AI

- Searching
  - Blind search – keep moving randomly till reach goal, independent of the domain depth first, breadth-first, iterative-deepening

    Heuristic – utilities domain specific features, best first A* i.e. Finding a word in a dictionary we know it starts with say Z so look there first

    Local – if only finding goal state matters i.e. Situations, hill-climbing, simulate annealing, genetic algorithms

Blind Search

- Tree Search – just go down the tree till hit sumthing good
  - Strategy's – Order of expansion
    - completeness – always find a solution if its there
    - time complexity – number of nodes generated / expanded how long will it take
    - space complexity – maximum number of nodes in memory at any one time
    - optimality – does it always find the least time cost solution ie fastest

    time and space complexity measured by
      - b = maximum branching factor
      - d = depth of best solution
      - m = maximum tree depth

    Breadth first
      - work along each level of the tree
      - expand shallowest unexpanded node
      - fringe as FIFO queue

      has to store all nodes in memory

    Depth first
      - work down to the bottom of each branch
      - expand deepest unexpanded nodes
      - fringe as LIFO queue

      Not optimal can get stuck on infinite paths
      Less memory than Breadth first

    Iterative Deepening
      - effectively breadth first but take advantage of depth search by doing a depth search to iteratively increasing depths

    Open and Closed nodes
      - Remember visited nodes to stop infinite loops

Heuristic Search

Use other known knowledge to search for solutions

Best first search – pick the seemingly best node among the search nodes based on an evaluation function that works for the data.

i.e. Search inwards towards a goal
i.e. With a compass

A* search

cost (s->n) called g(n)
cost (n->g) called h(n)

g(n) is path cost to n
h(n) is estimate of least cost path from n to g
f(n) = g(n) + h(n) estimated cost of cheapest solutions
among search nodes select node where f(n) is lowest

depending on h(n) this can be complete and optimal
if h(n) =< true cost(n->g)

for A* if h(n) is admissible then its optimal

complete, unless infinite many nodes f=<f(g)
time, exponential
space, needs to store all nodes
optimal, yes

Iterative Improvement
i.e. The chess game

Hill Climbing
trying to find the shallowest part of the sea using a probe
step along if its increasing keep going, if it starts going down go back and try and find highest

can miss highest and home in on a smaller point

Simulated Annealing
escape from local maximums by allowing some bad moves, have a probability that you can move

used a lot as it allows solutions to hard problems to be effectively guessed / worked towards

$P(x) \infty \exp(-E(x) / T)$

T slowly reaches so that it finds goal

Genetic algorithms
Pick 2 parents
apply mutation bit flip with probability
replace population with offspring

1 point crossover just cut and join the 2 strings somewhere

- selection
    - give individuals chances and place these on a roulette wheel then spin the wheel n times to get n individuals

- works on local search in uncertain conditions

MiniMax search

- Max is player A, aims to pick best moves
- Min is player B, aims to pick best moves to minimise A's advantage

- The faster the computer the deeper it can look

- Cant look forever, if opponent looks 1 more than us then we have no chance, especially if the opponent notices this.

- Optimise
    - take into account board size

    - Alpha-Beta pruning
        - Alpha lower bound on node evaluations (worst we can do)
            - Associate with max nodes
            - Never decreases
        - Beta represents the upper bound
            - Associate with min nodes
            - Never increases