COMP3411/9414/9814: Artificial Intelligence

Week 5: Games

Russell & Norvig, Chapter 5.

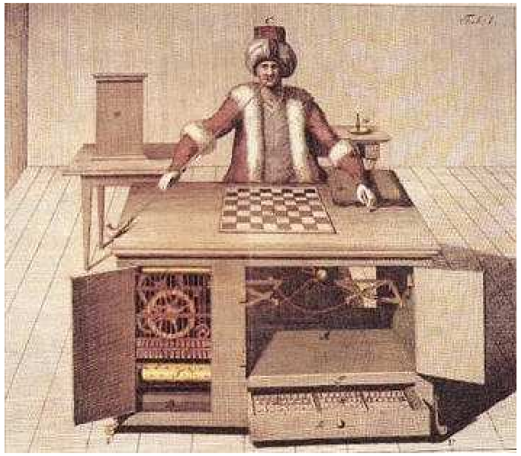
Outline

- origins
- motivation
- minimax search
- resource limits and heuristic evaluation
- α-β pruning
- stochastic games
- partially observable games
- continuous, embodied games

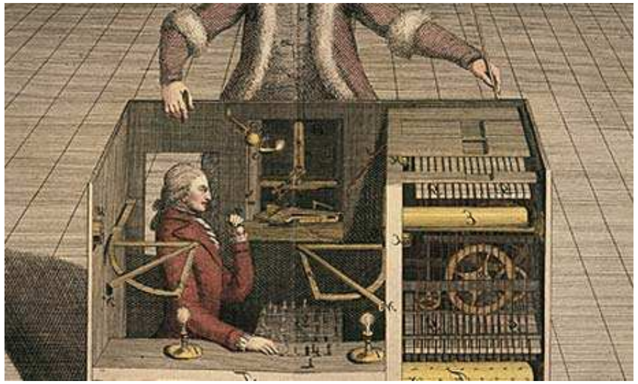
Origins

- 1769 Wolfgang von Kempelen (Mechanical Turk)
- 1846 Charles Babbage & Ada Lovelace (tic-tac-toe)
- 1952 Alan Turing (Chess algorithm)
- 1959 Arthur Samuel (Checkers)
- 1961 Donald Michie (MENACE machine learner)

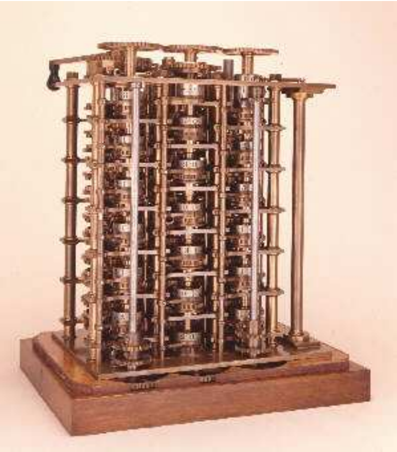
Mechanical Turk



Mechanical Turk



Charles Babbage Difference Engine



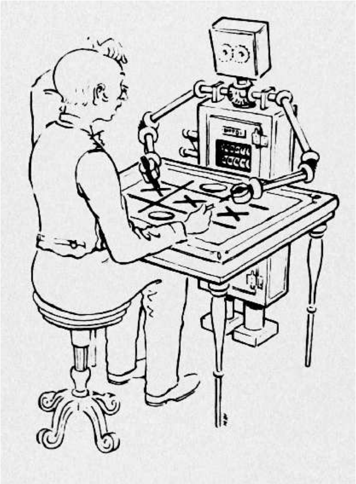
Funding Problems

"What shall we do to get rid of Mr. Babbage and his calculating machine?" (Prime Minister Robert Peel, 1842)

Ada Lovelace

"For the machine is not a thinking being, but simply an automation which acts according to the laws imposed upon it." (Ada Lovelace, 1843)

Babbage & Lovelace tic-tac-toe machine



Types of Games

- Discrete Games
  - fully observable, deterministic (chess, checkers, go, othello)
  - fully observable, stochastic (backgammon, monopoly)
  - partially observable (bridge, poker, scrabble)
- Continuous, embodied games
  - robocup soccer, pool (snooker)

Key Ideas

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952-57)
- Pruning to allow deeper search (McCarthy, 1956)

Why Games ?

- "Unpredictable" opponent ⇒ solution is a strategy
  - must respond to every possible opponent reply
- Time limits ⇒ must rely on approximation
  - tradeoff between speed and accuracy
- Games have been a key driver of new techniques in CS and AI

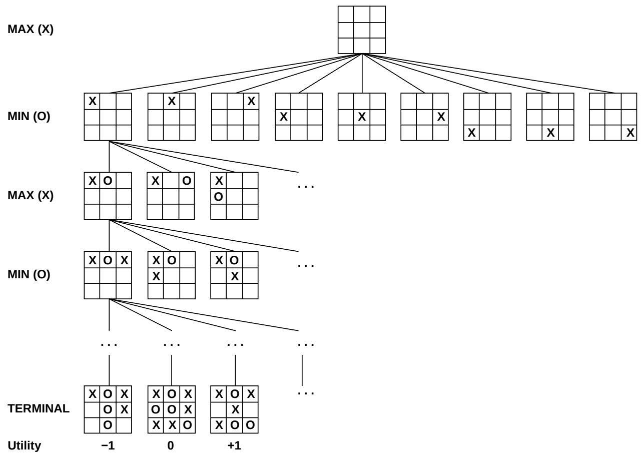
Samuel's Checkers Program

> "Elaborate table-lookup procedures, fast sorting and searching procedures, and a variety of new programming tricks were developed..."

Samuel's 1959 paper contains groundbreaking ideas in these areas:

- hash tables
- data compression
- parameter tuning via machine learning

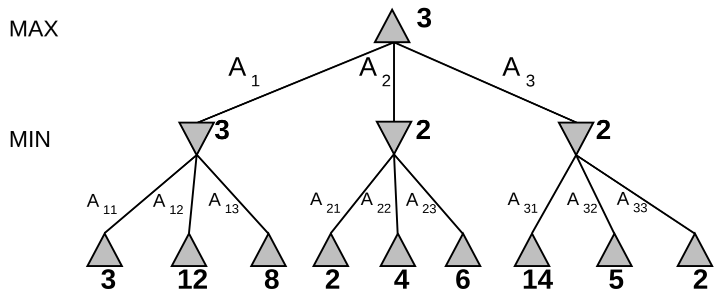
Game Tree (2-player, deterministic)



Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value = best achievable payoff against best play



Minimax algorithm

```
function minimax( node, depth )
    if node is a terminal node or depth = 0
        return heuristic value of node
    if we are to play at node
        let α = −∞
        foreach child of node
            let α = max( α, minimax( child, depth-1 ))
        return α
    else // opponent is to play at node
        let β = +∞
        foreach child of node
            let β = min( β, minimax( child, depth-1 ))
        return β
```

Minimax and Negamax

The above formulation of Minimax assumes that all nodes are evaluated with respect to a *fixed player* (e.g. White in Chess).

If we instead assume that each node is evaluated with respect to *the player whose turn it is to move*, we get a simpler formulation known as Negamax.

Negamax formulation of Minimax

```
function negamax( node, depth )
    if node is terminal or depth = 0
        return heuristic value of node
        // from perspective of player whose turn it is to move
    let α = −∞
    foreach child of node
        let α = max( α, -negamax( child, depth-1 ))
    return α
```


Properties of Minimax

- Complete?
- Optimal?
- Time complexity?
- Space complexity?

Reducing the Search Effort

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games $\Rightarrow$ exact solution completely infeasible

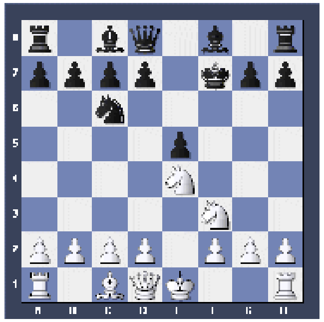
Two ways to make the search feasible:

- don't search to final position; use heuristic evaluation at the leaves
- α-β pruning

Heuristic Evaluation for Chess

- material
  - Queen = 9, Rook = 5, Knight = Bishop = 3, Pawn = 1
- position
  - some (fractional) score for a particular piece on a particular square
- interaction
  - some (fractional) score for one piece attacking another piece, etc.
- KnightCap used 2000 different features, but evaluation is rapid because very few features are non-zero for any particular board state (e.g. Queen can only be on one of the 64 squares at a time)
- the value of individual features can be determined by reinforcement learning

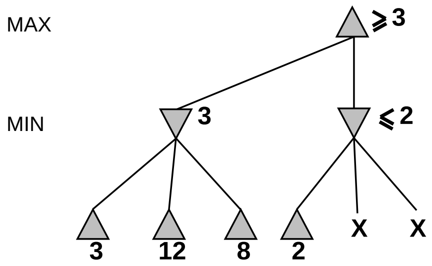
Pruning – Motivation



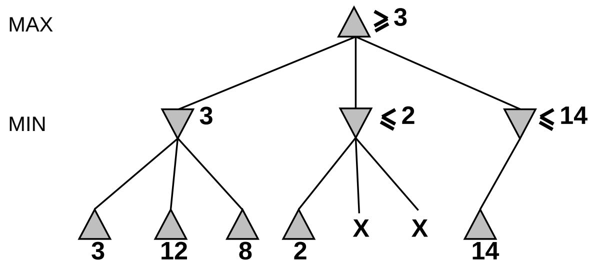
Q1: Why would "Queen to G5" be a bad move for Black?
Q2: How many White "replies" did you need to consider in answering?

Once we have seen one reply scary enough to convince us the move is really bad, we can abandon this move and continue searching elsewhere.

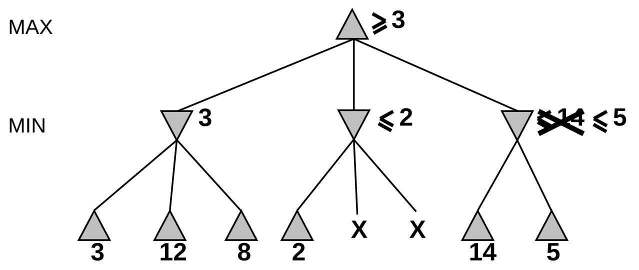
α-β pruning example



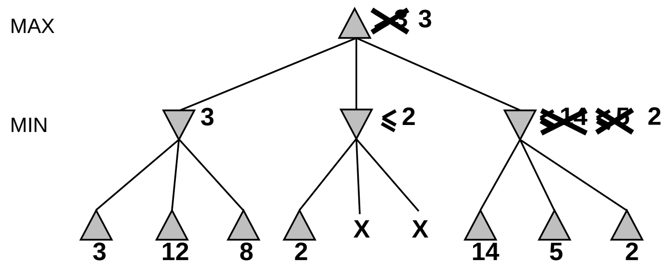
α-β pruning example



α-β pruning example



α-β pruning example



α-β search algorithm

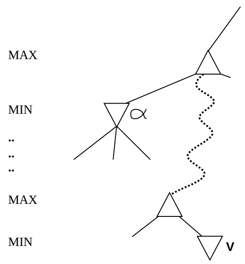
```
function alphabeta( node, depth, α, β )
    if node is terminal or depth = 0 { return heuristic value of node }
    if we are to play at node
        foreach child of node
            let α = max( α, alphabeta( child, depth-1, α, β ))
            if α ≥ β { return α }
        return α
    else // opponent is to play at node
        foreach child of node
            let β = min( β, alphabeta( child, depth-1, α, β ))
            if β ≤ α { return β }
        return β
```


Negamax formulation of α-β search

```
function minimax( node, depth )
    return alphabeta( node, depth, −∞, ∞ )

function alphabeta( node, depth, α, β )
    if node is terminal or depth = 0
        return heuristic value of node
        // from perspective of player whose turn it is to move
    foreach child of node
        let α = max( α, -alphabeta( child, depth-1, -β, -α ))
        if α ≥ β
            return α
    return α
```

Why is it called α-β?



α is the best value <u>for us</u> found so far, off the current path
β is the best value <u>for opponent</u> found so far, off the current path

If we find a move whose value exceeds α, pass this new value up the tree.

If the current node value exceeds β, it is "too good to be true", so we "prune off" the remaining children.

Properties of α-β

α-β pruning is guaranteed to give the same result as minimax, but speeds up the computation substantially

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity = $O(b^{m/2})$
To prove that a "bad" move is bad, we only need to consider one (good) reply.
But to prove that a "good" move is good, we need to consider all replies.

This means α-β can search twice as deep as plain minimax.
An increase in search depth from 6 to 12 could change a very weak player into a quite strong one.

Chess

Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997.

Traditionally, computers played well in the opening (using a database) and in the endgame (by deep search) but humans could beat them in the middle game by "opening up" the board to increase the branching factor. Kasparov tried this, but because of its speed Deep Blue remained strong.

Some experts believe Kasparov should have been able to defeat Deep Blue in 1997 if he hadn't "lost his nerve". However, chess programs stronger than Deep Blue are now running on standard PCs and could definitely defeat the strongest humans.

Modern chess programs rely on quiescent search, transposition tables and pruning heuristics.

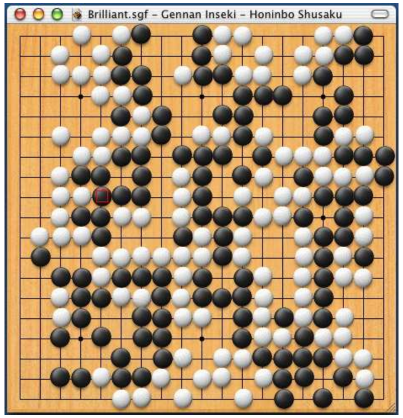
Checkers

Chinook failed to defeat human world champion Marion Tinsley prior to his death in 1994, but has beaten all subsequent human champions.

Chinook 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. This database has since been expanded to include all positions with 10 or fewer pieces (38 trillion positions).

In 2007, Jonathan Shaeffer released a new version of Chinook and published a proof that it will never lose. His proof method fills out the game tree incrementally, ignoring branches which are likely to be pruned. After many months of computation, it eventually converges to a skeleton of the real (pruned) tree which is comprehensive enough to complete the proof.

Go



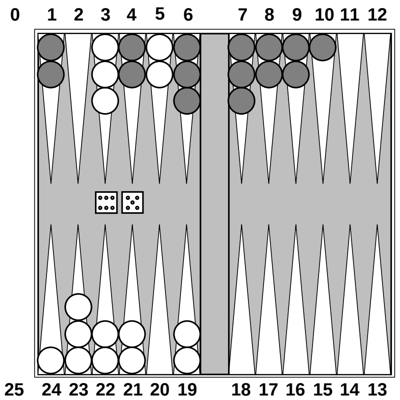
Go

The branching factor for Go is greater than 300, and static board evaluation is difficult. Traditional Go programs broke the board into regions and used pattern knowledge to explore each region.

Since 2006, new “Monte Carlo” players have been developed using UCB search. A tree is built up stochastically. After a small number of moves, the rest of the game is played out randomly, using fast pattern matching to give preference to “urgent” moves.

In March 2016, AlphaGo defeated the human Go champion Lee Sedol in a 4-1 match. AlphaGo uses MCTS, with deep learning neural networks for move selection and board evaluation. The networks are trained initially on a database of thousands of human championship Go games, and then refined with millions of games of self-play.

Stochastic games: backgammon



Stochastic games in general

In stochastic games, chance introduced by dice, card-shuffling, etc.

Expectimax is an adaptation of Minimax which also handles chance nodes.

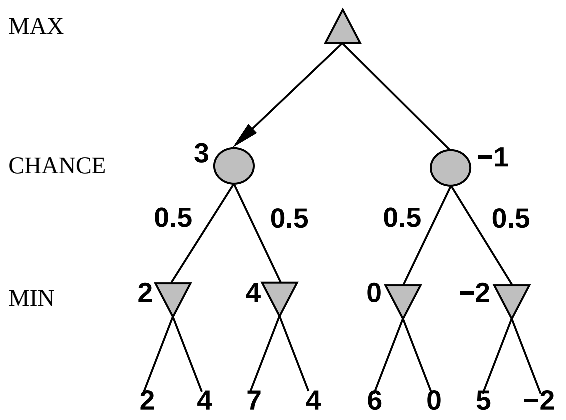
...

    if node is a chance node
        return average of values of successor nodes

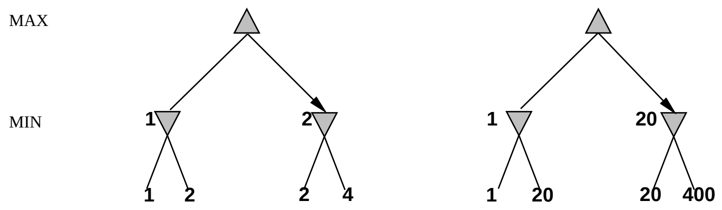
...

Adaptations of α-β pruning are possible, provided the evaluation is bounded.

Expectimax algorithm



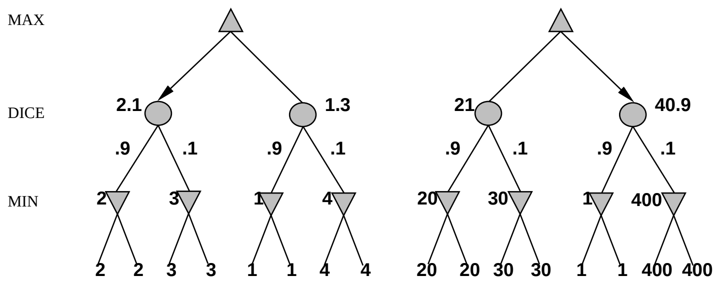
For Minimax, Exact values don't matter



Move choice is preserved under any monotonic transformation of EVAL.

Only the order matters:
payoff in deterministic games acts as an ordinal utility function.

For Expectimax, Exact values DO matter



Move choice only preserved by positive linear transformation of EVAL

Hence EVAL should be proportional to the expected payoff.

Partially Observable games

Card games are partially observable, because (some of) the opponents' cards are unknown.

This makes the problem very difficult, because some information is known to one player but not to another.

Typically we can calculate a probability for each possible deal.

Idea: compute the minimax value of each action in each deal,
then choose the action with highest expected value over all deals.

GIB, a strong and well-known bridge program, approximates this idea by

1) generating 100 deals consistent with bidding information
2) picking the action that wins most tricks on average

Infinite Mario

Currently best solution uses A*Search, after reverse engineering the world model.

Pacman

Combines path planning, low-level control, reasoning under uncertainty and (for ghosts) multi-agent coordination.

Robocup Soccer

Deep Green pool playing robot

Deep Green pool playing robot

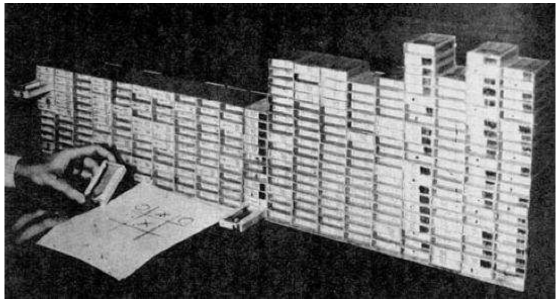
Low level technical issues

- undistortion of overhead camera image
- ball appears “egg-shaped”, need to find centre accurately

High level strategy

- easy to sink current ball
- more complicated to “set up” for the next ball
- competition using physical simulator

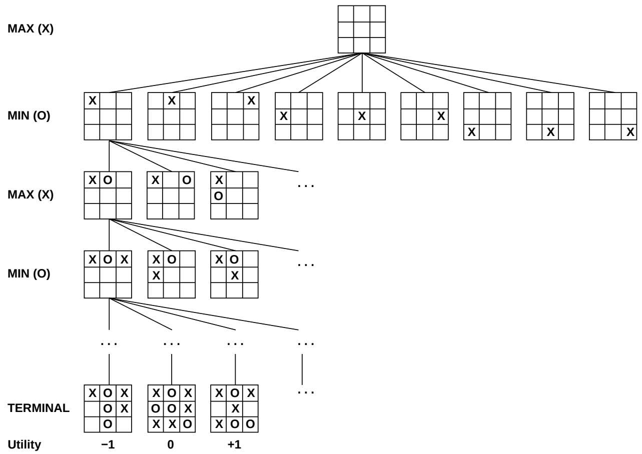
MENACE



Machine Educable Noughts And Crosses Engine
Donald Michie, 1961

MENACE

Game Tree (2-player, deterministic)



Summary

- games are fun to work on!
- games continue to be a driver of new technology
- tradeoff between speed and accuracy
- probabilistic reasoning
- force us to build “whole systems” – chain is as strong as its weakest link

References

Tom Standage, 2002. The Mechanical Turk, Penguin Books.

Arthur Samuel, 1959. Some studies in machine learning using the game of checkers, IBM Journal on Research and Development, pages 210-229.

Chinook: www.cs.ualberta.ca/˜chinook

Robocup: www.robocup.org

[look for Infinite Mario and Deep Green on youtube]