Solutions for Chapter 5
Adversarial Search

5.1 The translation uses the model of the opponent $OM(s)$ to fill in the opponent's actions, leaving our actions to be determined by the search algorithm. Let $P(s)$ be the state predicted to occur after the opponent has made all their moves according to OM. Note that the opponent may take multiple moves in a row before we get a move, so we need to define this recursively. We have $P(s) = s$ if PLAYERs is us or TERMINAL-TESTs is true, otherwise $P(s) = P(\text{RESULT}(s, OM(s))$.

The search problem is then given by:

a. Initial state: $P(S_0)$ where S_0 is the initial game state. We apply P as the opponent may play first

b. Actions: defined as in the game by ACTIONSs.

c. Successor function: $\text{RESULT}'(s, a) = P(\text{RESULT}(s, a))$

d. Goal test: goals are terminal states

e. Step cost: the cost of an action is zero unless the resulting state s' is terminal, in which case its cost is $M - \text{UTILITY}(s')$ where $M = \max_s \text{UTILITY}(s)$. Notice that all costs are non-negative.

Notice that the state space of the search problem consists of game state where we are to play and terminal states. States where the opponent is to play have been compiled out. One might alternatively leave those states in but just have a single possible action.

Any of the search algorithms of Chapter 3 can be applied. For example, depth-first search can be used to solve this problem, if all games eventually end. This is equivalent to using the minimax algorithm on the original game if $OM(s)$ always returns the minimax move in s.

5.2

a. Initial state: two arbitrary 8-puzzle states. Successor function: one move on an unsolved puzzle. (You could also have actions that change both puzzles at the same time; this is OK but technically you have to say what happens when one is solved but not the other.) Goal test: both puzzles in goal state. Path cost: 1 per move.

b. Each puzzle has $9!/2$ reachable states (remember that half the states are unreachable). The joint state space has $(9!)^2/4$ states.

c. This is like backgammon; expectiminimax works.

第5章的解决方案
对抗性搜索

5.1翻译使用对手OM（s）的模型来填写对手的动作，让我们的动作由搜索算法决定。 设P（s）是对手根据OM做出所有动作后预测发生的状态。 请注意，在我们得到一个移动之前，oppent可能会连续采取多个移动，所以我们需要递归地定义这个。 我们有P(s)=s如果P(s是us或T　AL−T EST s是真的，否则P(s)=P(R ESULT(s,OM(s))。

搜索问题然后由给出:

a. 初始状态：P(S0)其中S0为初始游戏状态。 我们应用P，因为对手可能会先打b。 动作:定义为在游戏中由一个CTS s. c. 后继函数：R ESULT'(s,a)=P(R ESULT(s,a))d。 目标测试：目标是终端状态e。 步骤成本：除非得到的状态s'是终端，否则操作的成本为零，在这种情况下，它的成本是M-U TILITY（s'），其中M=max S U TILITY（s）。 请注意，所有费用均为非负数。

请注意，搜索问题的状态空间由游戏状态和终端状态组成。 对手要上场的州已经编好了. 人们也可以把这些国家留在那里，但只有一个可能的行动。

可以应用第3章的任何搜索算法。 例如，深度优先搜索可用于解决此问题，如果所有游戏最终结束。 这相当于在原始游戏上使用minimax算法，如果OM（s）总是返回s中的minimax移动。

5.2

a. 初始状态：两个任意的8拼图状态。 后继功能：在未解决的难题上移动一步。（你也可以在同一时间改变两个谜题的动作;这是可以的，但从技术上讲，你必须说当一个解决而不是另一个时会发生什么。）

目标测试：两个难题在目标状态。 路径成本：每移动1。 b. 每个谜题都有9个！ / 2可达状态（记住一半的状态是不可达的）。 联合状态空间具有（9！）2/4个州。 c. 这就像步步高;expectiminimax工作。

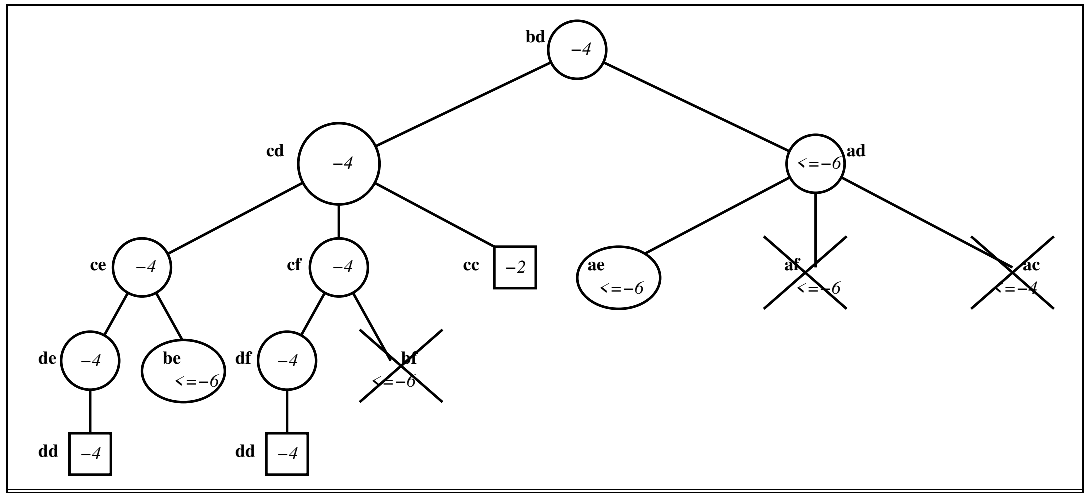


Figure S5.1 Pursuit-evasion solution tree.

d. Actually the statement in the question is not true (it applies to a previous version of part (c) in which the opponent is just trying to prevent you from winning—in that case, the coin tosses will eventually allow you to solve one puzzle without interruptions). For the game described in (c), consider a state in which the coin has come up heads, say, and you get to work on a puzzle that is 2 steps from the goal. Should you move one step closer? If you do, your opponent wins if he tosses heads; or if he tosses tails, you toss tails, and he tosses heads; or any sequence where both toss tails n times and then he tosses heads. So his probability of winning is *at least* $1/2+1/8+1/32+\cdots = 2/3$. So it seems you're better off moving *away* from the goal. (There's no way to stay the same distance from the goal.) This problem unintentionally seems to have the same kind of solution as suicide tictactoe with passing.

5.3

a. See Figure S5.1; the values are just (minus) the number of steps along the path from the root.

b. See Figure S5.1; note that there is both an upper bound and a lower bound for the left child of the root.

c. See figure.

d. The shortest-path length between the two players is a lower bound on the total capture time (here the players take turns, so no need to divide by two), so the "?" leaves have a capture time greater than or equal to the sum of the cost from the root and the shortest-path length. Notice that this bound is derived when the Evader plays very badly. The true value of a node comes from best play by both players, so we can get better bounds by assuming better play. For example, we can get a better bound from the cost when the Evader simply moves backwards and forwards rather than moving towards the Pursuer.

e. See figure (we have used the simple bounds). Notice that once the right child is known

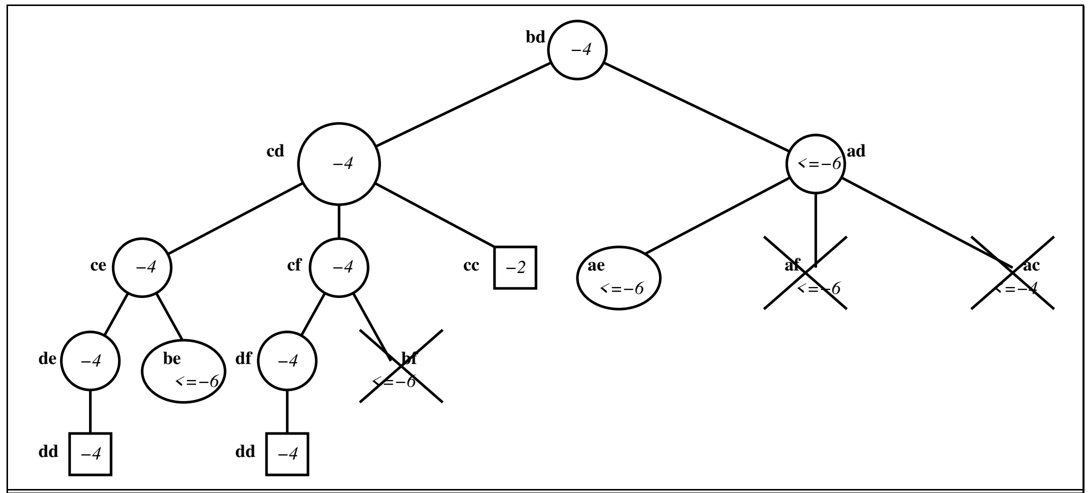


Figure S5.1 追逃解树。

d. 实际上问题中的陈述是不正确的（它适用于以前版本的部分（c），其中对手只是试图阻止你获胜—在这种情况下，硬币投掷最终将允许你解决一个难题而不 对于（c）中描述的游戏，考虑一个状态，在这个状态下，硬币已经出现了头，比如说，你可以在一个离目标2步的谜题上工作。 你应该再靠近一步吗？ 如果你这样做，你的对手就会赢，如果他抛头；或者如果他抛尾，你抛尾，他抛头；或者任何一个序列都抛尾n次，然后他抛头。 所以他获胜的概率至少是1 / 2+1 / 8+1 / 32+ · · · = 2 / 3 . 所以看来你最好离开球门。 （没有办法与目标保持相同的距离。）这个问题无意中似乎与通过自杀tictactoe具有相同的解决方案。

5.3

a. 参见图S5.1；这些值只是（减去）从根路径沿路径的步数。 b. 请参阅图S5.1;注意，根的左子项同时存在上界和下界。 c. 见图。 d. 两个玩家之间的最短路径长度是总捕获时间的下限（这里玩家轮流，所以不需要除以两个），所以"？"叶子具有大于或等于从根和最短路径长度的成本之和的捕获时间。 请注意，此绑定是在逃避者玩得很糟糕时派生的。 节点的真正价值来自两个玩家的最佳发挥，因此我们可以通过假设更好的发挥来获得更好的边界。 例如，当逃避者简单地向后和向前移动而不是向追求者移动时，我们可以从成本中获得更好的约束。 e. 见图（我们使用了简单的边界）。 请注意，一旦知道合适的孩子

to have a value below –6, the remaining successors need not be considered.

f. The pursuer always wins if the tree is finite. To prove this, let the tree be rooted as the pursuer's current node. (I.e., pick up the tree by that node and dangle all the other branches down.) The evader must either be at the root, in which case the pursuer has won, or in some subtree. The pursuer takes the branch leading to that subtree. This process repeats at most d times, where d is the maximum depth of the original subtree, until the pursuer either catches the evader or reaches a leaf node. Since the leaf has no subtrees, the evader must be at that node.

5.4 The basic physical state of these games is fairly easy to describe. One important thing to remember for Scrabble and bridge is that the physical state is not accessible to all players and so cannot be provided directly to each player by the environment simulator. Particularly in bridge, each player needs to maintain some best guess (or multiple hypotheses) as to the actual state of the world. We expect to be putting some of the game implementations online as they become available.

5.5 Code not shown.

5.6 The most obvious change is that the space of actions is now continuous. For example, in pool, the cueing direction, angle of elevation, speed, and point of contact with the cue ball are all continuous quantities.

The simplest solution is just to discretize the action space and then apply standard methods. This might work for tennis (modelled crudely as alternating shots with speed and direction), but for games such as pool and croquet it is likely to fail miserably because small changes in direction have large effects on action outcome. Instead, one must analyze the game to identify a discrete set of meaningful local goals, such as "potting the 4-ball" in pool or "laying up for the next hoop" in croquet. Then, in the current context, a local optimization routine can work out the best way to achieve each local goal, resulting in a discrete set of possible choices. Typically, these games are stochastic, so the backgammon model is appropriate provided that we use sampled outcomes instead of summing over all outcomes.

Whereas pool and croquet are modelled correctly as turn-taking games, tennis is not. While one player is moving to the ball, the other player is moving to anticipate the opponent's return. This makes tennis more like the simultaneous-action games studied in Chapter 17. In particular, it may be reasonable to derive *randomized* strategies so that the opponent cannot anticipate where the ball will go.

5.7 Consider a MIN node whose children are terminal nodes. If MIN plays suboptimally, then the value of the node is greater than or equal to the value it would have if MIN played optimally. Hence, the value of the MAX node that is the MIN node's parent can only be increased. This argument can be extended by a simple induction all the way to the root. *If the suboptimal play by* MIN *is predictable*, then one can do better than a minimax strategy. For example, if MIN always falls for a certain kind of trap and loses, then setting the trap guarantees a win even if there is actually a devastating response for MIN. This is shown in Figure S5.2.

5.8

若要使值低于-6，则不需要考虑剩余的继任者。 f. 如果树是有限的，追求者总是赢。 为了证明这一点，让树根植为追求者的当前节点。 （即，拿起该节点的树，并将所有其他分支悬挂下来。 逃避者必须在根部，在这种情况下，追逐者赢了，或者在某个子树中。 追求者采取导致该子树的分支。 这个过程最多重复d次，其中d是原始子树的最大深度，直到追逐者抓住逃避者或到达叶节点。 由于叶子没有子树，因此逃避者必须位于该节点。

5.4这些游戏的基本物理状态相当容易描述。 拼字游戏和桥牌需要记住的一件重要事情是，物理状态不是所有玩家都可以访问的，因此环境模拟器不能直接提供给每个玩家。 特别是在桥牌中，每个玩家都需要对世界的实际状态保持一些最佳猜测（或多个假设）。 我们希望将一些游戏实现放在网上，因为它们变得可用。

5.5 未示出的代码。

5.6最明显的变化是行动空间现在是连续的。 例如，在池中，提示方向、仰角、速度和与母球的接触点都是连续的量。

最简单的解决方案只是将动作空间离散化，然后应用标准的甲基ods。 这可能适用于网球（模拟为速度和方向交替击球），但对于台球和槌球等游戏来说，它可能会惨遭失败，因为方向的微小变化对动作结果有很大的影响。 相反，必须分析游戏以确定一组离散的有意义的本地目标，例如在游泳池中"灌封4球"或在槌球中"铺设下一个箍"。 然后，在当前上下文中，局部优化例程可以制定出实现每个局部目标的最佳方式，从而产生离散的pos sible选择集合。 通常情况下，这些游戏是随机的，所以五子棋模型是合适的，前提是我们使用采样结果而不是对所有结果求和。

而游泳池和槌球被正确地模拟为转弯游戏，网球则不是。
当一名球员向球移动时，另一名球员正在移动以预测对手的回归。 这使得网球更像是第17章中研究的同时动作游戏。 特别是，推导随机化策略可能是合理的，这样对手就无法预测球会去哪里。

5.7考虑一个MIN节点，其子节点是终端节点。 如果MIN以次优方式播放，则节点的值大于或等于MIN以最佳方式播放时所具有的值。 因此，作为MIN节点的父节点的MAX节点的值只能增加。 这个论点可以通过一个简单的归纳一直延伸到根。 如果MIN的次优游戏是可预测的，那么人们可以做得比minimax策略更好。 例如，如果MIN总是为某种陷阱而摔倒并且输了，那么设置陷阱可以保证胜利，即使实际上对MIN有毁灭性的反应。 这在图S5.2中示出。

5.8

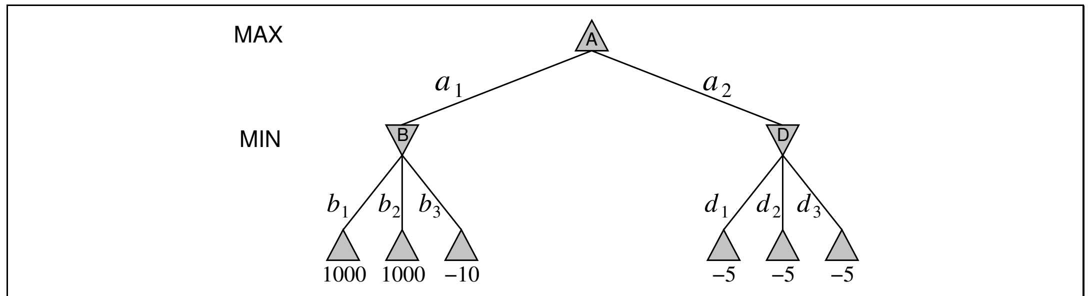


Figure S5.2 A simple game tree showing that setting a trap for MIN by playing a_i is a win if MIN falls for it, but may also be disastrous. The minimax move is of course a_2, with value -5.

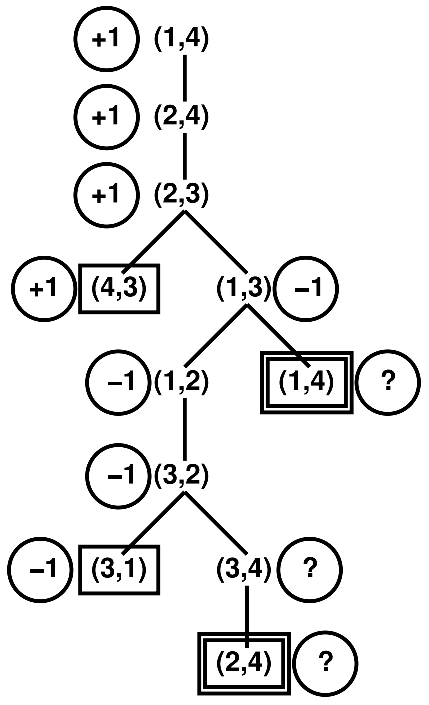
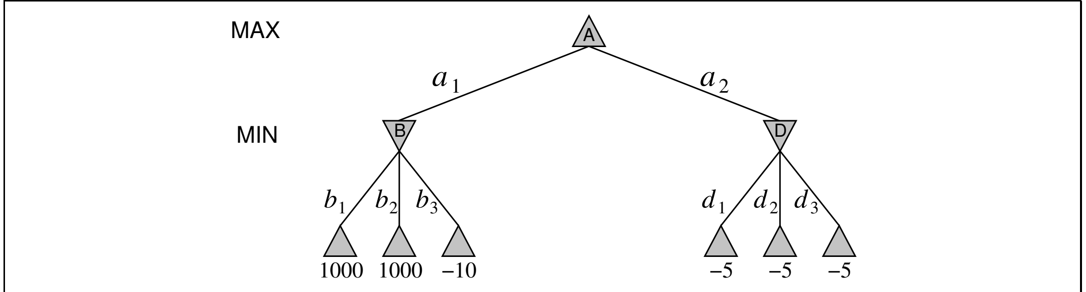


Figure S5.3 The game tree for the four-square game in Exercise 5.8. Terminal states are in single boxes, loop states in double boxes. Each state is annotated with its minimax value in a circle.

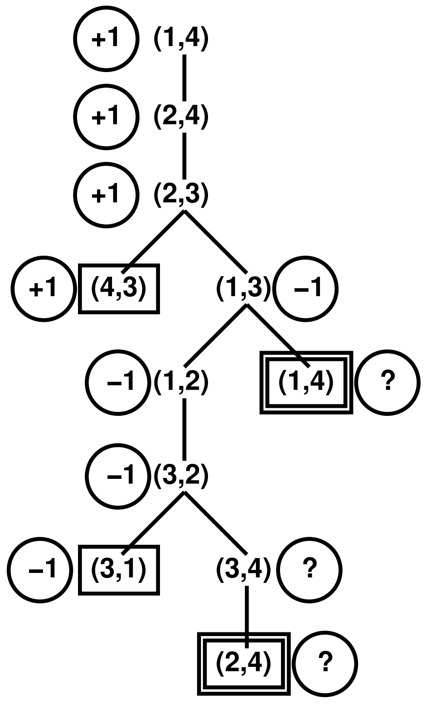
a. (5) The game tree, complete with annotations of all minimax values, is shown in Figure S5.3.

b. (5) The "?" values are handled by assuming that an agent with a choice between winning the game and entering a "?" state will always choose the win. That is, min(–1,?) is –1 and max(+1,?) is +1. If all successors are "?", the backed-up value is "?".

c. (5) Standard minimax is depth-first and would go into an infinite loop. It can be fixed



图S5.2一个简单的游戏树显示，通过玩i为MIN设置一个陷阱是一个胜利，如果MIN跌倒了，但也可能是灾难性的。 最小移动当然是2,值-5.



图S5.3练习5.8中四平方游戏的游戏树。 终端状态在单盒中，循环状态在双盒中。 每个状态都以其极小值在一个圆圈中注释。

a.（5）游戏树，包括所有极小值的注释，如图S5.3所示。 b.（5）的"? "价值是通过假设一个代理人在赢得比赛和进入a之间做出选择来处理的"? "国家总是会选择胜利。 即，min(−1，？ ）是−1和max（+1，？)为+1。 如果所有的继任者都是" ？ "，备份值是"? ″. c.（5）标准极小值是深度优先的，会进入一个无限循环。 它可以固定

by comparing the current state against the stack; and if the state is repeated, then return a "?" value. Propagation of "?" values is handled as above. Although it works in this case, it does not *always* work because it is not clear how to compare "?" with a drawn position; nor is it clear how to handle the comparison when there are wins of different degrees (as in backgammon). Finally, in games with chance nodes, it is unclear how to compute the average of a number and a "?". Note that it is *not* correct to treat repeated states automatically as drawn positions; in this example, both (1,4) and (2,4) repeat in the tree but they are won positions.

What is really happening is that each state has a well-defined but initially unknown value. These unknown values are related by the minimax equation at the bottom of 164. If the game tree is acyclic, then the minimax algorithm solves these equations by propagating from the leaves. If the game tree has cycles, then a dynamic programming method must be used, as explained in Chapter 17. (Exercise 17.7 studies this problem in particular.) These algorithms can determine whether each node has a well-determined value (as in this example) or is really an infinite loop in that both players prefer to stay in the loop (or have no choice). In such a case, the rules of the game will need to define the value (otherwise the game will never end). In chess, for example, a state that occurs 3 times (and hence is assumed to be desirable for both players) is a draw.

d. This question is a little tricky. One approach is a proof by induction on the size of the game. Clearly, the base case $n=3$ is a loss for A and the base case $n=4$ is a win for A. For any $n > 4$, the initial moves are the same: A and B both move one step towards each other. Now, we can see that they are engaged in a subgame of size $n-2$ on the squares $[2,\ldots,n-1]$, *except* that there is an extra choice of moves on squares 2 and $n-1$. Ignoring this for a moment, it is clear that if the "$n-2$" is won for A, then A gets to the square $n-1$ before B gets to square 2 (by the definition of winning) and therefore gets to n before B gets to 1, hence the "n" game is won for A. By the same line of reasoning, if "$n-2$" is won for B then "n" is won for B. Now, the presence of the extra moves complicates the issue, but not too much. First, the player who is slated to win the subgame $[2,\ldots,n-1]$ never moves back to his home square. If the player slated to lose the subgame does so, then it is easy to show that he is bound to lose the game itself—the other player simply moves forward and a subgame of size $n-2k$ is played one step closer to the loser's home square.

5.9 For **a**, there are at most 9! games. (This is the number of move sequences that fill up the board, but many wins and losses end before the board is full.) For **b–e**, Figure S5.4 shows the game tree, with the evaluation function values below the terminal nodes and the backed-up values to the right of the non-terminal nodes. The values imply that the best starting move for X is to take the center. The terminal nodes with a bold outline are the ones that do not need to be evaluated, assuming the optimal ordering.

5.10

a. An upper bound on the number of terminal nodes is $N!$, one for each ordering of squares, so an upper bound on the total number of nodes is $\sum_{i=1}^{N} i!$. This is not much

通过将当前状态与堆栈进行比较；如果状态重复，则返回"？"价值。 传播"？" 值如上处理。 虽然它在这种情况下工作，但它并不总是有效，因为不清楚如何比较"？"用一个绘制的位置;也不清楚如何处理比较时，有不同程度的胜利（如在步步高）。 最后，在具有机会节点的游戏中，目前还不清楚如何计算一个数字和一个"的平均值？".请注意，自动将重复状态视为绘制位置是不正确的;在这个例子中，（1,4）和（2,4）都在树中重复，但它们是赢得的位置。

真正发生的是，每个状态都有一个定义明确但最初未知的值。 这些未知值由164底部的minimax方程相关。 如果游戏树是非循环的，那么minimax算法通过从叶子传播来解决这些方程。 如果游戏树有循环，那么必须使用动态编程方法，如第17章所述。 （练习17.7特别研究这个问题。）这些算法可以确定每个节点是否具有确定的值（如本例中所示），或者是否真的是一个无限循环，因为两个玩家都喜欢留在循环中（或者别无选择）。 在这种情况下，游戏规则将需要定义值（否则游戏将永远不会结束）。 例如，在国际象棋中，发生3次的状态（因此被认为对两个玩家都是可取的）是平局。 d. 这个问题有点棘手。 一种方法是通过对游戏大小的归纳来证明。 显然，基本情况n=3是A的损失，基本情况n=4是A的胜利。对于任何n>4，初始移动都是相同的：A和B都向对方移动一步。 现在，我们可以看到他们在正方形[2,]上参与了一个大小为n-2的子游戏。 ..，n−1]，除了在方格2和n-1上有额外的移动选择。 忽略这一点，很明显，如果A赢得了"n-2"，那么A在B到达正方形2之前（根据获胜的定义）到达正方形n-1，因此在B到达1之前到达n，因此"n"游戏为A赢得。 首先，谁是计划赢得子游戏的球员[2，。 ..,n-1]永远不会搬回他的家乡广场。 如果打算输掉子游戏的玩家这样做，那么很容易表明他必然会输掉游戏本身—另一个玩家只需向前移动，一个大小为n-2k的子游戏就会离输家的主场更近一步。

5.9对于a，最多有9个！ 游戏。 （这是填满棋盘的移动序列的数量，但许多胜利和失败在棋盘充满之前结束。）对于b–e，图S5.4显示了游戏树，评估函数值低于终端节点，备份值位于非终端节点的右侧。 这些值意味着X的最佳起始移动是取中心。具有粗体轮廓的终端节点是不需要评估的节点，假设为最佳排序。

5.10

a. 终端节点个数上界为N！ ，每个平方排序一个，所以节点总数的上界是N
i=1我！ . 这不多

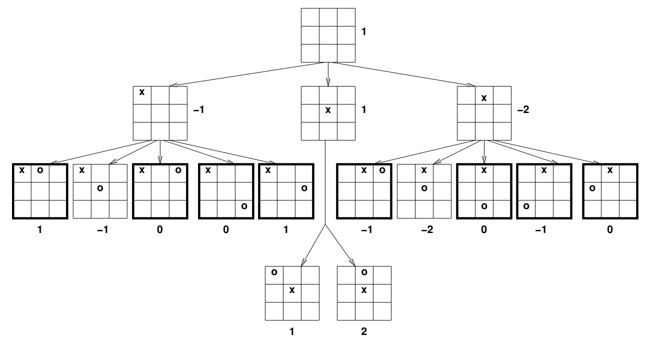


Figure S5.4 Part of the game tree for tic-tac-toe, for Exercise 5.9.

bigger than $N!$ itself as the factorial function grows superexponentially. This is an overestimate because some games will end early when a winning position is filled.

This count doesn't take into account transpositions. An upper bound on the number of distinct game states is 3^N, as each square is either empty or filled by one of the two players. Note that we can determine who is to play just from looking at the board.

b. In this case no games terminate early, and there are $N!$ different games ending in a draw. So ignoring repeated states, we have exactly $\sum_{i=1}^{N} i!$ nodes.

At the end of the game the squares are divided between the two players: $\lceil N/2 \rceil$ to the first player and $\lfloor N/2 \rfloor$ to the second. Thus, a good lower bound on the number of distinct states is $\binom{N}{\lceil N/2 \rceil}$, the number of distinct terminal states.

c. For a state s, let $X(s)$ be the number of winning positions containing no O's and $O(s)$ the number of winning positions containing no X's. One evaluation function is then $Eval(s) = X(s) - O(S)$. Notice that empty winning positions cancel out in the evaluation function.

Alternatively, we might weight potential winning positions by how close they are to completion.

d. Using the upper bound of $N!$ from (a), and observing that it takes $100NN!$ instructions. At 2GHz we have 2 billion instructions per second (roughly speaking), so solve for the largest N using at most this many instructions. For one second we get $N = 9$, for one minute $N = 11$, and for one hour $N = 12$.

5.11 See `"search/algorithms/games.lisp"` for definitions of games, game-playing agents, and game-playing environments. `"search/algorithms/minimax.lisp"` contains the minimax and alpha-beta algorithms. Notice that the game-playing environment is essentially a generic environment with the update function defined by the rules of the game. Turn-taking is achieved by having agents do nothing until it is their turn to move.

See `"search/domains/cognac.lisp"` for the basic definitions of a simple game (slightly more challenging than Tic-Tac-Toe). The code for this contains only a trivial evaluation function. Students can use minimax and alpha-beta to solve small versions of the game to termination (probably up to 4×3); they should notice that alpha-beta is far faster

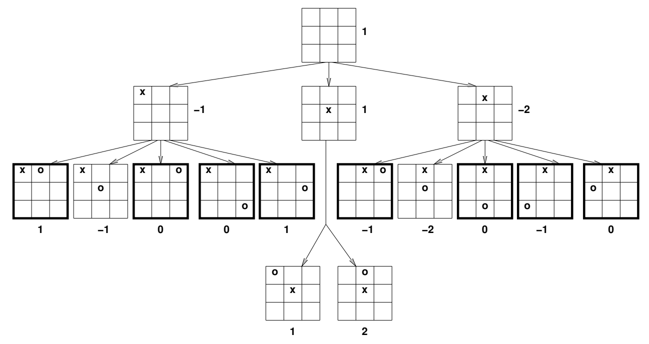


Figure S5.4 游戏树的一部分井字棋，用于练习5.9。

比N大！ 本身作为阶乘函数超指数增长。 这是一个过高的估计，因为有些游戏会提前结束时，一个获胜的位置填补。

这个计数没有考虑到转置。 不同游戏状态数量的上限是3n，因为每个方块要么是空的，要么由两个玩家中的一个填满。 请注意，我们可以确定谁是发挥只是从看板。 b. 在这种情况下，没有游戏提前终止，并且有N！ 不同的游戏以平局结束。 所以忽略重复的状态，我们有确切的N

i=1我！ 节点。 在游戏结束时，方块在两个玩家之间划分： N/2 到第一个玩家， N/2 到第二个玩家。 因此，不同状态数量的一个很好的下限是

，不同终端状态的数量。

c. 对于状态s，设X(s)是不包含O's的获胜位置数，O(s)是不包含X's的获胜位置数。 请注意，空的获胜位置在eval uation函数中取消。

或者，我们可能会通过他们接近完成的程度来衡量潜在的获胜位置。 d. 使用N的上界！ 从（a），并观察它需要100NN！ 指示。 在2GHz时，我们每秒有20亿条指令（粗略地说），所以最多使用这么多指令来求解最大的N。 一秒钟我们得到N=9，一分钟N=11，一小时N=12。

5.11参见"搜索/算法/游戏"。lisp"用于游戏、游戏代理和游戏环境的定义。 "搜索/算法/minimax。lisp"con tains minimax和alpha-beta算法。 请注意，游戏环境本质上是一个通用环境，具有由游戏规则定义的更新功能。 轮流是通过让代理人在轮到他们移动之前什么都不做来实现的。

请参阅"搜索/域/科涅克。lisp"用于简单游戏的基本定义（比井字游戏稍微更具挑战性）。 此代码只包含一个简单的eval uation函数。 学生可以使用minimax和alpha-beta来解决游戏的小版本终止（可能高达4×3）;他们应该注意到alpha-beta要快得多

than minimax, but still cannot scale up without an evaluation function and truncated horizon. Providing an evaluation function is an interesting exercise. From the point of view of data structure design, it is also interesting to look at how to speed up the legal move generator by precomputing the descriptions of rows, columns, and diagonals.

Very few students will have heard of kalah, so it is a fair assignment, but the game is boring—depth 6 lookahead and a purely material-based evaluation function are enough to beat most humans. Othello is interesting and about the right level of difficulty for most students. Chess and checkers are sometimes unfair because usually a small subset of the class will be experts while the rest are beginners.

5.12 The minimax algorithm for non-zero-sum games works exactly as for multiplayer games, described on p.165–6; that is, the evaluation function is a vector of values, one for each player, and the backup step selects whichever vector has the highest value for the player whose turn it is to move. The example at the end of Section 5.2.2 (p.165) shows that alpha-beta pruning is not possible in general non-zero-sum games, because an unexamined leaf node might be optimal for both players.

5.13 This question is not as hard as it looks. The derivation below leads directly to a definition of α and β values. The notation n_i refers to (the value of) the node at depth i on the path from the root to the leaf node n_j. Nodes $n_{i1} \ldots n_{i_{b_i}}$ are the siblings of node i.

a. We can write $n_2 = \max(n_3, n_{31}, \ldots, n_{3b_3})$, giving

$$n_1 = \min(\max(n_3, n_{31}, \ldots, n_{3b_3}), n_{21}, \ldots, n_{2b_2})$$

Then n_3 can be similarly replaced, until we have an expression containing n_j itself.

b. In terms of the l and r values, we have

$$n_1 = \min(l_2, \max(l_3, n_3, r_3), r_2)$$

Again, n_3 can be expanded out down to n_j. The most deeply nested term will be $\min(l_j, n_j, r_j)$.

c. If n_j is a max node, then the lower bound on its value only increases as its successors are evaluated. Clearly, if it exceeds l_j it will have no further effect on n_1. By extension, if it exceeds $\min(l_2, l_4, \ldots, l_j)$ it will have no effect. Thus, by keeping track of this value we can decide when to prune n_j. This is exactly what α-β does.

d. The corresponding bound for min nodes n_k is $\max(l_3, l_5, \ldots, l_k)$.

5.14 The result is given in Section 6 of Knuth (1975). The exact statement (Corollary 1 of Theorem 1) is that the algorithms examines $b^{\lfloor m/2 \rfloor} + b^{\lceil m/2 \rceil} - 1$ nodes at level m. These are exactly the nodes reached when Min plays only optimal moves and/or Max plays only optimal moves. The proof is by induction on m.

5.15 With 32 pieces, each needing 6 bits to specify its position on one of 64 squares, we need 24 bytes (6 32-bit words) to store a position, so we can store roughly 80 million positions in the table (ignoring pointers for hash table bucket lists). This is about 1/22 of the 1800 million positions generated during a three-minute search.

比极小值，但仍然不能扩大没有评估函数和截断地平线。 提供一个评估函数是一个有趣的练习。 从数据结构设计的角度来看，看看如何通过预先计算行，列和对角线的描述来加速合法移动生成器也很有趣。

很少有学生会听说过kalah，所以这是一个公平的作业，但游戏很无聊-深度6lookahead和纯粹基于材料的评估功能足以击败大多数人类。 奥赛罗很有趣,对大多数学生来说,它的难度是正确的. 国际象棋和跳棋有时是不公平的，因为通常班级的一小部分将是专家，而其余的是初学者。

5.12非零和游戏的minimax算法与多人游戏的工作原理完全相同，如第165-6页所述；也就是说，评估函数是一个值的向量，每个玩家一个，备份步骤选择哪个向量对轮到移动的玩家具有最高值。 第5.2.2节（第165页）末尾的例子表明，在一般的非零和游戏中，阿尔法贝塔修剪是不可能的，因为未经检查的叶节点可能对两个玩家都是最佳的。

5.13这个问题并不像看起来那么难。 下面的推导直接导致α和β值的定义. 符号n i是指从根到叶节点n j的路径上深度i处的节点的值。 节点n i1。 . . n i bi是节点i的兄弟姐妹。

a. 我们可以写出n2=max(n3,n31,. . . ，n3b3），给予

$$n_1 = \min(\max(n_3, n_{31}, \ldots, n_{3b_3}), n_{21}, \ldots, n_{2b_2})$$

然后n3可以被类似地替换，直到我们有一个包含n j本身的表达式。 b. 在l和r值方面，我们有

$$n_1 = \min(l_2, \max(l_3, n_3, r_3), r_2)$$

同样，n3可以向下扩展到n j。 嵌套最深的项将是min(l j,n j,r j)。 c. 如果n j是max节点，则其值的下限仅随着其后续节点的计算而增加。 显然，如果它超过l j它将对n 1没有进一步的影响。 通过扩展，如果超过min(l2，l4，。. . ，l j）它不会有任何影响。 因此，通过跟踪该值，我们可以决定何时修剪n j。 这正是α β所做的。 d. Min节点n k的对应绑定为max(l3,l5,。 . . ，l k)。

5.14
定理1）是算法在m级检查b　m/2　+b　m/2　-1节点。 这些正是Min只播放最佳移动和/或Max只播放最佳移动时到达的节点。 证明是通过对m的归纳。

5.15对于32个块，每个块需要6位来指定它在64个正方形中的一个上的位置，我们需要24个字节（6个32位字）来存储一个位置，所以我们可以在表中存储大约8000万个位置（忽略哈希表桶列表的指针）。 这大约是三分钟搜索期间产生的1800百万个职位的1/22。

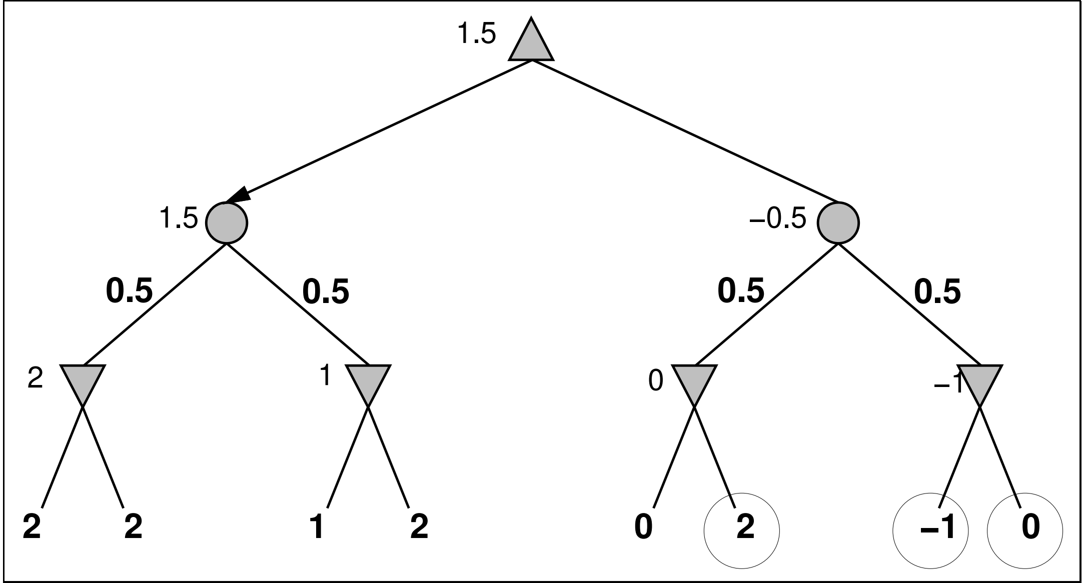


Figure S5.5 Pruning with chance nodes solution.

Generating the hash key directly from an array-based representation of the position might be quite expensive. Modern programs (see, e.g., Heinz, 2000) carry along the hash key and modify it as each new position is generated. Suppose this takes on the order of 20 operations; then on a 2GHz machine where an evaluation takes 2000 operations we can do roughly 100 lookups per evaluation. Using a rough figure of one millisecond for a disk seek, we could do 1000 evaluations per lookup. Clearly, using a disk-resident table is of dubious value, even if we can get some locality of reference to reduce the number of disk reads.

5.16

a. See Figure S5.5.

b. Given nodes 1–6, we would need to look at 7 and 8: if they were both $+\infty$ then the values of the min node and chance node above would also be $+\infty$ and the best move would change. Given nodes 1–7, we do not need to look at 8. Even if it is $+\infty$, the min node cannot be worth more than -1, so the chance node above cannot be worth more than -0.5, so the best move won't change.

c. The worst case is if either of the third and fourth leaves is -2, in which case the chance node above is 0. The best case is where they are both 2, then the chance node has value 2. So it must lie between 0 and 2.

d. See figure.

5.18 The general strategy is to reduce a general game tree to a one-ply tree by induction on the depth of the tree. The inductive step must be done for min, max, and chance nodes, and simply involves showing that the transformation is carried though the node. Suppose that the values of the descendants of a node are $x_1 \ldots x_n$, and that the transformation is $ax+b$, where

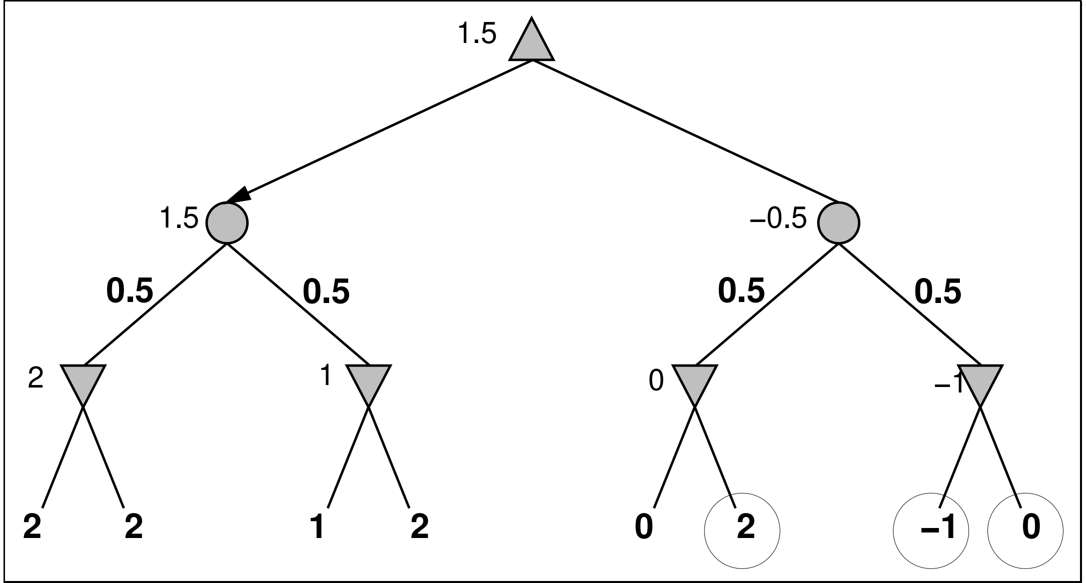


Figure S5.5 用机会节点解决方案修剪。

直接从基于数组的位置表示生成哈希密钥可能非常昂贵。 现代程序（参见，例如Heinz，2000）携带哈希密钥，并在每个新位置生成时对其进行修改。 假设这需要20个操作的顺序;然后在评估需要2000个操作的2GHz机器上，我们可以每个评估进行大约100次查找。 使用一毫秒的磁盘查找粗略数字，我们可以在每次查找中进行1000次评估。 显然，使用磁盘驻留表具有可疑的价值，即使我们可以获得一些引用的局部来减少磁盘读取次数。

5.16

a. 见图S5.5。 b. 给定节点1-6，我们需要查看7和8：如果它们都是+∞，那么上面的min节点和chance节点的值也将是+∞，并且最佳移动将发生变化。 给定节点1-7，我们不需要看8。 即使是+∞，min节点的值也不能超过-1，所以上面的chance节点的值也不能超过-0。 5，所以最好的举动不会改变。 c. 最坏的情况是，如果第三和第四叶中的任何一个是-2，在这种情况下，上面的机会节点是0。 最好的情况是它们都是2，那么机会节点的值为2。 所以它必须介于0和2之间。 d. 见图。

5.18一般策略是通过对树的深度进行归纳，将一般游戏树减少为单层树。 归纳步骤必须针对min，max和chance节点进行，并且简单地涉及显示转换通过节点进行。 假设一个节点的后代的值是x1。 . . x n，并且变换是ax+b，其中

a is positive. We have

$$\begin{aligned}\min(ax_1+b, ax_2+b, \ldots, ax_n+b) &= a\min(x_1, x_2, \ldots, x_n)+b\\ \max(ax_1+b, ax_2+b, \ldots, ax_n+b) &= a\min(x_1, x_2, \ldots, x_n)+b\\ p_1(ax_1+b)+p_2(ax_2+b)+\cdots+p_n(ax_n+b) &= a(p_1x_1+p_2x_2+\cdots p_nx_n)+b\end{aligned}$$

Hence the problem reduces to a one-ply tree where the leaves have the values from the original tree multiplied by the linear transformation. Since $x > y \Rightarrow ax+b > ay+b$ if $a > 0$, the best choice at the root will be the same as the best choice in the original tree.

5.19 This procedure will give incorrect results. Mathematically, the procedure amounts to assuming that averaging commutes with min and max, which it does not. Intuitively, the choices made by each player in the deterministic trees are based on full knowledge of future dice rolls, and bear no necessary relationship to the moves made without such knowledge. (Notice the connection to the discussion of card games in Section 5.6.2 and to the general problem of fully and partially observable Markov decision problems in Chapter 17.) In practice, the method works reasonably well, and it might be a good exercise to have students compare it to the alternative of using expectiminimax with sampling (rather than summing over) dice rolls.

5.20

- **a**. No pruning. In a max tree, the value of the root is the value of the best leaf. Any unseen leaf might be the best, so we have to see them all.
- **b**. No pruning. An unseen leaf might have a value arbitrarily higher or lower than any other leaf, which (assuming non-zero outcome probabilities) means that there is no bound on the value of any incompletely expanded chance or max node.
- **c**. No pruning. Same argument as in (a).
- **d**. No pruning. Nonnegative values allow *lower* bounds on the values of chance nodes, but a lower bound does not allow any pruning.
- **e**. Yes. If the first successor has value 1, the root has value 1 and all remaining successors can be pruned.
- **f**. Yes. Suppose the first action at the root has value 0.6, and the first outcome of the second action has probability 0.5 and value 0; then all other outcomes of the second action can be pruned.
- **g**. (ii) Highest probability first. This gives the strongest bound on the value of the node, all other things being equal.

5.21

- **a**. *In a fully observable, turn-taking, zero-sum game between two perfectly rational players, it does not help the first player to know what strategy the second player is using—that is, what move the second player will make, given the first player's move.*
  True. The second player will play optimally, and so is perfectly predictable up to ties. Knowing which of two equally good moves the opponent will make does not change the value of the game to the first player.

a为正。 我们有

$$\begin{aligned}\min(ax_1+b, ax_2+b, \ldots, ax_n+b) &= a\min(x_1, x_2, \ldots, x_n)+b\\ \max(ax_1+b, ax_2+b, \ldots, ax_n+b) &= a\min(x_1, x_2, \ldots, x_n)+b\\ p_1(ax_1+b)+p_2(ax_2+b)+\cdots+p_n(ax_n+b) &= a(p_1x_1+p_2x_2+\cdots p_nx_n)+b\end{aligned}$$

因此，问题简化为单层树，其中叶子具有原始树的值乘以线性变换。 由于x>y≠ax+b >ay+b如果a>0，则根部的最佳选择将与原始树中的最佳选择相同。

5.19这个程序会给出不正确的结果。 在数学上，该过程相当于假设平均与min和max上下班，而事实并非如此。 直观地说，每个玩家在确定性树中所做的选择都是基于对未来掷骰子的充分知识，并且与没有这些知识所做的动作没有必要的关系。 （请注意与第5.6.2节中纸牌游戏的讨论以及第17章中完全和部分可观察的马尔可夫决策问题的一般问题的联系。）在prac tice中，该方法工作得相当好，让学生将其与使用expectiminimax与采样（而不是求和）掷骰子的替代方法进行比较可能是一个很好的练习。

5.20

a. 没有修剪。 在max树中，根的值是最佳叶子的值。 任何看不见的叶子都可能是最好的，所以我们必须看到它们。 b. 没有修剪。 一个看不见的叶子的值可能比任何其他叶子任意高或低，这（假设结果概率为非零）意味着任何未完全扩展的机会或最大节点的值都没有限制。 c. 没有修剪。 与（a）中相同的论点。 d. 没有修剪。 非负值允许机会节点值的下限，但下限不允许任何修剪。 e. 是的. 如果第一个继承者具有值1，则根具有值1，并且可以修剪所有剩余的继承者。 f. 是的. 假设根部的第一个动作具有值0.6，并且第二个动作的第一个结果具有概率0.5和值0;然后可以修剪第二个动作的所有其他结果。 g. (ii)最高概率优先。 这给出了节点值的最强约束，所有其他东西都是相等的。

5.21

a. 在两个完全理性的玩家之间的完全可观察，转弯，零和游戏中，它无助于第一个玩家知道第二个玩家正在使用什么策略—也就是说，考虑到第一个玩家的 没错。 第二个球员将发挥最佳，所以是完全可预测的关系。 知道对手会做出两个同样好的动作中的哪一个并不会改变游戏对第一个玩家的价值。

b. *In a partially observable, turn-taking, zero-sum game between two perfectly rational players, it does not help the first player to know what move the second player will make, given the first player's move.*
False. In a partially observable game, knowing the second player's move tells the first player additional information about the game state that would otherwise be available only to the second player. For example, in Kriegspiel, knowing the opponent's future move tells the first player where one of the opponent's pieces is; in a card game, it tells the first player one of the opponent's cards.

c. *A perfectly rational backgammon agent never loses.*
False. Backgammon is a game of chance, and the opponent may consistently roll much better dice. The correct statement is that the *expected* winnings are optimal. It is suspected, but not known, that when playing first the expected winnings are positive even against an optimal opponent.

5.22 One can think of chance events during a game, such as dice rolls, in the same way as hidden but preordained information (such as the order of the cards in a deck). The key distinctions are whether the players can influence what information is revealed and whether there is any asymmetry in the information available to each player.

a. Expectiminimax is appropriate only for backgammon and Monopoly. In bridge and Scrabble, each player knows the cards/tiles he or she possesses but not the opponents'. In Scrabble, the benefits of a fully rational, randomized strategy that includes reasoning about the opponents' state of knowledge are probably small, but in bridge the questions of knowledge and information disclosure are central to good play.

b. None, for the reasons described earlier.

c. Key issues include reasoning about the opponent's beliefs, the effect of various actions on those beliefs, and methods for representing them. Since belief states for rational agents are probability distributions over all possible states (including the belief states of others), this is nontrivial.

b. 在两个完全理性的玩家之间的部分可观察的，转弯，零和游戏中，鉴于第一个玩家的举动，它无助于第一个玩家知道第二个玩家会做出什么举动。 错误。 在一个部分可观察的游戏中，知道第二个玩家的移动告诉第一个玩家关于游戏状态的额外信息，否则这些信息只能提供给第二个玩家。 例如，在Kriegspiel中，知道对手未来的移动会告诉第一个玩家对手的一个棋子在哪里;在纸牌游戏中，它会告诉第一个玩家对手的一张牌。 c. 一个完全理性的步步高特工永远不会失败。

错误。 步步高是一个机会游戏，对手可能会始终如一地掷出更好的骰子。 正确的说法是预期的奖金是最优的。 它是sus pected，但不知道，当第一次玩预期的奖金是积极的，即使对一个最佳的对手。

5.22人们可以用隐藏但预先规定的信息（如牌组中的牌顺序）来思考游戏中偶然发生的事件，如掷骰子。 关键的区别在于玩家是否能够影响所揭示的信息，以及每个玩家可用的信息是否存在任何不对称性。

a. Expectiminimax仅适用于步步高和垄断。 在桥牌和拼字游戏中，每个玩家都知道他或她拥有的牌/牌，但不知道对手。 在拼字游戏中，包括推理对手的知识状态的完全理性，随机化策略的好处可能很小，但在bridge中，知识和信息披露问题是良好游戏的核心。 b. 没有，由于前面所述的原因。 c. 关键问题包括对手信念的推理，各种行为对这些信念的影响，以及表示它们的方法。 由于理性代理的信念状态是所有可能状态（包括其他人的信念状态）的概率分布，因此这不是平凡的。