Adversarial Search

"Simple games" e.g. chess

- Deterministic
- Two player
- Turn taking
- Perfect information ≡ fully observable
- Zero sum
- Action = a move
- A state contains:
  - Complete representation of the "board"
  - Whose turn it is to move next
- A terminal state is where the game is over
- Utility function measures value of a terminal state
  - But useful to also have one for intermediate states
- State graph or search tree may be enormous, chess > 10^{40}
  - But not always - noughts and crosses 362880

Things to have

- S_0 - the initial state
- ToMove(s) - whose go is it?
- Actions(s) - set of possible actions
- Result(s, a) - state an action will result in
- IsTerminal(s) - is the game over?
- Utility(s, p) - goodness from p's point of view

Names of players taken from one particular player's point of view

- "Max" is that player, he/she is trying to maximise his/her final score
- "Min" is the other player, he/she is trying to minimise Max's score
- Max's strategy must be conditional - depends on Min's moves

Minimax search

- Minimax value of a state:
  - Max's value of being in that state, assuming both play perfectly
  - Terminal node: just its utility function value
  - Internal node:
    - Max's go: maximum value for all children
    - Min's go: minimum value for all children
  - Often tree is too deep and some estimation is substituted instead
- Simple recursive algorithm, two functions - minmove and maxmove
- More than two players, same idea works
  - Utility function returns list of values, one from each player's view
  - But (temporary) alliances

Alpha-beta pruning

- As each subtree is explored, keep range of possible values, init. -∞ to +∞
  - e.g. max(min(bigs, …), min(small, …), min(others, …))
- Alpha = looking for best, beta = looking for worst

- Effectiveness depends on order of subtrees: what if small didn't come first?
- Again, estimate when the tree is too deep

Monte-Carlo Tree Search

- Too branchy or no good evaluation function
- Playout = simulation of complete game from current state
- Try a lot of random playouts, take average final score
- How many? impose a time limit
- Tree - each node has #wins, #playouts so far
  - Select the node with the best ratio?
  - Select the node that has been least explored?
  - After playout from node n, back-propagate to all ancestors
- Example utility $UCB1(n) = U(n)/ N(n) + C \times \sqrt{(\log(N(Parent(n)) / N(n))}$
  - U(n) = total utility of playouts from n
  - N(n) = number of playouts from n
  - U(n) / N(n) is the exploitation term, average utility
  - √ ... is the exploration term, higher for less-explored
  - C is a constant to balance the two, often $\sqrt{2}$

Stochastic games

- There is an element of chance
- Don't know what opponent's possible moves will be
- Tree must include Chance Nodes
  - Each arc labelled with outcome (e.g. dice roll) and probability
- Expectimax - just like minimax but uses expected value

Go

- Branching factor initially 361
- No known good evaluation/heuristic function
- Monte-Carlo search:
    - Do a bunch of whole-game simulations from current state
        - Random moves
        - Use expert-guided playout policies
    - Take average of all their final scores

Stochastic means randomness is involved, e.g. dice, shuffled cards, etc