西南交通大學

《算法分析与设计》

课程实践报告

报告题目：基于多种智能方法的的斗地主 AI 算法

学院名称：信息科学与技术学院

年　　级：2018 级

授课教师：杜圣东

辅导教师：

学号	姓名	分工	成绩
2018112664	江胤佐	算法设计，代码实现	
2018110610	吴玮	算法分析，代码修改	
2018115053	孙硕人	算法分析，代码修改	

二〇二〇 年 六 月

目录

一. 选题动机

1.1 为何选择机器学习

机器学习是研究怎样使用计算机模拟或实现人类学习活动的科学，是人工智能中最具智能特征，最前沿的研究领域之一，同样也是学习人工智能的基础与核心。通过对机器学习、人工智能相关方面的研究进行调研后，我们发现机器学习的一个重要算法——蒙特卡洛树搜索算法被广泛地应用在各种博弈对抗类游戏的 AI 上，从比较知名且复杂的 AlphaGo 到一些较为简单的五子棋，象棋等棋类都采用了该算法，训练出的 AI 甚至比人类更具博弈性和竞争力。

1.2 为何选择斗地主

斗地主是最少由 3 个玩家进行，用一副 54 张的扑克（包括王牌），其中一方为地主，其余两家为另一方，双方对战，先出完牌的一方获胜的卡牌游戏。之所以选择斗地主作为本次研究的对象，是因为与上文所提到的那些棋类博弈对抗游戏相比，斗地主是一种不完全信息博弈，由于是 2 对 1 的不公平对抗，而且双方手牌都是随机发放，这使得斗地主的对局更为不确定，这就需要牌手具有对当前牌局有深刻理解，对己方手牌，敌方手牌有明确的推断，所以对于传统的基于策略的算法而言，复杂度将会非常高，甚至难以实现，而这种高复杂度的场景正好适用蒙特卡洛算法这种近似推断方法。

我们小组希望通过对蒙特卡洛树搜索算法的调研，查阅资料，对其进行优化、调整，最终设计一种能够训练斗地主 AI 的算法，以更深入地了解机器学习算法的实际应用。

二. 相关研究

2.1 基于蒙特卡洛树搜索的相关研究

1. 围棋人工智能 AlphaGo

阿尔法围棋（AlphaGo）是第一个击败人类职业围棋选手、第一个战胜围棋世界冠军的人工智能机器人。阿尔法围棋用到了很多新技术，如神经网络、深度学习、蒙特卡洛树搜索法等。阿尔法围棋系统主要由几个部分组成：策略网络（Policy Network），给定当前局面，预测并采样下一步的走棋；快速走子（Fast rollout），目标和策略网络一样，但在适当牺牲走棋质量的条件下，速度要比策略网络快 1000 倍；价值网络（Value Network），给定当前局面，估计是白胜概率大还是黑胜概率大；蒙特卡洛树搜索（Monte Carlo Tree Search），把以上这四个部分连起来，形成一个完整的系统。

2. CGF 战术决策

计算机生成兵力(Computer Generated Forces, CGF)的行为建模一直是作战仿真技术中的重点和难点。在一些典型的 CGF 系统开发实践中,行为模型需要大量地依赖领域相关人员参与构建,以形成足够的领域知识驱动复杂行为。但是这些模型主要产生反应式行为,除了构建过程繁琐之外,也无法对人的高级认知活动(如态势感知、任务规划和推理决策等)进行有效表示。将蒙特卡洛树搜索应用于 CGF 战术决策能够使 CGF 在虚拟战场中表现出更具智能性的行为。同时,其在推理过程中考虑了敌我双方的可能行动,在此基础上形成对不同方案的评估,这种方式非常适合分析类似作战这种存在动态对抗的环境。

2.2 支持向量机的相关背景

1. 文本信息监测

基于集成学习策略 SVM 分类法对中文文本中具有欺骗性的信息进行检测识别. 通过改进的二分 k-均值划分法划分实验中的训练样本, 进而对各个训练样本子集构建相应的分类器, 称之为子分类器. 最终通过集成所有子 SVM 分类器的结果确定实验的分类效果, 研究人员对其良好的识别率进行分析总结。

2. 手写文档分类

通过结构化学习技术确定 SVM 参数, 并构造松弛结构 SVM 训练估计最优的参数. 结构化 SVM 方法降低识别的计算复杂度并解决了手写文档不规则和多样化的干扰因素. 该文仅在拉丁语和印度语文档中证明其可靠性, 可通过将该方法应用于英文以及汉语文本中, 扩大其文本应用范围。

2.3 Q-Learning 的相关背景

由 DeepMind 公司开发的利用深度卷积神经网络来进行的 Q-Learning 的算法。在使用非线性函数逼近的时候，强化学习经常会有不稳定性或者发散性：这种不稳定性来于当前的观测中有比较强的自相关。DeepMind 通过使用经历回放，也就是每次学习的时候并不直接从最近的经历中学习，而是从之前的经历中随机采样来进行训练。

三. 算法分析

3.1 支持向量机

支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM 的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等

价于正则化的合页损失函数的最小化问题。SVM 学习算法就是求解凸二次规划的最优化算法。

SVM 学习的基本想法是求解能够正确划分训练数据集并且几何间隔最大的分离超平面。如下图所示，$\omega \cdot x + b = 0$即为分离超平面，对于线性可分的数据集来说，这样的超平面有无穷多个（即感知机），但是几何间隔最大的分离超平面却是唯一的。

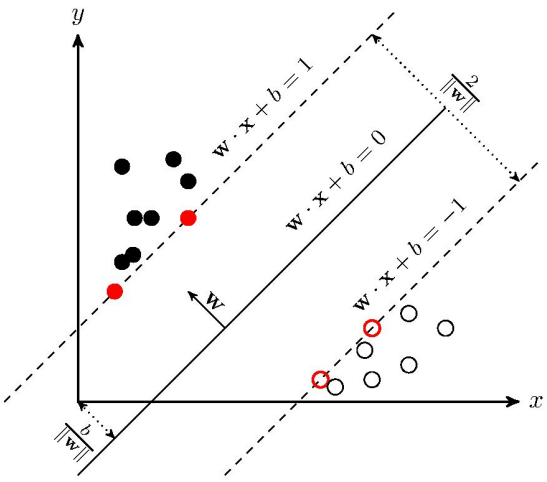


图 3.1（1） 几何间隔最大的分离超平面

具体的算法原理说明由于过于复杂在此按下不表。

直接给出线性支持向量机学习算法：

输入：训练数据集$T = (x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$其中，$x_i \in \mathbb{R}^n, y_i \in +1, -1, i = 1,2, \dots n$

输出：分离超平面和分类决策函数

（1） 选择惩罚参数

选择惩罚参数$C > 0$，构造并求解凸二次规划问题

$$\min_{\alpha} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{N}\alpha_i$$

$$s.t. \quad \sum_{i=1}^{N}\alpha_i y_i = 0 \quad 其中 \ 0 \le q_i \le C, i = 1,2, \dots, n$$

得到最优解$\alpha^* = (a_1^*, a_2^*, \dots, a_N^*)^T$

（2） 计算

$$\omega^* = \sum_{i=1}^{N}\alpha_i^* y_i x_i$$

选择α^*的一个分量α_j^*满足条件$0 < \alpha_j^* < C$,计算

$$b^* = y_j - \sum_{i=1}^{N}\alpha_i^* y_j (x_i \cdot x_j)$$

（3） 求分离超平面

$$\omega^* \cdot x + b^* = 0$$

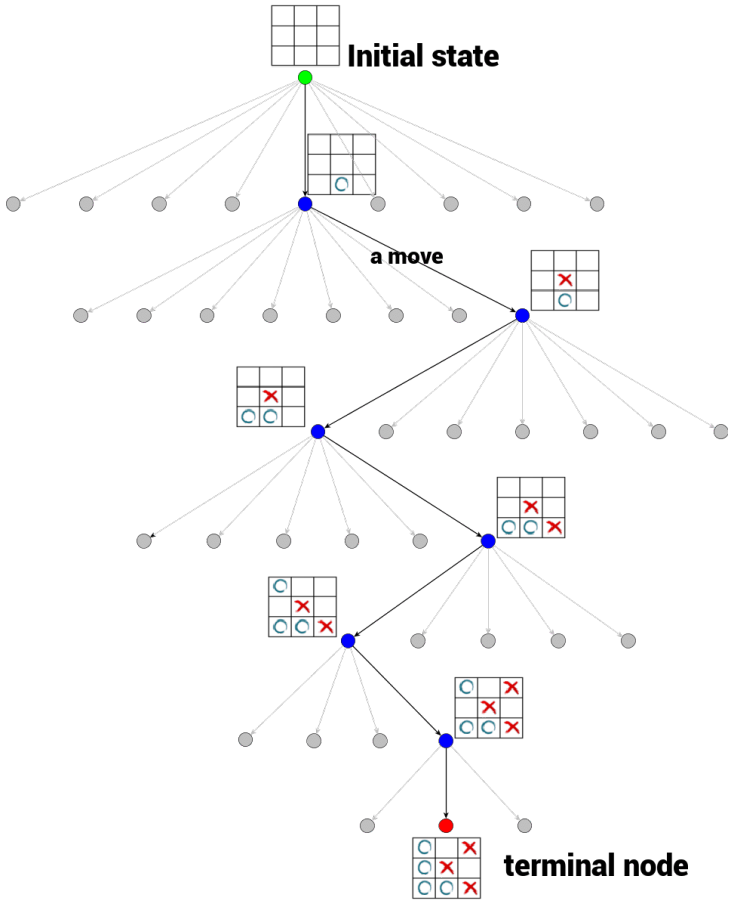
分类决策函数：

$$f(x) = sign(\omega^* \cdot x + b^*)$$

3.2 蒙特卡洛树搜索

如图 3.2(1)我们采用博弈树(Game Tree)来表示一局游戏，定义如下：

1. 每个结点(Node)都代表一个状态(State)
2. 从一个结点移动一步，将会到达它的子节点(Children Node)
3. 子节点的个数叫做分支因子(Branching Factor)
4. 根节点(Root Node)表示初始状态(Initial State)
5. 终止节点(terminal nodes)则没有子节点。



3.2(1) 一局游戏的表示

每次都是从初始状态、树的根结点开始。在 tic-tac-toe 游戏里面初始状态就是一张空的棋盘。从一个节点转移到另一个节点叫作一个 move。分支因子(Branching Factor)，tic-tac-toe 中树越深，分支因子也越少，也就是子节点(Children Node)的数量越少。游戏结束表示终止节点。从根节点到终止节点一次表示一局单个游戏。

我们希望找到的就是最佳策略(The Most Promising Next Move)。如果你知道对手的策略那你可以针对这个策略求解，但是大多数情况下是不知道对手的策略的，所以我们需要用 Minimax 的方法，假设你的对手是非常机智的，每次他都会

采取最佳策略。

假设 A 与 B 博弈，A 期望最大化自己的收益，因为是零和博弈，所以 B 期望 A 的收益最小，Minimax 算法可描述为如下形式：

$$v_A(S_i) = max_{a_i} v_B\big(move(s_i, a_i)\big) \qquad v_A(\hat{s}) = eval(\hat{s})$$

$$v_B(S_i) = max_{a_i} v_A\big(move(s_i, a_i)\big) \qquad v_B(\hat{s}) = -eval(\hat{s})$$

v_A和v_B是玩家 A 和 B 的效益函数。

$move$表示从当前状态S_i和采取的动作a_i转移到下一个状态。

$eval$评估最终的游戏分数。

$\hat{s}$是最终的游戏状态。

简单地说，就是给定一个状态期望找到一个动作在对手最小化你的奖励的同时找到一个最大化自己的奖励。

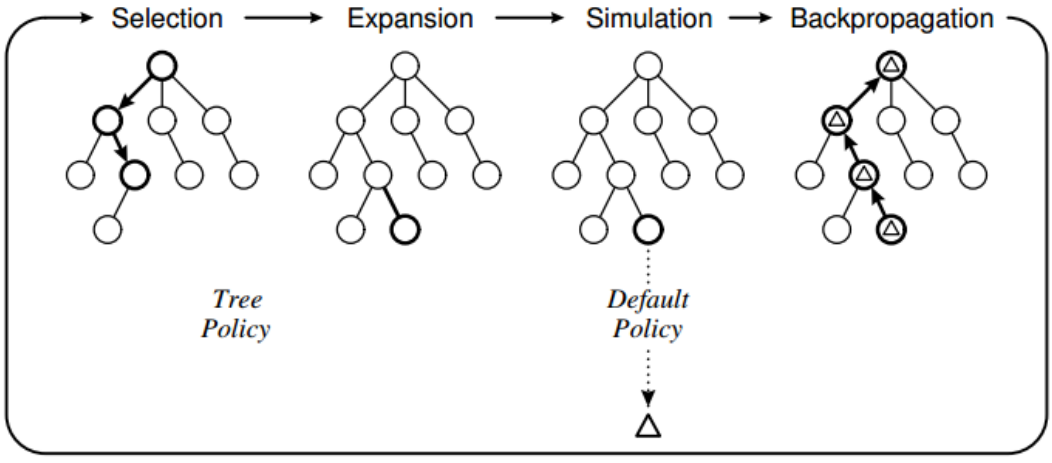
Minimax 算法最大的弱点就是需要扩展整棵树，对于高分支因子的游戏，像围棋、象棋这种，算法就很难处理。

对于上述问题的一种解决方法就是扩展树结构到一定的阈值深度(Threshold Depth)。因此我们需要一个评估函数，评估每一个非终止节点。

另一种解决树扩展太大的方法就是 Alpha-Beta 剪枝算法。它会避免一些分支的展开，它最好的结果就是与 Minimax 算法效果相同，因为它减少了搜索空间。

蒙特卡洛通过多次模拟仿真，预测出最佳策略。最核心的东西就是搜索。搜索是对整棵博弈树的组合遍历，单次的遍历是从根结点开始，到一个未完全展开的节点(Not Full Expanded)。未完全展开的意思就是它至少有一个孩子节点未被访问，或者称作未被探索过。当遇到未被完全展开过的节点，选择它的一个未被访问的子节点作为根结点，进行一次模拟。仿真的结果反向传播(Propagated Back)用于更新当前树的根结点,并更新博弈树节点的统计信息。当整棵博弈树搜索结束后，就相当于拿到了这颗博弈树的策略。

基本的 MCTS 算法非常简单：根据模拟的输出结果，按照节点构造搜索树。其过程可以分为下面的若干步：



1. 选择(Selection)：从根节点 R 开始，递归选择最优的子节点（后面会解释）直到达到叶子节点 L。
2. 扩展(Expansion)：如果 L 不是一个终止节点（也就是，不会导致博弈游戏终止）那么就创建一个或者更多的字子节点，选择其中一个 C。

3. 模拟(Simulation)：从 C 开始运行一个模拟的输出，直到博弈游戏结束。
4. 反向传播(Backpropagation)：用模拟的结果输出更新当前行动序列。

3.3 Q-Learning 算法

Q-Learning 是强化学习算法中 value-based 的算法，Q即为$Q(s,a)$就是在某一时刻的 s 状态下$(s \in S)$，采取动作$a(a \in A)$，动作能够获得收益的期望，环境会根据智能体$agent$的动作反馈相应的回报r，所以算法的主要思想就是将$State$与$Action$构建成一张$Q-table$来存储Q值，然后根据Q值来选取能够获得最大的收益的动作。

核心思想：

1. 智能体 $Agent$ 在状态 s_t，采取行为 a_t；
2. 后移至下一个状态 s_{t+1}；
3. 估计价值 $Q(s_{t+1})$;
4. 更新$Q-table$；
5. 反复 1-4 过程，更新$Q-table$。

更新公式(Bellman Equation)：

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma * max_{a'}Q(s',a') - Q(s,a)]$$

其中：

α为学习率：学习速率越大，保留之前训练的效果就越少；学习率=0，不考虑之前的训练结果。

γ为折扣因子：折扣因子越大，更多地考虑远期收益；折扣因子越小，更多地考虑眼前的收益。

此公式即根据下一个状态s'中选取最大的$Q(s',a')$值乘以折扣因子γ加上真实回报值最大的为Q现实，而根据过往Q表里面的$Q(s,a)$作为Q估计

算法描述：

- Initialize the memory D
- Initialize the action-value network Q with random weights
- **For** episode = 1, M **do**
  - **For** t, T **do**
    - With probability ϵ select a random action a_t, otherwise select $a_t = \text{argmax}_a Q(s,a)$
    - Execute action a_t in simulator and observe reward r_{t+1} and new state s_{t+1}
    - Store transition $< s_t, a_t, r_{t+1}, s_{t+1} >$ in memory D
    - Sample random mini-batch from D: $< s_j, a_j, r_j, s'_j >$
    - Set $\hat{Q}_j = r_j$ if the episode ends at $j+1$, otherwise set $\hat{Q}_j = r_j + \gamma \max_{a'} Q(s'_j, a')$
    - Make a gradient descent step with loss $(\hat{Q}_j - Q(s_j, a_i))^2$
  - **endfor**
- **endfor**

图 3.3(1) Q-Learning 算法描述

四. 设计概述

4.1 规则设计

想要让计算机理解什么是斗地主，我们首先要为其构建斗地主的规则，包括：

1. 牌型规则：明确所有的牌型组合
2. 出牌规则：明确不同牌型间的大小比较与炸弹王炸等特殊换手机制
3. 角色规则：明确地主与农民两个角色的差异
4. 胜利规则：明确胜利的条件为出光手牌

4.2 算法设计

斗地主的游戏过程主要由三个主要过程：出牌、跟牌、其他（包括规则定义、叫地主、出牌大小判断、出牌合法性判断、发牌等等）。

由于使用手牌作为状态，可能性过多，导致算法可行性较差，所有需要另外进行动作与状态的定义。

叫地主部分使用支持向量机学习实现。

牌型（出牌合法性）判断与出牌大小判断定义的判断算法实现

在出牌跟牌部分（即涉及到博弈的部分）需要使用蒙特卡洛树搜索算法。

在模型训练时，需要使用强化学习算法使 AI 自我对战。

五. 实现步骤

5.1 规则设计实现

1. 牌型规则

规则定义：根据斗地主规则定义了各种牌型的大小（每种牌型对应一个不同大小的值，对于同一牌型，值越大表示该牌越大）。

2. 出牌规则

出牌大小判断：根据牌型规则集不同牌型所代表值对当前出牌及上家出牌大小进行比较，如果大于则合法。

出牌合法性判断：判断所出手牌是否为提前定义的牌型规则集的子集。比如：单牌顺子一次性不能少于 5 张且不能带 2，出“3456”、“JQKA2”都会判定为不合法。

```
def _three(di, value) -> int:
    # 飞机 或3带1 或3带2
    if _is_consequent(di[3], 1):
        if not di[1]:
            # 无翼
            if not di[2]:
                return len(di[3]) * 1000 + 300 + value

            # 大翼 或3带2
            if len(di[2]) == len(di[3]):
                return 20000 + len(di[3]) * 1000 + 300 + value

        # 小翼 或3带1
        if len(di[1]) + len(di[2]) * 2 == len(di[3]):
            return 10000 + len(di[3]) * 1000 + 300 + value
    return INVALID_BIT
```

5.1(1)牌型定义、出牌大小及合法性判断代码示例

3. 角色规则

首先将一副牌随机打乱，之后针对不同角色进行发牌，采用在一整副扑克牌中对三个玩家每个玩家随机发 17 张牌的方式，当有任意玩家叫地主时，则将剩下的三张牌发给叫地主的玩家。

```
def shuffle(self) -> None:
    """
    洗牌
    """
    self.cards = np.concatenate(self.cards, axis=0)
    np.random.shuffle(self.cards)
    self.cards = np.split(self.cards, [17, 34, 51])
    for c in self.cards:
        c.sort()

# 卡牌二维数组，前3个代表玩家0、1、2的初始手牌（各17张）最后一项代表3张地主牌
self.cards: List[np.ndarray] = np.split(np.asarray([card for card in range(1, 14)] * 4 + [14, 15], dtype=int),
                                        [17, 34, 51])
```

5.1(3)发牌代码示例

4. 胜利规则

判断胜利：当有任何一人的手牌列表为空即判断其所属阵容（地主或农民）获胜。

```
if len(self.cards[self.turn]) == 0:
    if self.land_lord == self.turn:
        print(self.cur_identity + '获胜')
    else:
        farmers = {0, 1, 2}
        farmers.remove(self.land_lord)
        print('农民(玩家{}、玩家{})获胜'.format(farmers.pop(), farmers.pop()))
    return
```

5.1(4)判断胜利代码示例

5.2 基于贪心法的斗地主拆牌策略实现

未经处理的情况下，斗地主的手牌组合、出牌组合和场上其它玩家手牌数的组合情况过多，传统的强化学习算法难以枚举如此多的状态，因此需要对斗地主的状态和动作进行简化。

记玩家当前手牌为 S， 当前手牌下的一种合法出牌动作为a，用 Q(S, a)代表S和a的情况下，a这个动作拆牌的好坏情况，$Q(S，a)$值越大，拆牌效果越好。

$Q(S,a)$定义如下：

$$Q=\begin{cases} length(a)+max(length(a|S-a)+BombCount(S) & length(S-a)>0 \\ MAXQ & length(S-a)<0 \end{cases}$$

其中$length(a)$代表动作 a 有几张牌；$S-a$代表状态S的牌去掉动作a后剩下的牌，即打出 a 后的下一个状态；$BombCount(S)$代表状态S下的炸弹数量；$MAXQ$表示一个很大的常数。

公式的前半部分$length(a)+max(length(a|S-a))$体现了贪心的思想，认为如果本次出牌和本次出牌后下一次能出的牌数量之和越大，那么这一次拆牌越好。

例如 S = ‘345667’，如果令 a=‘66’，那么下一次出牌最多只能出单，$Q(S，a)=3$，而如果令 a='34567'，则下一次出单，$Q(S,a)=6$。

再考虑 S='44445'这种情况，则需要对拆炸弹的行为进行惩罚，因此用$BombCount(S-a)-BombCount(S)$来衡量破坏了多少炸弹。

考虑 S='66677'这种情况，显然此时可以一次性出完所有牌，因此计作最大的Q值。

由于 2 和大小王由于无法加入连对，故单独计算。在实际处理中，我们还需要将手牌 S 分成若干连续的状态，例如将'34567 JJQQ' 分成'34567'和'JJQQ'两部分。因为拆牌的好坏只会影响到和它连续的牌。

5.3 动作与状态定义

1. 出牌动作

AI 出牌时的动作被化简为表 5.2(1)：（其中的大小是针对符合规则的牌型值的序列而言的）

表 5.3(1) 出牌动作定义表

值	动作
0-4	强行出最小、小、中、大、强行最大的单牌
5-7	小、中、大的对子
8-9	出三，表示出最小、较大的三只
10-12	出炸弹，表示出小炸弹或大炸弹或王炸
13-14	出长度为 5 的顺子，表示出较小的或较大的顺子
15	出其它连对顺子飞机
16	出 4 带 2

2. 出牌状态

将 AI 当前出牌状态转换为一个长度为 12 的特征向量，如表 5.3(2)：

表 5.3(2) 当前出牌状态定义表

状态说明	属性名	取值范围
f1_min(单张)	solo_min	[0, 3]
f1_max(单张)	solo_max	[0, 3]
f2_min(对子)	pair_min	[0, 2]
f2_max(对子)	pair_max	[0, 2]
三的数量(大于 2 记作 2,含飞机)	trios	[0, 2]
最大单顺（长为 5）	seq_solo_5	[0, 2]
存在其它牌型(不含 4 带 2)	other_seq_count	[0, 1]
炸弹数量（大于 2 记作 2）	bomb_count	[0, 2]
是否有王炸	rocket	[0, 1]
玩家位置	player	[0, 2]
上家手牌数	hand_p(大于 7 都记作 7)	[0, 7]
下家手牌数	hand_n(大于 7 都记作 7)	[0, 7]

3. 跟牌动作

AI 跟牌时的动作被化简为表 5.3(3)：

表 5.3(3) 跟牌动作定义表

状态说明	属性名	取值范围（均为整数）
最佳拆牌的 delta_q	delta_q	[0, 5]
玩家身份	player	[0, 2]
上一个出牌者	last_combo_owner	[0, 2]
上家手牌数-1	hand_p(大于 5 都记作 7)	[0, 7]
下家手牌数-1	hand_n(大于 5 都记作 7)	[0, 7]
上一个牌的数量	last_combo_len	[0, 5]

4. 跟牌状态

将 AI 当前跟牌状态定义为表 5.3(4)：

表 5.3(4) 当前跟牌状态定义表

值	动作
0	空过
1-4	跟最小、较小、较大、最大
5	可能拆坏牌的情况下跟最大
6	小炸弹
7	大炸弹
8	王炸

5.4 SVM 二分类手牌叫地主算法实现

通过研究实际的斗地主高端对局发现，牌手叫地主的意向与当前手牌中的大小王（王炸）情况，除大小王以外的炸弹数量，手牌 2 的数量有着直接的关系。

1. **数据采集与处理**

   数据来自小组成员自行人工标注的 4000 条左右的叫地主记录；数据文件见 dataset/call.csv；数据格式如下：

2	3	4	5	5	5	6	7	8	10	11	12	12	12	13	13	14	1
2	2	4	5	5	5	6	6	7	8	8	10	11	11	12	13	14	0
1	1	2	3	3	3	4	5	7	8	10	11	11	11	11	12	13	0
1	3	4	7	7	8	8	10	10	11	11	11	12	12	12	12	13	0
1	2	2	5	6	7	8	8	9	9	12	12	12	13	13	13	14	1
1	2	2	3	4	5	6	7	7	8	9	9	10	11	11	12	12	0

图 5.4(1) 叫地主数据格式示例图

一条数据有 17 个特征，1 个结果。17 个特征代表初始 17 张手牌大小。最后的结果 1 代表叫地主，0 代表不叫。

2. **训练 SVM 分类器**

   采用 sklearn 算法包，选取 linear 线性核进行训练

3. **计算 SVC 分类器的准确率**

   采用测试集进行验证，计算出分类器的准确率为：0.932，符合预期

4. **绘制图像**

   在坐标范围内以 x，y，z 轴分别表示三个特征，绘制散点图 5.4(2)(3)

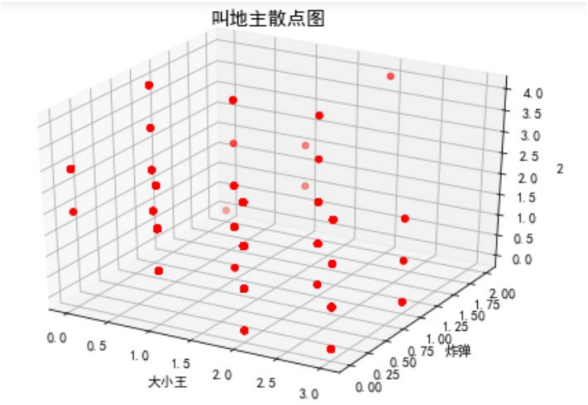


图 5.4(2) 叫地主决策的特征散点图

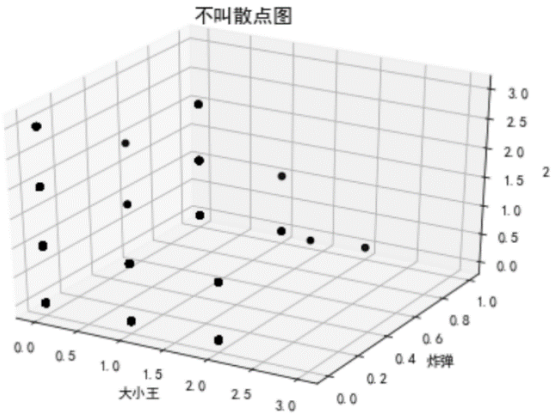


图 5.4(3) 不叫地主决策的特征散点图

5.5 蒙特卡洛树搜索树设计实现

如图 5.5(1)所示，对于本算法而言一局斗地主中的一次出牌的决策流程大致可以分为如下步骤：

1. 更新当前游戏信息
2. 牌型拆分
3. 可选牌型枚举
4. 蒙特卡洛树搜索算法
5. 最佳出牌

如此循环直至游戏结束。其中蒙特卡洛树搜索算法便是其中的核心，利用它才能给出每次的最佳决策，直至获得最终胜利

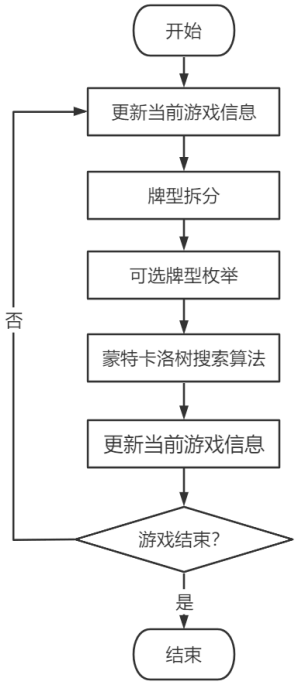


图 5.5(1) 出牌的决策流程

很显然，在“斗地主”游戏中，其他玩家手牌信息对于当前玩家而言是隐藏的，这无疑会大大增加了博弈树中的节点数，导致博弈树空间进一步增大，增加解决“斗地主”游戏的难度.。所以普通的博弈树搜索算法，对于“斗地主”这类不完全信息博弈游戏不能提供很好的解决方法。针对问题规模大的问题，在斗地主游戏中，玩家根据自己手牌、两个玩家已出扑克以及两个玩家手牌张数等信息，通过对游戏未完成的博弈过程进行不断的模拟，以确定在当前情况下玩家最佳的决策(出牌)。

具体决策流程如下:

步骤 1 首先，根据当前玩家的手牌，以及所有玩家已出扑克等信息，基于“斗地主”的较小拆分算法，求解出待出牌集合，再将求解结果中的每个元素增加到博弈树中作为叶子节点.。

步骤 2 选择过程：在博弈树中所有的叶节点中，选取一个叶节点作为本次模拟的根节点。在选取的时候，由于需要权衡探索和利用的关系，所以将博弈树中的叶节点通过 UCT 算法计算该节点的选择评估值 γ，选择评估值 γ 最大的叶节点

作为新一次模拟的根节点进行模拟。UCT 算法为：

$$\gamma_i = \bar{V}_i + C\sqrt{\frac{2ln\sum_j^n n_j}{n_i}}$$

式中γ_i表示节点 i 的选择评估值，$\bar{V}_i$表示节点 i 的平均收益，C是常数,其作用是为了平衡探索和利用。n_i是以第 i 个节点作为模拟搜索的根节点的次数。

步骤 3 扩展过程：在“斗地主”的过程中，由于“斗地主”游戏属于不完全信息博弈的一种，在对其叶节点进行扩展的时候，选中的叶节点的子节点数量较大。因此，在本游戏中，扩展阶段只将本次抽样模拟的根节点加入博弈树中。

步骤 4 模拟过程：从抽样的博弈环境信息开始，3 个玩家根据手牌在满足游戏规则的前提下，随机进行交替出牌，直到到达博弈终止状态。

步骤 5 反馈过程：游戏到达终止状态后，将收益值返回到根节点，并沿途更新节点的统计估计值。实验过程中，在进行蒙特卡洛抽样过程中涉及到随机性问题，但其结果收敛性与实验环境无关，因为根据大数定律可知：当抽样次数足够多时，抽样收益的均值趋近于期望值。在“斗地主”游戏中的决策主体身份分为地主和农民，但博弈过程中，无论智能体身份为地主或农民，进行蒙特卡洛抽样的时候，对于其身份仅仅在最终反馈收益时加以区分，在抽样的过程中，所有决策主体无任何区别.

5.6 Q-Learning 学习算法设计实现

在斗地主中，我们对出牌和跟牌的状态和动作进行区分，出牌和跟牌的动作定义见（5.2 动作与状态定义）。因此，需要在一个训练过程中对出牌和跟牌两张 Q 表进行训练。此外，当玩家 0 打出一个动作后，需要等待另外玩家 1 和玩家 2 做出行动，才能计算出玩家 0 打出这一动作后得到的奖励和状态。

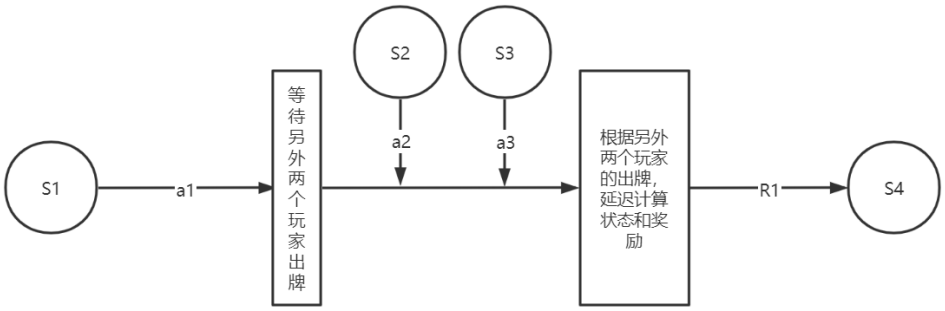


图 5.6(1) 状态动作转移图

具体地，对 Q 表的训练如下所述：

1、根据拆牌结果将 Q 表初始化

2、根据当前的 Q 表的状态给当前 state 选择一个 action 并执行，执行过程一直到本轮训练停止才算完成，同时要采取 epsilon 贪心策略来选择最有动作。epsilon 贪心策略在大多数时候有$1-epsilon$的概率挑选最优动作，也有$epsilon$的概率随机挑选一个状态，从而达到更接近人类水平的目的。

3、采取行动结合规定的奖励后就可以用 Q 函数更新 $Q(s,a)$：

$$Q(S,A) \leftarrow (1-\alpha)Q(S,A) + \alpha[R(S,a) + \gamma \max_a Q(S',a)]$$

其中α和γ均为 0 到 1 之间的数， α越大，智能体越注重过去经验，当$\alpha = 0$时，Q 表不再更新时，$\alpha = 1$时 Q 表将完全抛弃之前学到的经验。；γ取值越大，智能体越注重长远收益，当$\gamma = 0$时，智能体只注重眼下的$Q(s,a)$，$\gamma = 1$时，智能体将注重所有长远的收益。在算法的实现过程中，我们令$\alpha = 0.5$，$\gamma = 0.8$。初始化时$Q(s,a)$均为 0。

六. 开发环境

6.1 硬件环境

硬件环境如表 6.1(1)所示：

表 6.1(1) 硬件环境表

硬件名称	型号
CPU	Intel®Core™ i5-8250U CPU @ 1.6GHz
GPU	NVIDIA® GeForce® MX150
内存	8G @ 2666mHz
硬盘	500G SSD

6.2 软件环境

Windows 10 专业版 2004
PyCharm 2020.1.2

七. 结果与分析

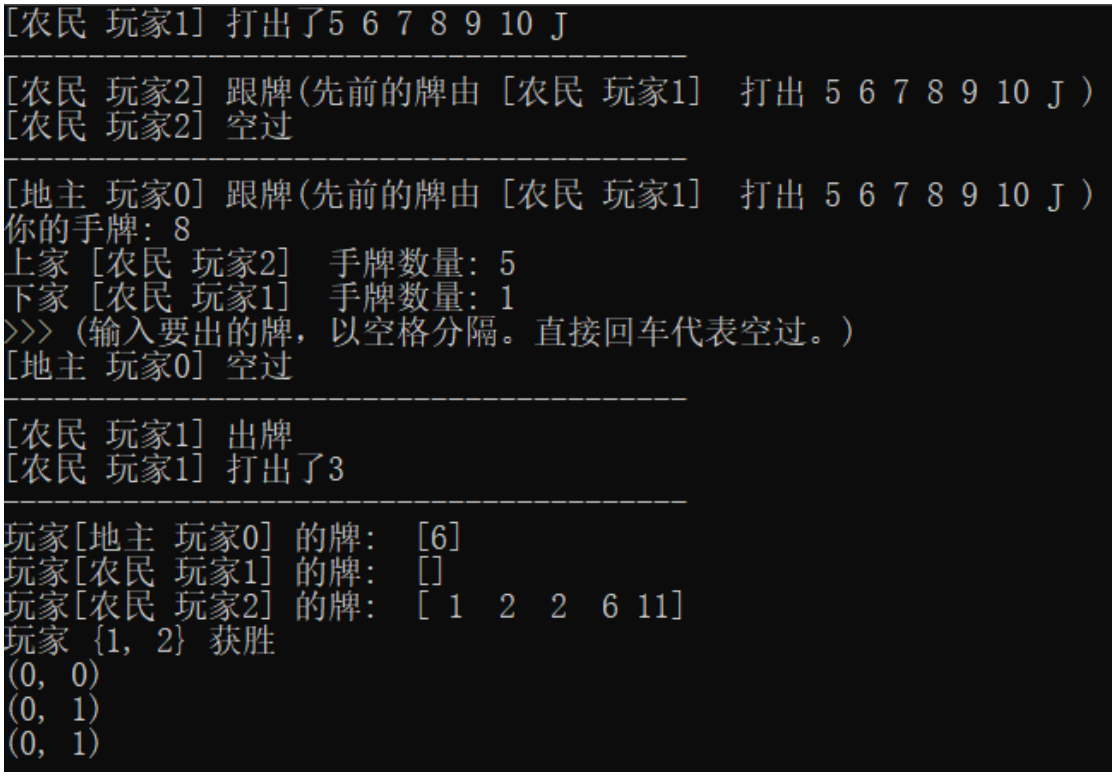
下面演示一局 1 人类 vs 2AI 的人机对战：

```
进入叫地主环节
玩家0的手牌: 3 3 4 5 5 5 7 7 8 8 9 10 J Q Q K A
>>> (输入1叫地主, 输入其它键不叫地主)1
---------------------------------------

玩家0叫了地主
地主获得了3张牌: 4 6 7
---------------------------------------

斗地主开始!
[地主 玩家0] 出牌
你的手牌: 3 3 4 4 5 5 5 6 7 7 7 8 8 9 10 J Q Q K A
上家 [农民 玩家2]  手牌数量: 17
下家 [农民 玩家1]  手牌数量: 17
>>> (输入要出的牌, 以空格分隔。直接回车代表空过。)5 5 5 3 3
[地主 玩家0] 打出了3 3 5 5 5
---------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [地主 玩家0]  打出 3 3 5 5 5 )
[农民 玩家1] 空过
---------------------------------------
[农民 玩家2] 跟牌(先前的牌由 [地主 玩家0]  打出 3 3 5 5 5 )
[农民 玩家2] 空过
---------------------------------------
[地主 玩家0] 出牌
你的手牌: 4 4 6 7 7 7 8 8 9 10 J Q Q K A
上家 [农民 玩家2]  手牌数量: 17
下家 [农民 玩家1]  手牌数量: 17
>>> (输入要出的牌, 以空格分隔。直接回车代表空过。)7 7 7 4 4
[地主 玩家0] 打出了4 4 7 7 7
---------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [地主 玩家0]  打出 4 4 7 7 7 )
[农民 玩家1] 空过
---------------------------------------
[农民 玩家2] 跟牌(先前的牌由 [地主 玩家0]  打出 4 4 7 7 7 )
[农民 玩家2] 空过
---------------------------------------
[地主 玩家0] 出牌
你的手牌: 6 8 8 9 10 J Q Q K A
上家 [农民 玩家2]  手牌数量: 17
下家 [农民 玩家1]  手牌数量: 17
>>> (输入要出的牌, 以空格分隔。直接回车代表空过。)8 9 10 J Q K A
[地主 玩家0] 打出了8 9 10 J Q K A
---------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [地主 玩家0]  打出 8 9 10 J Q K A )
[农民 玩家1] 空过
---------------------------------------
[农民 玩家2] 跟牌(先前的牌由 [地主 玩家0]  打出 8 9 10 J Q K A )
[农民 玩家2] 空过
```

```
----------------------------------------
[地主 玩家0] 出牌
你的手牌: 6 8 Q
上家 [农民 玩家2]  手牌数量: 17
下家 [农民 玩家1]  手牌数量: 17
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)6
[地主 玩家0] 打出了6
----------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [地主 玩家0]  打出 6 )
[农民 玩家1] 打出了B
----------------------------------------
[农民 玩家2] 跟牌(先前的牌由 [农民 玩家1]  打出 B )
[农民 玩家2] 空过
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家1]  打出 B )
你的手牌: 8 Q
上家 [农民 玩家2]  手牌数量: 17
下家 [农民 玩家1]  手牌数量: 16
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)
[地主 玩家0] 空过
----------------------------------------
[农民 玩家1] 出牌
[农民 玩家1] 打出了2 2
----------------------------------------
[农民 玩家2] 跟牌(先前的牌由 [农民 玩家1]  打出 2 2 )
[农民 玩家2] 空过
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家1]  打出 2 2 )
你的手牌: 8 Q
上家 [农民 玩家2]  手牌数量: 17
下家 [农民 玩家1]  手牌数量: 14
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)
[地主 玩家0] 空过
----------------------------------------
[农民 玩家1] 出牌
[农民 玩家1] 打出了K K
----------------------------------------
[农民 玩家2] 跟牌(先前的牌由 [农民 玩家1]  打出 K K )
[农民 玩家2] 打出了2 2
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家2]  打出 2 2 )
你的手牌: 8 Q
上家 [农民 玩家2]  手牌数量: 15
下家 [农民 玩家1]  手牌数量: 12
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)
[地主 玩家0] 空过
----------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [农民 玩家2]  打出 2 2 )
[农民 玩家1] 空过
----------------------------------------
[农民 玩家2] 出牌
```

```
[农民 玩家2] 出牌
[农民 玩家2] 打出了A A
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家2] 打出 A A )
你的手牌: 8 Q
上家 [农民 玩家2] 手牌数量: 13
下家 [农民 玩家1] 手牌数量: 12
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)
[地主 玩家0] 空过
----------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [农民 玩家2] 打出 A A )
[农民 玩家1] 空过
----------------------------------------
[农民 玩家2] 出牌
[农民 玩家2] 打出了9
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家2] 打出 9 )
你的手牌: 8 Q
上家 [农民 玩家2] 手牌数量: 12
下家 [农民 玩家1] 手牌数量: 12
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)Q
[地主 玩家0] 打出了Q
----------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [地主 玩家0] 打出 Q )
[农民 玩家1] 打出了A
----------------------------------------
[农民 玩家2] 跟牌(先前的牌由 [农民 玩家1] 打出 A )
[农民 玩家2] 打出了G
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家2] 打出 G )
你的手牌: 8
上家 [农民 玩家2] 手牌数量: 11
下家 [农民 玩家1] 手牌数量: 11
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)
[地主 玩家0] 空过
----------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [农民 玩家2] 打出 G )
[农民 玩家1] 空过
----------------------------------------
[农民 玩家2] 出牌
[农民 玩家2] 打出了Q Q
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家2] 打出 Q Q )
你的手牌: 8
上家 [农民 玩家2] 手牌数量: 9
下家 [农民 玩家1] 手牌数量: 11
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)
[地主 玩家0] 空过
----------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [农民 玩家2] 打出 Q Q )
[农民 玩家1] 空过
----------------------------------------
```

```
----------------------------------------
[农民 玩家2] 出牌
[农民 玩家2] 打出了J J
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家2]  打出 J J )
你的手牌: 8
上家 [农民 玩家2]  手牌数量: 7
下家 [农民 玩家1]  手牌数量: 11
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)
[地主 玩家0] 空过
----------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [农民 玩家2]  打出 J J )
[农民 玩家1] 空过
----------------------------------------
[农民 玩家2] 出牌
[农民 玩家2] 打出了6 6
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家2]  打出 6 6 )
你的手牌: 8
上家 [农民 玩家2]  手牌数量: 5
下家 [农民 玩家1]  手牌数量: 11
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)
[地主 玩家0] 空过
----------------------------------------
[农民 玩家1] 跟牌(先前的牌由 [农民 玩家2]  打出 6 6 )
[农民 玩家1] 打出了10 10
----------------------------------------
[农民 玩家2] 跟牌(先前的牌由 [农民 玩家1]  打出 10 10 )
[农民 玩家2] 空过
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家1]  打出 10 10 )
你的手牌: 8
上家 [农民 玩家2]  手牌数量: 5
下家 [农民 玩家1]  手牌数量: 9
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)
[地主 玩家0] 空过
----------------------------------------
[农民 玩家1] 出牌
[农民 玩家1] 打出了9
----------------------------------------
[农民 玩家2] 跟牌(先前的牌由 [农民 玩家1]  打出 9 )
[农民 玩家2] 空过
----------------------------------------
[地主 玩家0] 跟牌(先前的牌由 [农民 玩家1]  打出 9 )
你的手牌: 8
上家 [农民 玩家2]  手牌数量: 5
下家 [农民 玩家1]  手牌数量: 8
>>> (输入要出的牌，以空格分隔。直接回车代表空过。)
[地主 玩家0] 空过
----------------------------------------
[农民 玩家1] 出牌
[农民 玩家1] 打出了5 6 7 8 9 10 J
```



最终由 AI 获得了胜利。

下面表格将展示，训练过程中，算法的效率提升：

*由于斗地主的规则复杂，采用单纯的随机出牌并不具有实际意义，故仅测试贪心拆牌+Q-Learning 决策与贪心拆牌+随机决策的性能差距（实际算法性能远大于胜率）：

1. 训练 400 次，约 26 秒

智能体	地主获胜场次	农民获胜场次	总获胜场次	胜率
QLAgent	176	370	546	54.6%
RandomAgent1	148	337	485	48.5%
RandomAgent2	155	335	490	49.3%

2. 训练 1000 次，约 64 秒

智能体	地主获胜场次	农民获胜场次	总获胜场次	胜率
QLAgent	155	395	550	55.5%
RandomAgent1	122	385	507	50.7%
RandomAgent2	143	380	523	52.3%

3. 训练 2000 次，约 128 秒

智能体	地主获胜场次	农民获胜场次	总获胜场次	胜率
QLAgent	141	403	544	54.4%
RandomAgent1	134	383	517	51.7%
RandomAgent2	148	368	516	51.6%

4. 训练 5000 次，约 325 秒

智能体	地主获胜场次	农民获胜场次	总获胜场次	胜率
QLAgent	174	382	566	56.6%
RandomAgent1	123	378	501	50.1%
RandomAgent2	132	382	514	51.4%

5. 训练 10000 次，约 643 秒

智能体	地主获胜场次	农民获胜场次	总获胜场次	胜率
QLAgent	167	392	559	55.9%
RandomAgent1	120	399	519	51.9%
RandomAgent2	127	381	508	50.8%

6. 训练 60000 次，约 3906 秒

智能体	地主获胜场次	农民获胜场次	总获胜场次	胜率
QLAgent	163	407	570	57.0%
RandomAgent1	138	352	490	49.0%
RandomAgent2	147	345	492	49.2%

八. 收获与总结

在本次课程设计中，我们分模块地使用了蒙特卡洛树搜索，支持向量机，Q-Learning 算法构建了一个能与人对战的斗地主 AI，在这个过程中我们掌握了各个算法的原理，与基于 Python 的实现，最终设计的人工智能算法具有不错的性能。

不足与反思：由于家用机机能受限，我们被迫对手牌状态进行了认为主观的压缩，这导致了我们的智能算法过快的收敛，无法达到更强的性能，希望能在之后，基于更强的机能进行改进。