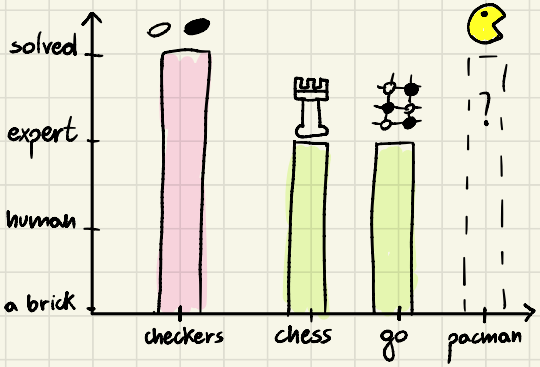


type of games

- Determenistic or stocastic?
- number of players?
- zero sum?
- perfect information? (knowing everything)

we want algorithms for calculating a strategy (policy) with recoemends a move for every state

formalizations, many possible, one is:

- States: S, starts at S_0
- Players: $P=\{1,...n\}$ they usually take turns
- Actions: A may depend on player/state
- Transition function: $S \times A \rightarrow S$
- Terminal test: $S \rightarrow \{t,f\}$
- Terminal utilities: $S \times P \rightarrow R$

Solution for a player is a policy: $S \rightarrow A$

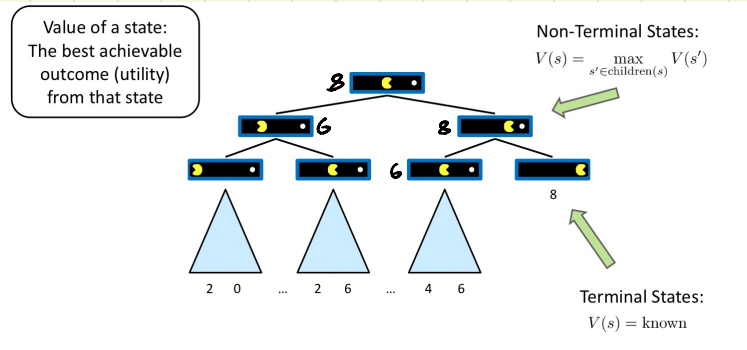
Zero-Sum games

- agents have opposite utilities (they fight over resources)
- think of a single value that one maximizes and the other minimizes
- adversial, competition

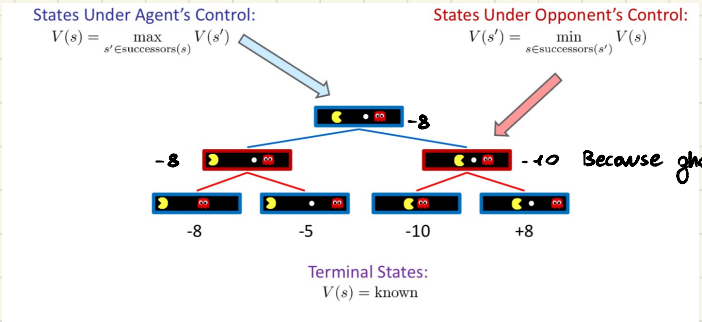
Adversial search

value of state: the best achievable outcome (utility) from that state

single agent



Dual agent, pacman tries to maximise, ghost to minimize



Because ghost prevents pacman from going to the +8

Minimax

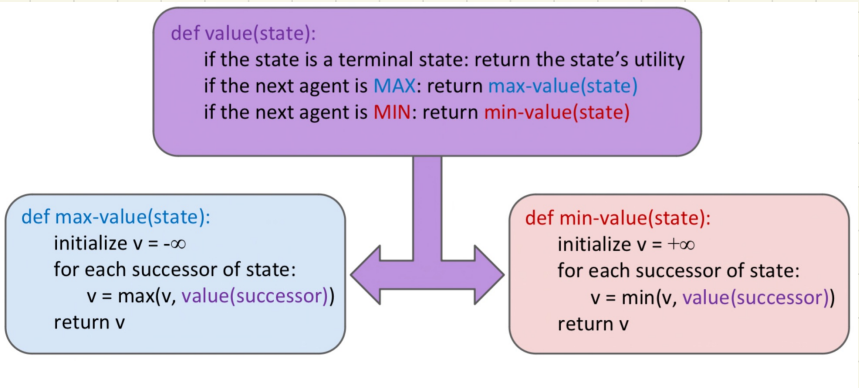
Deterministic, zero-sum games

one player max one min

minimax search:

- state space search tree
- players alternate turns
- compute each node's minimax value:

best achievable utility against a rational optimal adversary



properties:

optimal against a perfect player, otherwise? next lecture

efficient?: like DFS time $O(b^m)$, space $O(bm)$

ex. chess $b \approx 35$ $m = 100$

exact solution unfeasable

but do we need to explore the whole tree? ⟹ game tree pruning

Alpha-Beta pruning (min version)

- computing min value at some node n
- looping through n's children
- n's estimate of the children's min is dropping
- who cares about n's value? MAX
- α = best value that MAX can get at any choice point along the current path from the root
- if n becomes worse than α, MAX will avoid it. So we can stop considering n's children

MAX version is symmetric

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

important not strict

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

- this method has no effect on minimax value computed for the root
- values of intermidiate nodes may be wrong
  children of root may have the wrong value

good child ordering improves effectiveness of pruning

perfect ordering:

time comp. drops to $O(b^{m/2})$

doubles solvable depth

example of metareasoning -> computing what to compute

Resource limits

in realistic games cannot search to leaves

solution: depth-limited search

search only to a limited depth

replace terminal utilities with an evaluation function for non-terminal positions

- no more optimality guarantee
- more plies makes big difference
- use iterative deepening for time left (anytime algorithm)

Evaluations functions

score non-terminals in depht-limited search

ideal: returns the actual minimax value of the position

practice: weighted linear sum of features

Evaluation functions are never perfect

The deeper in the tree the eval f. is buried, the less quality of evaluation f. matters

You can use eval f. to improve the ordering in Alpha-Beta

the amount of pruning depends on expansion ordering

- value at a min-node will only keep going down
- once value of min-node lower than better option for max above, prune
- Hence: if eval. funct. provides upper-bound on value at min-node, and upper-bound already lower than better option for max above then can prune

Uncertainty and utilities

Uncertain outcomes controlled by chance, not an adversary

- why wouldn't we know what the result will be?
  - explicit randomness
  - unpredictable opponents
  - actions can fail
- values should now reflect average-case (expectimax) outcomes, not worst-case (minimax)

Expectimax Search: compute the average score under optimal play

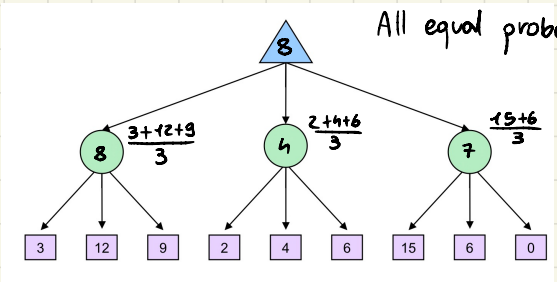
- max nodes as minimax
- chance nodes are like min nodes but the outcome is uncertain
- calculate the excepted utilities
  i.e. weighted average (expected) of children

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

example



Can't do pruning

Depht limited expectimax

estimate of true expectimax value (that would require a lot of computing)

Probabilities

random variable represents an event whose outcome is unknown

probability distribution is an assignment of weights to outcomes

- probabilities are always non-negative
- probabilities over all possible outcomes sum to 1
- probabilities may change as we get information

Expectations

The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes

What probabilities to use?

in expectimax search we have a probabilistic model of how the opponent (or environment) will behave in every state.

i.e.
- simple uniform distribution (roll die)
- sophisticated, require a lot of computation
- we have a chance node for any outcome of our control: opponent or environment
- model may say adversial actions are likely

for now we assume each chance node comes along with probabilities that specify the distribution over its outcomes.

Dangers of optimism and pessimism

assuming chance when the world is adversary

assuming the worst case when it's not likely

we can mix expectimax and minimax