approximation: instead of adding up all the deals, we take a random sample of $N$ deals, where the probability of deal $a$ appearing in the sample is proportional to $P(s)$:

$$\text{argmax} \frac{1}{N} \sum_{i=1}^{N} \text{MINIMAX}(\text{RESULT}(s_i, a)).$$

(Notice that $P(s)$ does not appear explicitly in the summation, because the samples are already drawn according to $P(s)$.) As $N$ grows large, the sum over the random sample tends to the exact value, but even for fairly small $N$—say, 100 to 1,000—the method gives a good approximation. It can also be applied to deterministic games such as Kriegspiel, given some reasonable estimate of $P(s)$.

For games like whist and hearts, where there is no bidding or betting phase before play commences, each deal will be equally likely and so the values of $P(s)$ are all equal. For bridge, play is preceded by a bidding phase in which each team indicates how many tricks it expects to win. Since players bid based on the cards they hold, the other players learn more about the probability of each deal. Taking this into account in deciding how to play the hand is tricky, for the reasons mentioned in our description of Kriegspiel player: may hide in such a way as to minimize the information conveyed to their opponents. Even so, the approach is quite effective for bridge, as we show in Section 5.7.

The strategy described in Equations 5.1 and 5.2 is sometimes called averaging over clairvoyance because it assumes that the game will become observable to both players immediately after the first move. Despite its intuitive appeal, the strategy can lead one astray. Consider the following story:

Day 1: Road A leads to a heap of gold; Road B leads to a fork. Take the left fork and you’ll find a bigger heap of gold, but take the right fork and you’ll be run over by a bus.

Day 2: Road A leads to a heap of gold; Road B leads to a fork. Take the right fork and you’ll find a bigger heap of gold, but take the left fork and you’ll be run over by a bus.

Day 3: Road A leads to a heap of gold; Road B leads to a fork. One branch of the fork leads to a bigger heap of gold, but take the wrong fork and you’ll be hit by a bus.

Unfortunately, you don’t know which fork is which.

Averaging over clairvoyance leads to the following reasoning: on Day 1, $B$ is the right choice; on Day 2, $B$ is the right choice; on Day 3, the situation is the same as either Day 1 or Day 2. So $B$ must still be the right choice.

Now we can see how averaging over clairvoyance fails: it does not consider the belief state that the agent will be in after acting. A belief state of total ignorance is not desirable, especially when one possibility is certain death. Because it assumes that every future state will automatically be one of perfect knowledge, the approach never selects actions that gather information (like the first move in Figure 5A 3); nor will it choose actions that hide information from the opponent or provide information to a partner because it assumes that they already know the information; and it will never bluff in poker, because it assumes the opponent can see its cards. In Chapter 17, we show how to construct algorithms that do all these things by virtue of solving the true partially observable decision problem.

Bluffing—betting as if one’s hand is good, even when it’s not—is a core part of poker strategy.
In 1965, the Russian mathematician Alexander Kronrod called chess "the *Drosophila* of artificial intelligence." John McCarthy disagrees: whereas geneticists use fruit flies to make discoveries that apply to biology more broadly, AI has used chess to do the equivalent of breeding very fast fruit flies. Perhaps a better analogy is that chess is to AI as Grand Prix motor racing is to the car industry: state-of-the-art game programs are blindingly fast, highly optimized machines that incorporate the latest engineering advances, but they aren’t much use for doing the shopping or driving off-road. Nonetheless, racing and game-playing generate excitement and a steady stream of innovations that have been adopted by the wider community. In this section we look at what it takes to come out on top in various games.

**CHESS**: IBM’s DEEP BLUE chess program, now retired, is well known for defeating world champion Garry Kasparov in a widely publicized exhibition match. Deep Blue ran on a parallel computer with 30 IBM RS/6000 processors doing alpha-beta search. The unique part was a configuration of 480 custom VLSI chess processors that performed move generation and move ordering for the last few levels of the tree, and evaluated the leaf nodes. Deep Blue searched up to 30 billion positions per move, reaching depth 14 routinely. The key to its success seems to have been its ability to generate singular extensions beyond the depth limit for sufficiently interesting lines of forcing/forced moves. In some cases the search reached a depth of 40 plies. The evaluation function had over 8000 features, many of them describing highly specific patterns of pieces. An "opening book" of about 4000 positions was used, as well as a database of 700,000 grandmaster games from which consensus recommendations could be extracted. The system also used a large endgame database of solved positions containing all positions with five pieces and many with six pieces. This database had the effect of substantially extending the effective search depth, allowing Deep Blue to play perfectly in some cases even when it was many moves away from checkmate.

The success of DEEP BLUE reinforced the widely held belief that progress in computer game-playing has come primarily from ever-more-powerful hardware—a view encouraged by IBM. But algorithmic improvements have allowed programs running on standard PCs to win World Computer Chess Championships. A variety of pruning heuristics are used to reduce the effective branching factor to less than 3 (compared with the actual branching factor of about 35). The most important of these is the **null move** heuristic, which generates a good lower bound on the value of a position, using a shallow search in which the opponent gets to move twice at the beginning. This lower bound often allows alpha—beta pruning without the expense of a full-depth search. Also important is **futility pruning**, which helps decide in advance which moves will cause a beta cutoff in the successor nodes.

HYDRA can be seen as the successor to DEEP BLUE. HYDRA runs on a 64-processor cluster with 1 gigabyte per processor and with custom hardware in the form of FPGA (Field Programmable Gate Array) chips. HYDRA reaches 200 million evaluations per second, about the same as Deep Blue, but HYDRA reaches 18 plies deep rather than just 14 because of aggressive use of the **null** move heuristic and forward pruning.