COMP219: Artificial Intelligence

Lecture 13: Game Playing

Overview

- Last time
  - Search with partial/no observations
  - Belief states
  - Incremental belief state search
  - Determinism vs non-determinism
- Today
  - We will look at how search can be applied to playing games
    - Types of games
    - Perfect play
      - minimax decisions
      - alpha–beta pruning
    - Playing with limited recourses

Games and Search

- In search we make all the moves. In games we play against an "unpredictable" opponent
  - Solution is a strategy specifying a move for every possible opponent reply
- Assume that the opponent is intelligent: always makes the best move
- Some method is needed for selecting good moves that stand a good chance of achieving a winning position, whatever the opponent does!
- There are time limits, so we are unlikely to find goal, and must approximate using heuristics

Types of Game

- In some games we have perfect information – the position is known completely
- In others we have imperfect information: e.g. we cannot see the opponent's cards
- Some games are deterministic – no random element
- Others have elements of chance (dice, cards)

Types of Games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

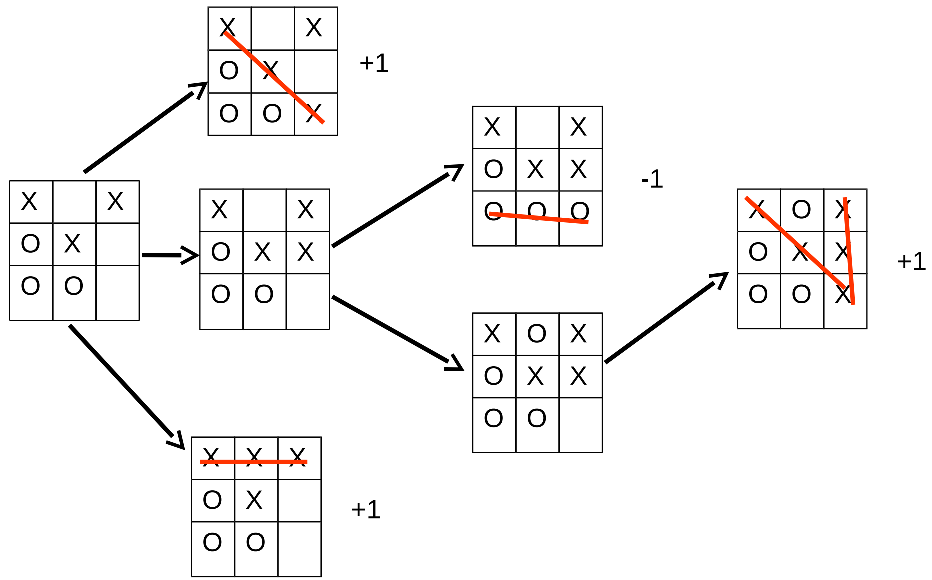
We will consider:

- Games that are:
  - Deterministic
  - Two-player
  - Zero-sum
    - the utility values at the end are equal and opposite
    - example: one wins (+1) the other loses (−1)
  - Perfect information
- E.g. Othello, Blitz Chess

Problem Formulation

- Initial state
  - Initial board position, player to move
- Transition model
  - List of (move, state) pairs, one per legal move
- Terminal test
  - Determines when the game is over
- Utility function
  - Numeric value for terminal states
  - e.g. Chess +1, -1, 0
  - e.g. Backgammon +192 to -192

Noughts and Crosses



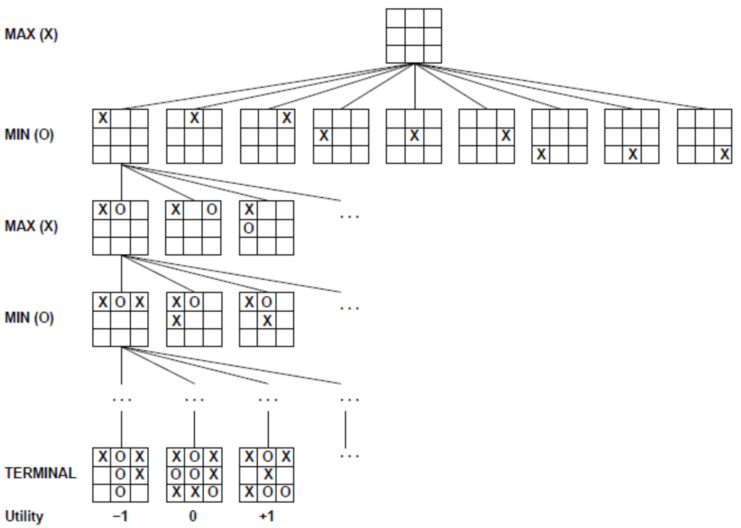
Game Tree

- Each level labelled with player to move
- Each level represents a ***ply***
  - Half a turn
- Represents what happens with competing agents

Introducing MIN and MAX

- MIN and MAX are two players:
  - MAX wants to win (maximise utility)
  - MIN wants MAX to lose (minimise utility for MAX)
  - MIN is the Opponent
- Both players will play to the best of their ability
  - MAX wants a strategy for maximising utility assuming MIN will do best to minimise MAX's utility
  - Consider ***minimax*** value of each node

Example Game Tree



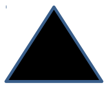
Minimax Value

- Utility for MAX of being in that state assuming both players play optimally to the end of the game
- Formally:

$$\text{MinimaxValue}(n) = \begin{cases} \text{Utility}(n) & \text{Terminal} \\ \max_{s \in \text{Successors}(n)} \text{MinimaxValue}(s) & \text{MAX} \\ \min_{s \in \text{Successors}(n)} \text{MinimaxValue}(s) & \text{MIN} \end{cases}$$

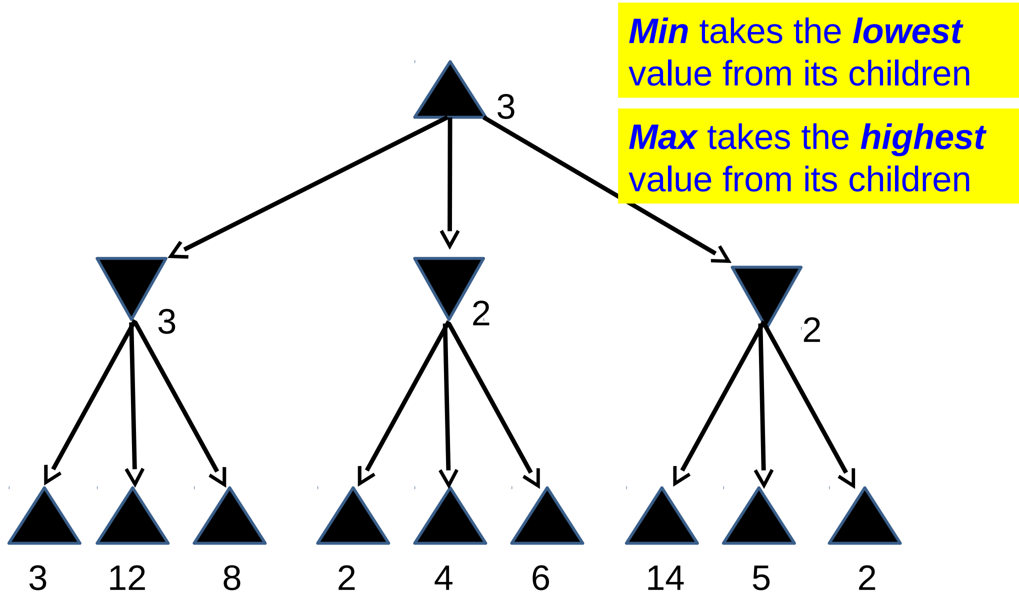
Minimax Algorithm

- Calculate minimaxValue of each node recursively
- Depth-first exploration of tree
- Game tree as *minimax tree*
- *Max Node:*



- *Min Node*

Minimax Tree



Exercise

Properties of Minimax

- Complete, if tree is finite
- Optimal, against an optimal opponent. Otherwise??
  - No. e.g. expected utility against random player
- Time complexity: b^m
- Space complexity: bm (depth-first exploration)
  - For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games
  - Infeasible – so typically set a limit on look ahead. Can still use minimax, but the terminal node is deeper on every move, so there can be surprises. No longer optimal
- But do we need to explore every path?

Pruning

- Basic idea:

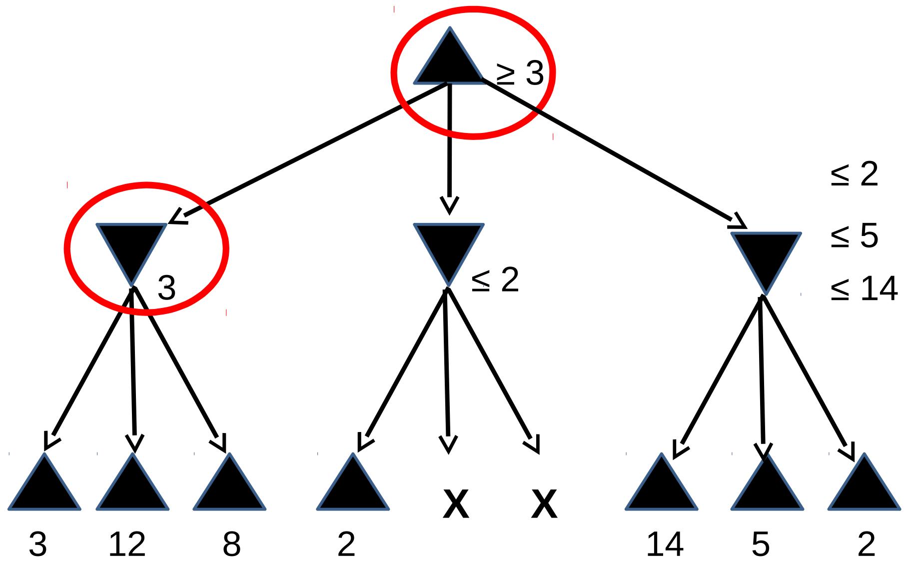
  If you know half-way through a calculation that it will succeed or fail, then there is no point in doing the rest of it

- For example, in Java it is clear that when evaluating statements like

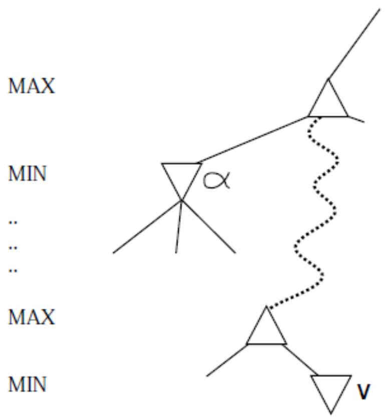
  if ((A > 4) || (B < 0))

- If A is 5 we do not have to check on B!

Alpha-Beta Pruning



Why is it called alpha-beta?



α is the best value (to MAX) found so far off the current path
If V is worse than α, MAX will avoid it $\Rightarrow$ prune that branch
Define β similarly for MIN

The Alpha-Beta Algorithm

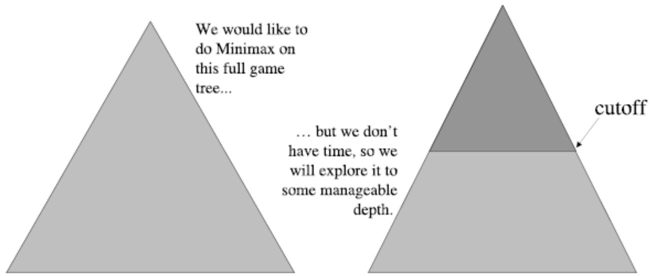
- alpha (α) is value of best (highest value) choice for MAX
- beta (β) is value of best (lowest value) choice for MIN
- If at a MIN node and value $\leq \alpha$, stop looking, because MAX node will ignore this choice
- If at a MAX node and value $\geq \beta$ beta, stop looking because MIN node will ignore this choice

Properties of Alpha-Beta

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering" time complexity $b^{m/2}$ and so doubles solvable depth
- A simple example of the value of reasoning about which computations are relevant (a form of *meta-reasoning*)
- Unfortunately, 35^{50} is still impossible, so chess not completely soluble

Cutoffs and Heuristics

- Cut off search according to some cutoff test
  - Simplest is a depth limit
- Problem: payoffs are defined only at terminal states
- Solution: Evaluate the pre-terminal leaf states using *heuristic evaluation function* rather than using the actual payoff function



Cutoff Value

- To handle the cutoff, in minimax or alpha-beta search we can make an alteration by making use of a cutoff value

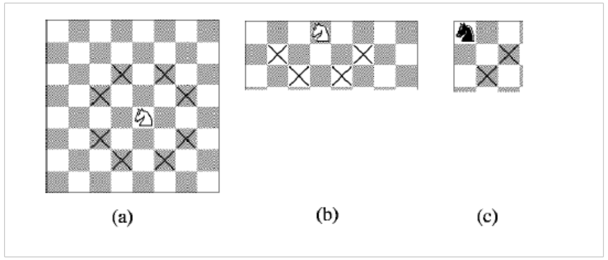
- *MinimaxCutoff* is identical to *MinimaxValue* except
  1. *Terminal* test is replaced by *Cutoff* test, which indicates when to apply the evaluation function
  2. *Utility* is replaced by *Evaluation* function, which estimates the position's utility

Example: Chess (I)

- Assume MAX is white
- Assume each piece has the following material value:
  - pawn = 1
  - knight = 3
  - bishop = 3
  - rook = 5
  - queen = 9
- let *w* = sum of the value of white pieces
- let *b* = sum of the value of black pieces

$$\text{Evaluation}(n) = \frac{w - b}{w + b}$$

Example: Chess (II)



- The previous evaluation function naively gave the same weight to a piece regardless of its position on the board…
  - Let X_i be the number of squares the *i*-th piece attacks
  - Evaluation(*n*) = $\text{piece}_1\text{value} * X_1 + \text{piece}_2\text{value} * X_2 + \ldots$

Example: Chess (III)

- Heuristics based on database search
  - Statistics of wins in the position under consideration
  - Database defining perfect play for all positions involving *X* or fewer pieces on the board (endgames)
  - Openings are extensively analysed, so can play the first few moves “from the book”

Deterministic Games in Practice

- Draughts: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions

- Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, used very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply

Deterministic Games in Practice

- Othello: human champions refuse to compete against computers, who are too good

- Go: a challenging game for AI (*b* > 300) so progress much slower with computers. AlphaGo was a recent breakthrough

See more at: University of Alberta GAMES Group

Summary

- Games have been an AI topic since the beginning. They illustrate several important features of AI:
  - perfection is unattainable so must approximate
  - good idea to think about what to think about
  - uncertainty constrains the assignment of values to states
  - optimal decisions depend on information state, not real state

- Next lecture:
  - We have now finished with the topic of search so we will move on to *knowledge representation*