + w_5 \cdot \text{is\_jump} \\ & + w_6 \cdot \text{reaches\_goal} \\ & + w_7 \cdot \text{is\_backwards} \\ & + w_8 \cdot \text{leaves\_home} \end{aligned}$$

信息流结构图



训练细节

- 经验回放（Replay Buffer）： deque，容量10000
- 每步采样32个样本进行小批量训练
- 目标网络每1000步同步一次
- 损失函数：均方误差（MSE）
- 优化器：Adam
- ϵ-greedy策略动态调整探索率
- 过滤掉反向动作，避免无效循环
- 终局阶段（仅剩2个未进目标区棋子）采用Minimax逻辑

Q值与更新公式

$$Q(s,a) = \text{network}(s,a)$$

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}\left(Q(s_i,a_i) - \left[r_i + \gamma \max_{a'} Q_{\text{target}}(s_i', a')\right]\right)^2$$

- Q_{target}：目标网络
- r_i：即时奖励，γ：折扣因子

实现细节

- 支持GPU加速（自动检测CUDA）
- 支持模型保存与加载
- 训练时动态调整ϵ，后期更倾向利用
- 经验回放提升样本利用率，目标网络提升训练稳定性
- 支持自我对弈训练和与Minimax等对手对弈

智能体（Agent）设计与实现详解

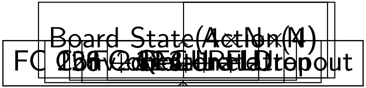
CS181-Project

June 2, 2025

Network Architecture

- **Input:** Board state (4 channels, $N \times N$) and action (4-dimensional vector)
- **Board encoder:** 3 convolutional layers (Conv2d)
- **Action encoder:** 1 fully connected (FC) layer
- **Fusion:** 3 FC layers with ReLU and Dropout
- **Output:** Q-value

Information Flow Diagram



Training Details

- Experience replay buffer (deque, size 10000)
- Mini-batch update (batch size 32)
- Target network updated every 1000 steps
- Loss: Mean Squared Error (MSE)
- Optimizer: Adam
- ϵ-greedy policy with dynamic exploration rate
- Filters out reverse actions to avoid oscillation
- Uses Minimax logic for endgame (when only 2 pieces not in goal)

Q-value and Update Formula

$$Q(s, a) = \text{network}(s, a)$$

$$\text{Loss} = \frac{1}{N} \sum_{i=1}^{N} \left(Q(s_i, a_i) - \left[r_i + \gamma \max_{a'} Q_{\text{target}}(s_i', a') \right] \right)^2$$

- Q_{target}: target network
- r_i: immediate reward, γ: discount factor

Implementation Details

- Supports GPU acceleration (auto-detects CUDA)
- Supports model saving and loading
- Dynamically adjusts ϵ during training (more exploitation in late stage)
- Experience replay improves sample efficiency; target network stabilizes training
- Supports self-play and training against Minimax or other agents

智能体对比总结

类型	策略核心	优点	缺点
Random	随机	实现极简	无智能
Greedy	贪心	推进快，简单	只看一步
Minimax	博弈树	考虑对手	分支爆炸
MCTS	蒙特卡洛树	探索与利用平衡	计算量大
AQL	特征Q学习	可解释性强	特征有限
NAQL	深度Q学习	泛化强	训练慢