Adversarial Search I: Playing: 2-Person, 0-Sum Games

CSE 415: Introduction to Artificial Intelligence
University of Washington
Spring, 2019

Outline

- Two-person, zero-sum games.
- Static evaluation functions.
- Minimax search.

Two-Person, Zero-Sum, Perfect Information Games

1. A two-person, zero-sum game is a game in which only one player wins and only one player loses. There may be ties ("draws"). There are no "win-win" or "lose-lose" instances.

2. Most 2PZS games involve turn taking. In each turn, a player makes a move. Turns alternate between the players.

3. Perfect information: no randomness as in Poker or bridge.

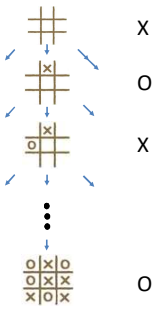
4. Examples of 2PZS games include Tic-Tac-Toe, Othello, Checkers, and Chess.

Why Study 2PZS Games in AI?

1. Games are idealizations of problems.
2. Some problems involve adversaries, such as companies competing in a marketplace.
3. A game opponent can serve as a proxy for the environment of an agent.
4. We'll transition from non-stochastic games to games with random aspects.

Tic-Tac-Toe

A tree search in Tic Tac Toe involves alternating levels for players X and O.



Static Evaluation Functions

In most of the interesting 2PZS games, state spaces are too large to exhaustively search each alternative evolutionary path to its end.

Thus we assume here that our agent does not have enough computational resources to completely explore the game's search tree starting from the current state, except when the game is almost over. So the search has to be cut off; this is typically based on a decision to look ahead some number *k* of moves to all states reachable in k or fewer moves.

To estimate how good a given game state *s* is, let's compute a real-valued function *h*(*s*) of the state: *h*(*s*) will be high if it is favorable to one player (the player we'll call Max) and unfavorable to the other player (whom we will call Min).

This function *h*(*s*) is called a *static evaluation function*.

Tic-Tac-Toe Static Eval. Fn.

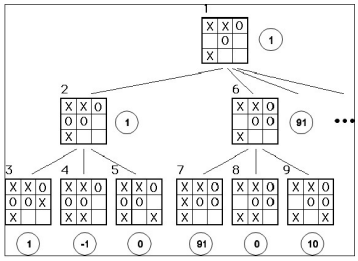
Here is a possible static evaluation function for Tic-Tac-Toe.

h(s) = 100 A + 10 B + C – (100 D + 10 E + F)

A = number of lines of 3 Xs in a row.
B = number of lines of 2 Xs in a row (not blocked by an O)
C = number of lines containing one X and no Os.

D = number of lines of 3 Os in a row.
E = number of lines of 2 Os in a row (not blocked by an X)
F = number of lines containing one O and no Xs.

Minimax Search (Illustration)



Minimax Search (Rationale)

If looking ahead one move, generate all successors of the current state, and apply the static evaluation function to each of them, and if we are Max, make the move that goes to the state with the maximum score.

If looking ahead two moves, we will be considering the positions that our opponent can get two in one move, from each of the positions that we can get to in one move.

Assuming that the opponent is playing rationally, the opponent, Min, will be trying to minimize the value of the resulting board.

Therefore, instead of using the static value at each successor of the current state, we examine the successors of each of those, computing their *static* values, and take the minimum of those as the value of our successor.

Minimax Search (Algorithm)

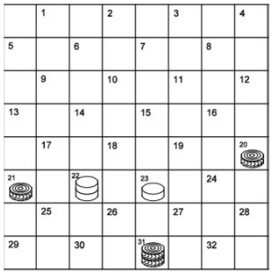
```
Procedure minimax(board, whoseMove, plyLeft):
        if plyLeft == 0:  return staticValue(board)
        if whoseMove == 'Max': provisional = -100000
        else: provisional = 100000
        for s in successors(board, whoseMove):
                newVal = minimax(s, other(whoseMove), plyLeft-1)
            if (whoseMove == 'Max' and newVal > provisional\
                or (whoseMove == 'Min' and newVal < provisional):
                        provisional = newVal
        return provisional
```


Checkers Static Evaluation

Example in Checkers:
$h(s) = 5x_1 + x_2$
Where x_1 = Max's king advantage;
x_2 = Max's single man advantage.

photo courtesy of thegamesupply.com.

Checkers Example



Black to move,
White = "Min",
Black = "Max"

Ss
State-space Search

Minimax Search Example