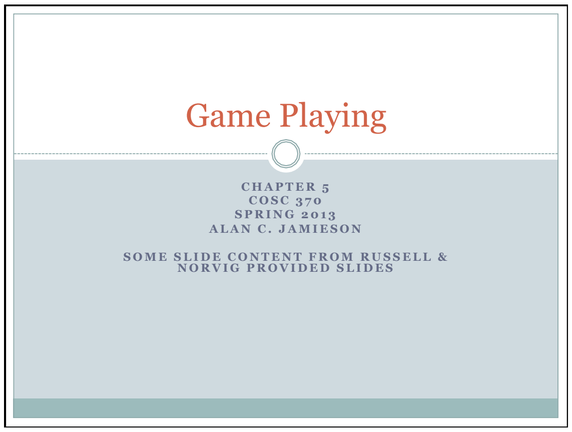
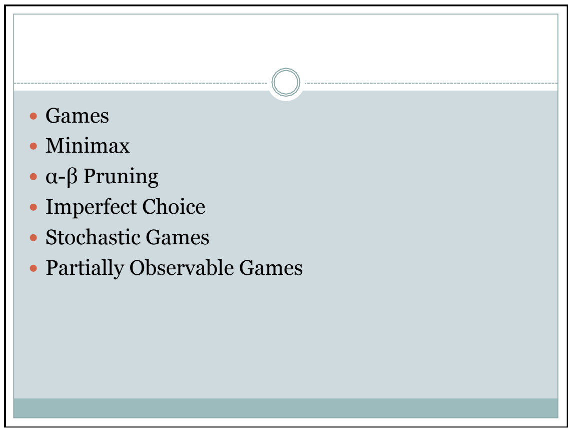


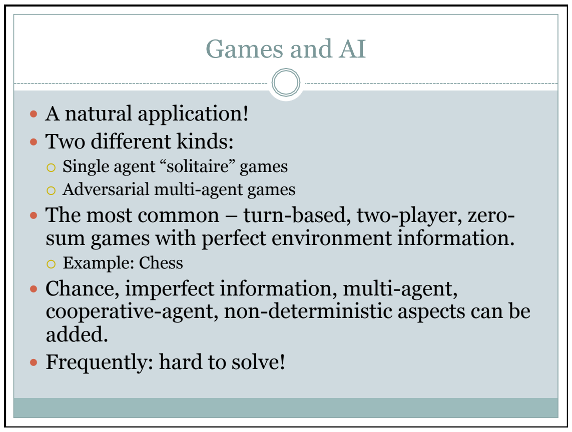
Game Playing

CHAPTER 5
COSC 370
SPRING 2013
ALAN C. JAMIESON

SOME SLIDE CONTENT FROM RUSSELL & NORVIG PROVIDED SLIDES

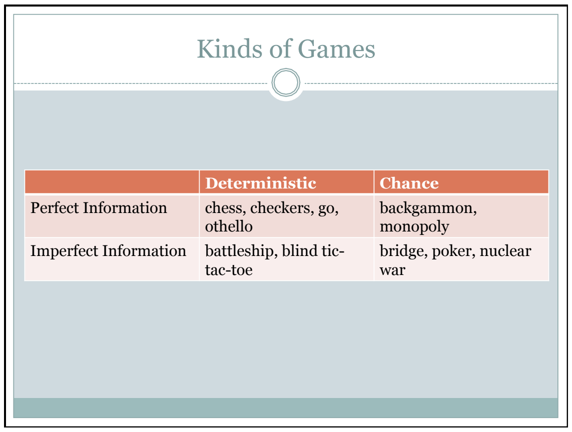


- Games
- Minimax
- α-β Pruning
- Imperfect Choice
- Stochastic Games
- Partially Observable Games



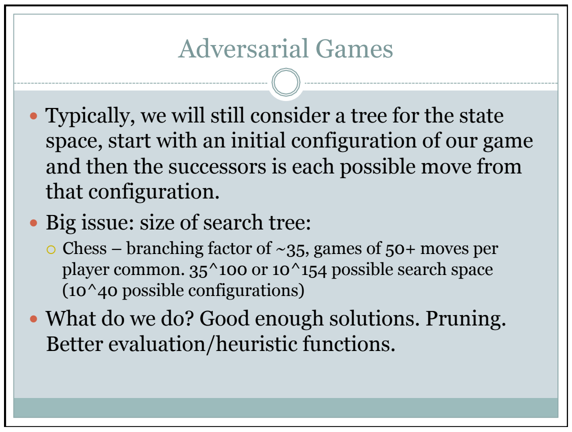
Games and AI

- A natural application!
- Two different kinds:
  - Single agent “solitaire” games
  - Adversarial multi-agent games
- The most common – turn-based, two-player, zero-sum games with perfect environment information.
  - Example: Chess
- Chance, imperfect information, multi-agent, cooperative-agent, non-deterministic aspects can be added.
- Frequently: hard to solve!



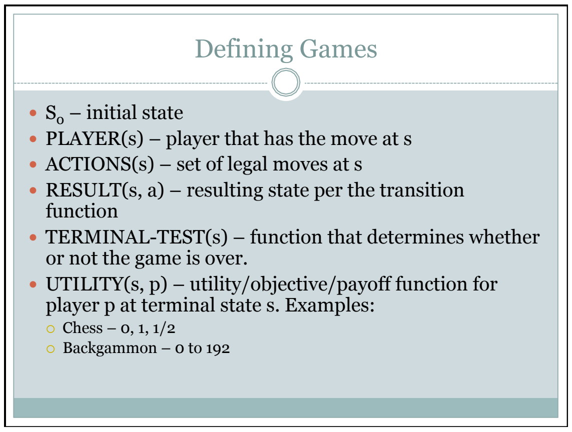
Kinds of Games

	Deterministic	Chance
Perfect Information	chess, checkers, go, othello	backgammon, monopoly
Imperfect Information	battleship, blind tic-tac-toe	bridge, poker, nuclear war



Adversarial Games

- Typically, we will still consider a tree for the state space, start with an initial configuration of our game and then the successors is each possible move from that configuration.
- Big issue: size of search tree:
  - Chess – branching factor of ~35, games of 50+ moves per player common. 35^100 or 10^154 possible search space (10^40 possible configurations)
- What do we do? Good enough solutions. Pruning. Better evaluation/heuristic functions.



Defining Games

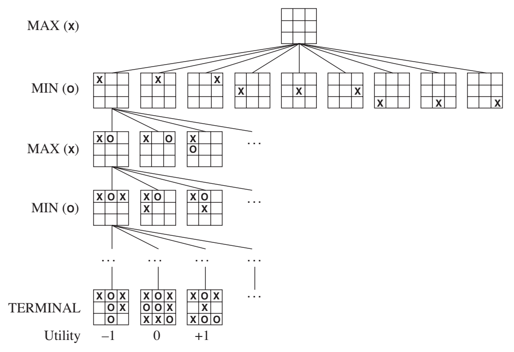
- S_0 – initial state
- PLAYER(s) – player that has the move at s
- ACTIONS(s) – set of legal moves at s
- RESULT(s, a) – resulting state per the transition function
- TERMINAL-TEST(s) – function that determines whether or not the game is over.
- UTILITY(s, p) – utility/objective/payoff function for player p at terminal state s. Examples:
  - Chess – 0, 1, 1/2
  - Backgammon – 0 to 192

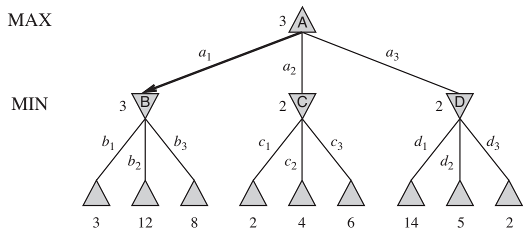
Minimax

- Label our players MAX and MIN. This represents the target utility value in reference to our first player.
- MAX – the first player wants to maximize his or her utility, the higher the better (traditionally).
- MIN – our second player wants to minimize the first player's utility with their move.
- Traditionally – expand all of our nodes then work backwards.
- We assume that our opponent will make optimal moves – minimax value represents best possible payoff against optimal opponent.

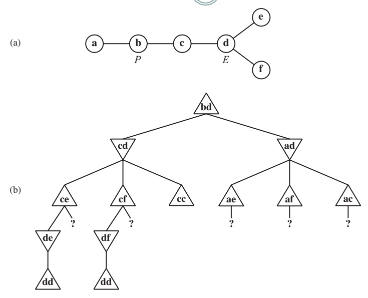
Minimax

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for a, s in SUCCESSORS(state) do $v \leftarrow$ MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow \infty$
    for a, s in SUCCESSORS(state) do $v \leftarrow$ MIN(v, MAX-VALUE(s))
    return v
```





Exercise 5.3



Properties of Minimax

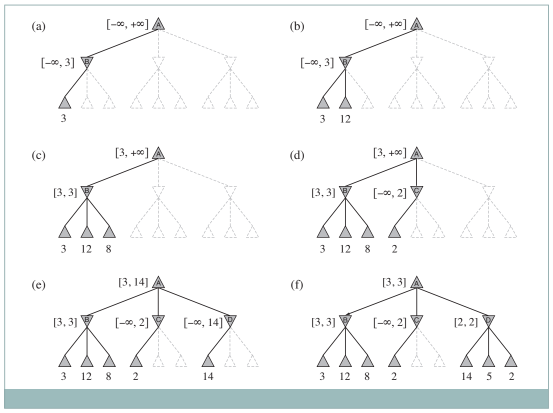
- Complete only if tree is finite.
- Optimal against an optimal opponent.
- Time complexity – exponential!
- Space complexity – linear!

α-β Pruning

- Trouble with Minimax – time! Exponential in the depth of the tree.
- How do we trim this? Pruning!
- Effectively cuts the time in half (still exponential).
- Pruning – elimination of subtrees/possible states without examining them due to some factor.
- Eliminate branches that cannot affect our final solution – still returns the same solution as minimax.

α-β Pruning

- General principal – consider a node n such that the player has a choice to moving to that node. If player has a better choice at that branch (m) or at any choice further up, n will never actually be reached!
- Basically the same properties as minimax.

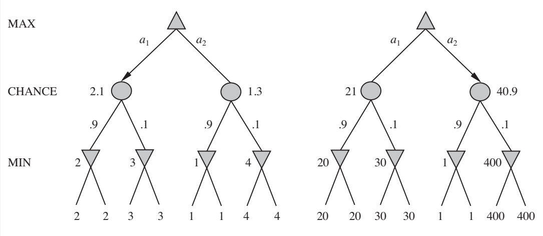


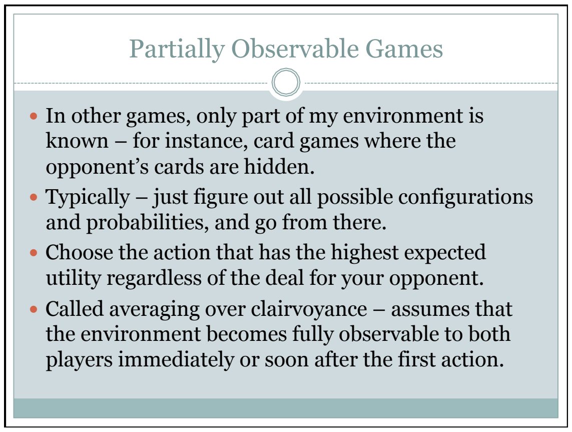
Dealing with Complexity

- Size is an issue (isn't it always)? How do we deal with it?
- Option 1 – cutoff test – use a heuristic to estimate the utility of a given move at the set maximum depth. If that heuristic meets a threshold (dependent on if that level is a min or a max) then keep it, otherwise, discard.
- Option 2 – forward pruning – consider only a selection of n best moves, prune all others.
- Neither option is guaranteed to be optimal!

Games of Chance

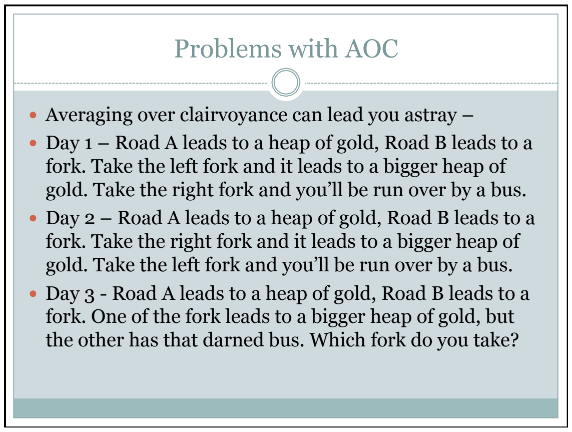
- Frequently, our games will include some element of chance (commonly, dice).
- We can still use minimax/α-β pruning in this case, but a small adjustment is required.
- Between each max and min we will add a chance branch – this represents the roll that the player at that level could make, including the probabilities (for instance, with 2 die, 7 is the most common roll at ~17%).
- We can only calculate expected utility here!





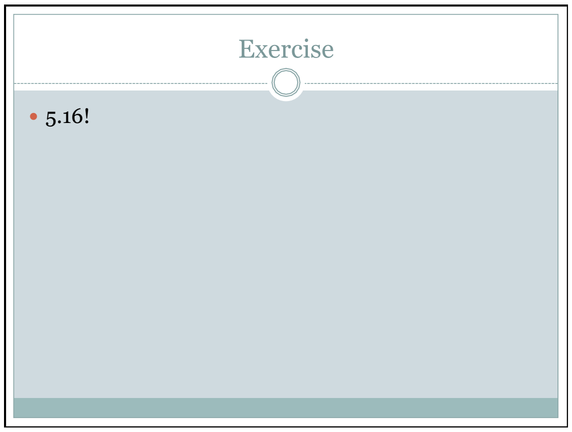
Partially Observable Games

- In other games, only part of my environment is known – for instance, card games where the opponent's cards are hidden.
- Typically – just figure out all possible configurations and probabilities, and go from there.
- Choose the action that has the highest expected utility regardless of the deal for your opponent.
- Called averaging over clairvoyance – assumes that the environment becomes fully observable to both players immediately or soon after the first action.



Problems with AOC

- Averaging over clairvoyance can lead you astray –
- Day 1 – Road A leads to a heap of gold, Road B leads to a fork. Take the left fork and it leads to a bigger heap of gold. Take the right fork and you'll be run over by a bus.
- Day 2 – Road A leads to a heap of gold, Road B leads to a fork. Take the right fork and it leads to a bigger heap of gold. Take the left fork and you'll be run over by a bus.
- Day 3 - Road A leads to a heap of gold, Road B leads to a fork. One of the fork leads to a bigger heap of gold, but the other has that darned bus. Which fork do you take?



Exercise

- 5.16!