

Adversarial Search

•Russell and Norvig, Chapter 5

Wednesday Tutorial

- Ekka holiday 17 August
- For this week, people signed on to the Wednesday tutorial should either attend another tutorial or submit online by 5pm on Thursday
- Every other week either submit online by 5pm on the Monday before the tutorial OR attend a tutorial and submit at the end of the tutorial

Assignment 1

- Up on the website sometime this week
- Will discuss the assignment during the lecture after the mid-semester exam

Mid Semester Exam

- 2 weeks time
- Normal lecture time, different place
- 2-3pm Tuesday 30 August 42-115
- Remember to bring a pencil and an eraser

Overview: aims

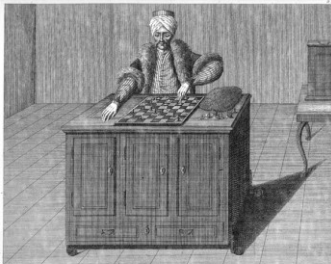
- understand the difficulties in search that arise as a consequence of agents acting in the environment
- understand the Minimax algorithm and its properties
- understand alpha-beta pruning and how it assists in adversarial search
- know how heuristic functions can be used for making imperfect decisions
- be able to (in principle) build your own computer vs. human chess program

Overview: topics

- Adversarial search problems
- The MiniMax algorithm
- Alpha-beta pruning
- Heuristics for adversarial search, making imperfect decisions

The Turk

The Chess player automaton by Baron von Kempelen



http://en.wikipedia.org/wiki/The_Turk

Deep Blue



http://www.thetech.org/robotics/universal/breakout_p11_ibm.html

AI and Game Playing

- Was Deep Blue (1997) a better Chess player than Garry Kasparov? (Won 3.5-2.5)
- Did Deep Blue possess Chess intelligence?
- Did Deep Blue have better Chess intelligence than Kasparov?

See <http://www.research.ibm.com/deepblue/>

Ratings:

1997: Kasparov (2806, FIDE), Deep Blue ?

2007: Kramnik (2769, FIDE), Rybka (2962, SSDF)

2009: Topalov (2813, FIDE), Deep Rybka 3 (3224, SSDF)

Games in AI

- Deterministic
- Turn-taking
 - Actions alternate
- Two-player
- Zero-sum
 - Utility functions are equal and opposite
- Perfect information
 - Fully observable environments

Games

- Multiplayer
 - Non-zero-sum
 - Stochastic
 - Imperfect information
 - Physical
 - ...
- For more information, read Chapter 17

Games and AI

- Chess (1950s – Konrad Zuse, Claude Shannon, Norbert Wiener, Alan Turing)
- Checkers
- Othello
- Backgammon
- Go

Chess



Photo from: commons.wikimedia.org

Checkers / Draughts

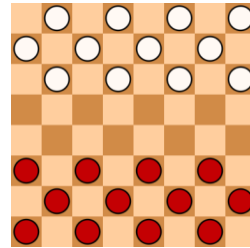


Image from: commons.wikimedia.org

Othello / Reversi

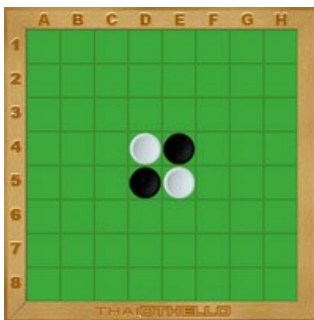


Image from: commons.wikimedia.org

Backgammon



Photo from: commons.wikimedia.org

Go



Photo from: commons.wikimedia.org

Adversarial search: Assumptions and aims

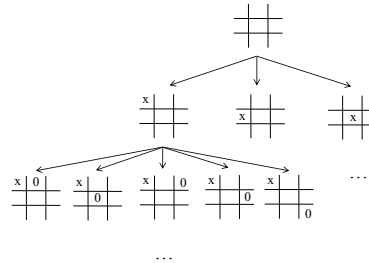
- Multiplayer
- Formal, well-defined problems, where perfect decisions are possible
- At any position in the game, the player wants to know what move to make
- Success depends on opponent's moves
- Search forward:
 - Exhaustive
 - Heuristic

Game search problem

- Initial state
 - Board position and player
- Successor function
 - Returns a list of (action, state) pairs, indicating a legal move and resulting state
- Terminal test
 - Determines when the game is over
- Utility function
 - Numeric value for a terminal state representing its utility for a given player.
 - In chess, the outcome is win, loss or draw with values +1, -1 or 0

Game Tree

- Defined by the initial state and the legal moves for each player



Terminal States

- Utility for player x

0 x x	0 x 0	0 x 0
x 0 x	x 0 x	x 0 x
0 0 x	x 0 x	0 x
1	0	-1

Optimal decisions in games

- Consider
 - Two players – MIN and MAX
 - MAX moves first
 - Take turns moving until the game is over
 - At end of game points are awarded to the winning player and penalties are given to the loser

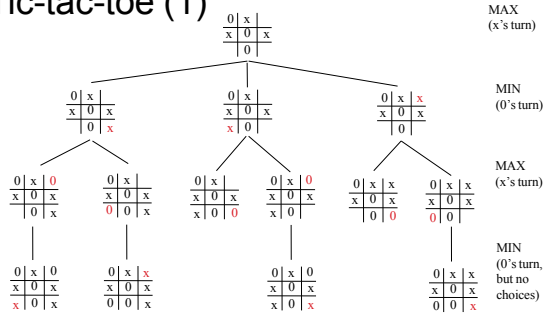
Strategies

- Optimal solution for MAX is a sequence of moves leading to a terminal state in which MAX wins
- However, MIN is also trying to win
- Strategy for MAX is to choose the path that leads to the best terminal state for MAX, given that MIN is trying to find the best terminal state for MIN

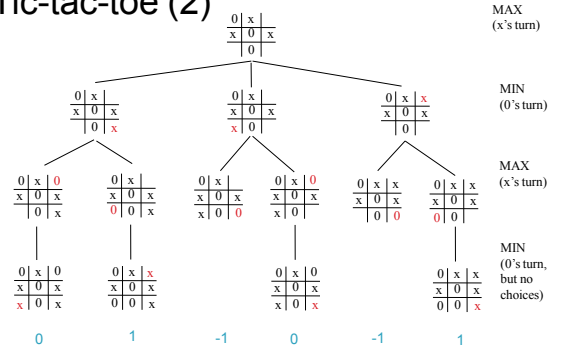
Optimal strategy and MiniMax

- MiniMax value of a node is the utility (for MAX) of being in a corresponding state from there until the end of the game
 - Assumes that both players play optimally
- MiniMax value of a terminal state is its utility
- MAX will prefer to move to a state of maximum value
- MIN will prefer to move to a state of minimum value (from MAX's point of view)

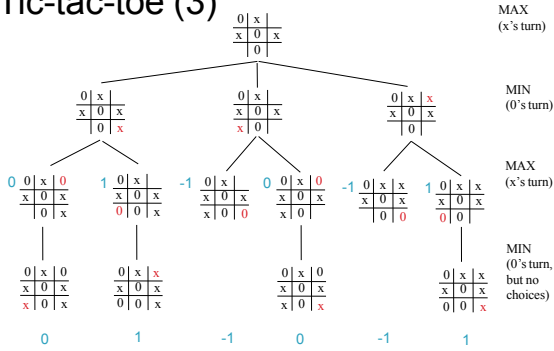
Tic-tac-toe (1)



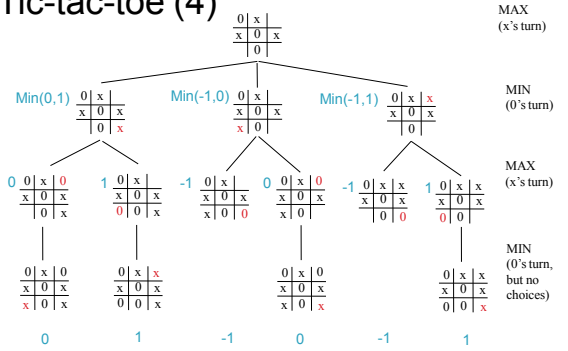
Tic-tac-toe (2)



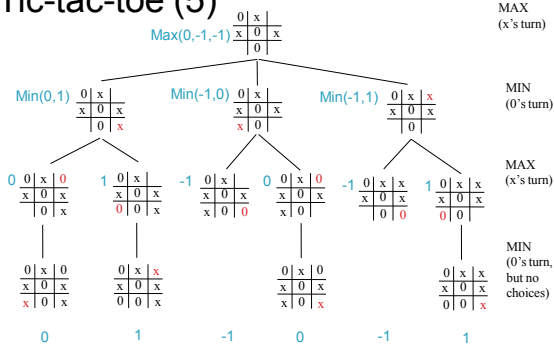
Tic-tac-toe (3)



Tic-tac-toe (4)



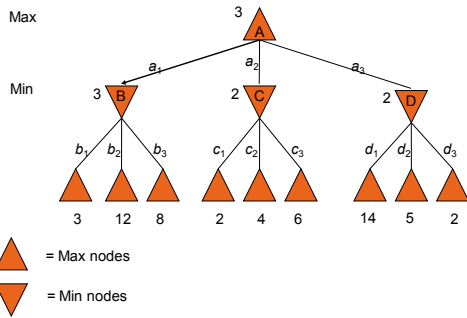
Tic-tac-toe (5)



MiniMax value =

- Utility if n is a terminal state
- Max of successors MiniMax values if n is a MAX node
- Min of successors MiniMax values if n is a MIN node

A 2-ply game tree



MiniMax

```

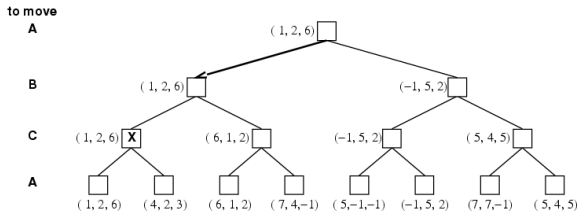
function MINIMAX(state) returns an action
inputs: state, current state in game

v ← MAX-VALUE(state)
return the action in SUCCESSORS(state) with value v

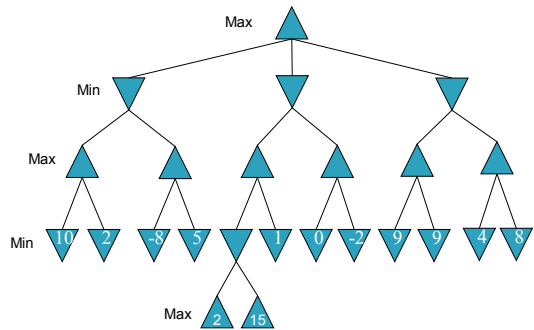
function MAX-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v ← -∞
for s in SUCCESSORS(state) do
    v ← max(v, MIN-VALUE(s))
return v

function MIN-VALUE(state) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
v ← +∞
for s in SUCCESSORS(state) do
    v ← min(v, MAX-VALUE(s))
return v
    
```

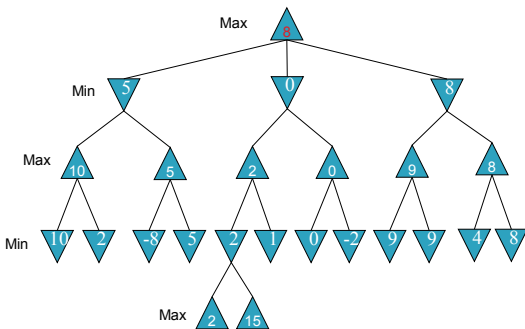
Multi-player?



MiniMax Example

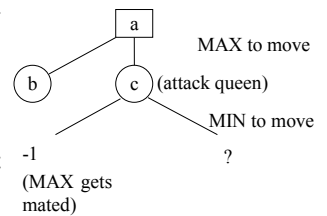


MiniMax Example



Pruning: Chess Example

- 2 players – MIN and MAX. MAX to move. Utilities of results: win=+1, draw=0, loss=-1.
- If MAX attacks opponent's Queen (option c), MAX can be checkmated next move.
- Thus, there is no need to consider any of c's other children.



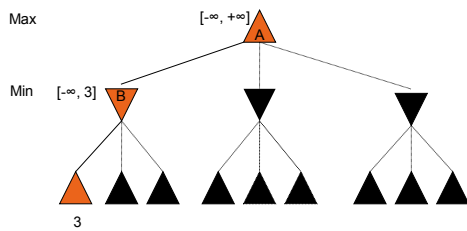
Pruning

- Consider a node n
- If a better choice m is available at the parent node (or further up the tree) then n will never be reached in actual play
- We may be able to determine this by only looking at some of n 's descendants
- If so, we can prune at this point without looking at all of n 's descendants

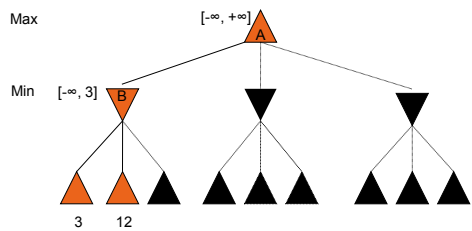
Alpha-beta pruning: use thresholds

- alpha
 - associated with maximising nodes
 - can never decrease
 - check whether the most recently created node is *less than* alpha
- beta
 - associated with minimising nodes
 - can never increase
 - check whether the most recently created node is *greater than* beta

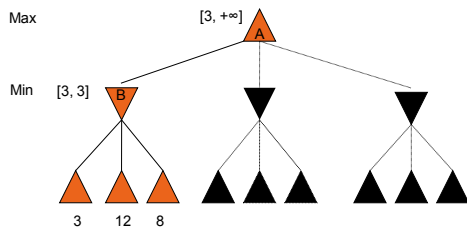
Alpha-beta pruning (1)



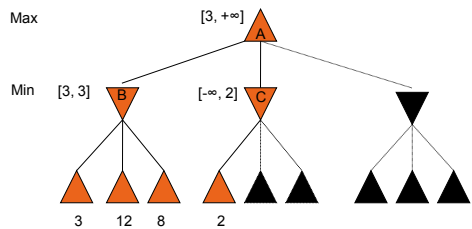
Alpha-beta pruning (2)



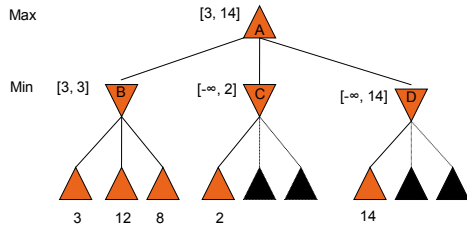
Alpha-beta pruning (3)



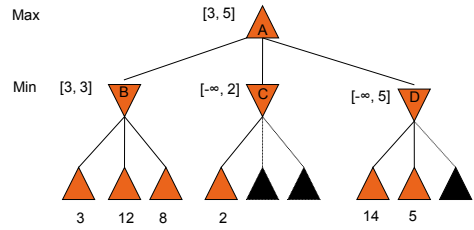
Alpha-beta pruning (4)



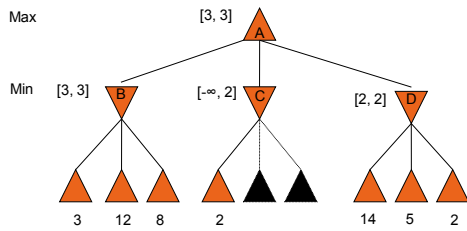
Alpha-beta pruning (5)



Alpha-beta pruning (6)



Alpha-beta pruning (7)



The pruning principle

- Evaluate leaf nodes (children) in first sub-tree
 - Determine maximum (for the parent)
 - beta value for the grandparent is this value
- Repeat:
 - Create one more grandchild, and evaluate
 - If value is greater than beta, ignore siblings of this grandchild
- Apply same process to the alpha values
 - entire sub-trees may be eliminated

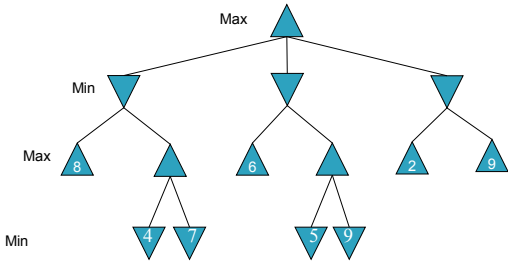
Alpha-beta pruning

- Effectiveness depends on the order in which descendent nodes are evaluated
- Best if best descendent node is examined first
- Can order nodes using an ordering function

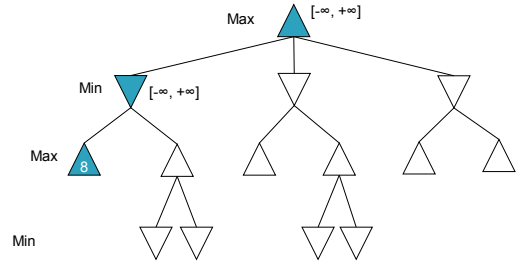
Luger and Stubblefield

- Alpha-beta pruning expresses a relation between nodes at ply n and nodes at ply $n+2$, under which entire sub-trees rooted at ply $n+1$ can be eliminated

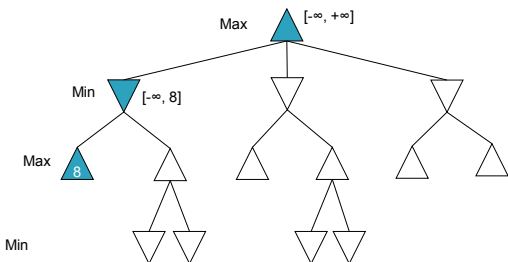
Alpha-Beta Pruning Example



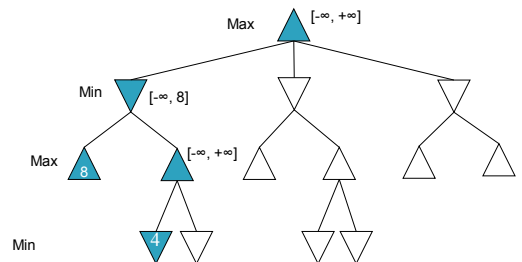
Alpha-Beta Pruning Example



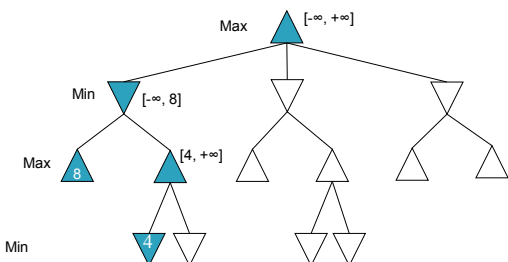
Alpha-Beta Pruning Example



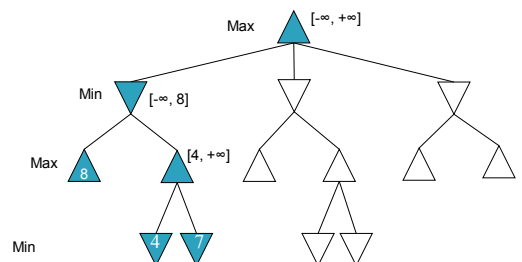
Alpha-Beta Pruning Example



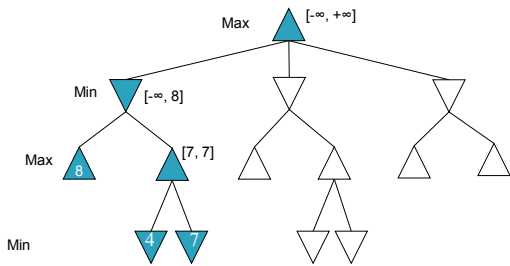
Alpha-Beta Pruning Example



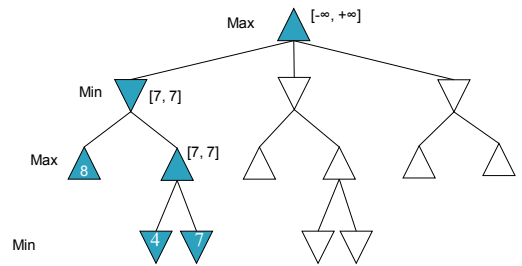
Alpha-Beta Pruning Example



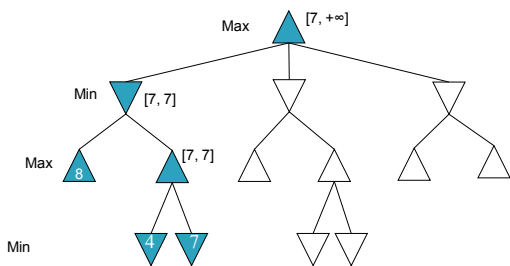
Alpha-Beta Pruning Example



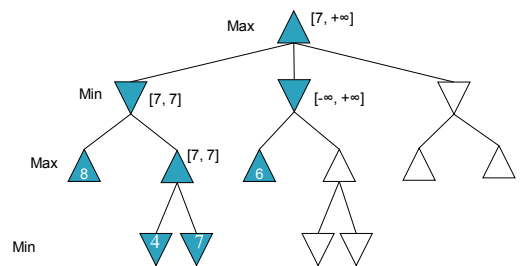
Alpha-Beta Pruning Example



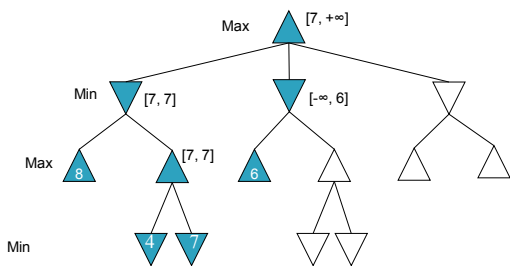
Alpha-Beta Pruning Example



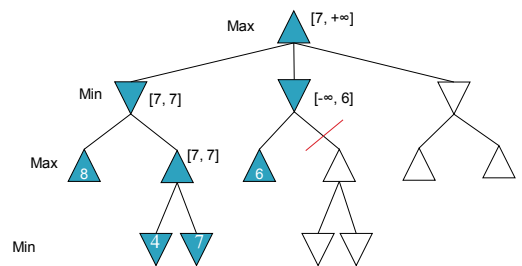
Alpha-Beta Pruning Example



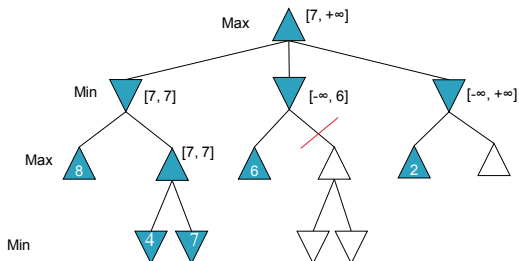
Alpha-Beta Pruning Example



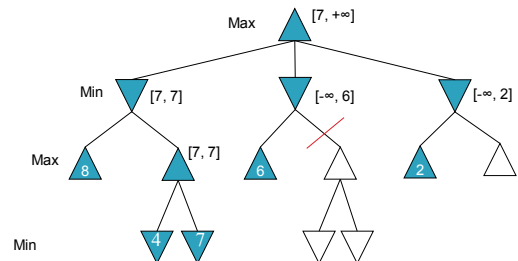
Alpha-Beta Pruning Example



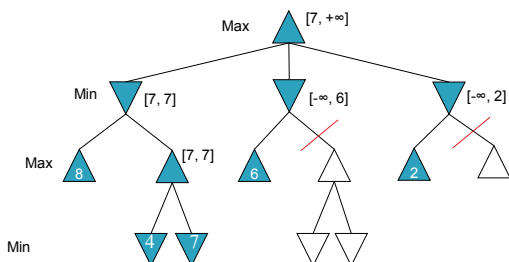
Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Does it work in practice?

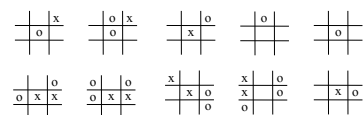
- The Minimax algorithm generates entire search space
- Alpha-beta pruning allows us to prune large parts
- Alpha-beta pruning still has to search to terminal states for a portion of the search space
 - Depth not typically practical – e.g. Chess
- Instead programs should cut off search earlier and apply an evaluation function

Evaluation Functions

- Returns an estimate of expected utility for the game from a given position
- Returns a numeric indicator of “goodness”
- Good evaluation function should:
 - Order terminal states in the same way as the true utility function
 - Computation must not take too long!
 - For non-terminal states, the evaluation function should be strongly correlated with the actual chances of winning

Tic-tac-toe heuristic

- Devise a suitable heuristic function (i.e. an evaluation function returning a numeric value) for the game of Tic-tac-toe
- Ensure that it gives appropriate values for the following board positions. The computer is playing x, and is to make the next move



Adding a cut off

- People don't analyse a game all the way to the end
- Apply this strategy to Minimax / alpha-beta pruning
- When we get to the maximum search depth, use an evaluation function to assign a value to the node
- Proceed with Minimax / alpha-beta pruning

Cutting off search

- For Minimax algorithm step 4
 - If x has not been assigned a value and either x is a terminal node or we have decided not to expand the tree further, compute its value using an evaluation function.
- Replace the terminal test with a cut off test
 - Can be at a set fixed depth limit
 - Iterative deepening
- May be used for alpha-beta pruning also

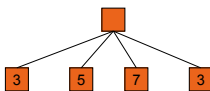
MiniMax with Cutoff and Evaluation

```
function MINIMAX(state) returns an action
  inputs: state, current state in game

  v ← MAX-VALUE(state)
  return the action in SUCCESSORS(state) with value v

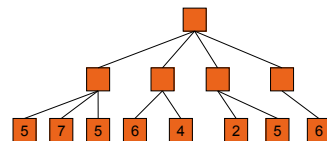
function MAX-VALUE(state) returns a utility value
  if CUTOFF-TEST(state, depth) then return EVAL(state)
  v ← -∞
  for s in SUCCESSORS(state) do
    v ← max(v, MIN-VALUE(s))
  return v

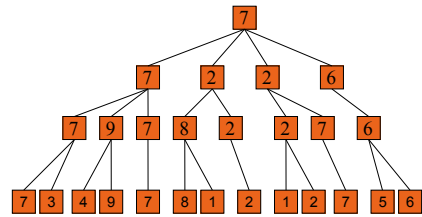
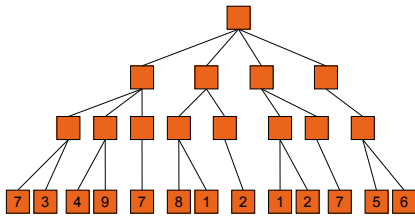
function MIN-VALUE(state) returns a utility value
  if CUTOFF-TEST(state, depth) then return EVAL(state)
  v ← +∞
  for s in SUCCESSORS(state) do
    v ← min(v, MAX-VALUE(s))
  return v
```



A non-exhaustive Minimax

- Minimax is three steps:
 - exhaustive generation to n ply
 - heuristic function evaluation of all leaf nodes
 - minimax up the tree





Issues / Improvements

- Quiescence
- Horizon effect
- Singular extensions
- Forward pruning

Quiescence

- Rapidly changing heuristic values (monitor the evaluation function)
- Unreliable values, extended tactical moves
- Search until values change slowly between ply
- Want to find a position that is quiescent, for which the evaluation function is not rapidly changing between a sequence of moves
- A search to a depth where the game becomes quiet

The horizon effect

- Opponent's killer move may just be visible at the fixed ply depth,
- A 'stalling move' will push the problem over the horizon.
- Continued stalling moves will merely delay the opponent's killer move.

Singular extensions

- 'Forced moves' reduce the branching factor (when few options are attractive for a player)
- More ply can be searched as it is now computationally practical

Forward Pruning

- Some moves are pruned immediately without further consideration
- Symmetric moves
- Deep in the search tree

Games of Chance

- Dice (e.g. Backgammon)
- Cards (e.g. Bridge, Poker)
- Game trees include nodes of chance as well as MAX and MIN
- Can use expected value – which chance node is most likely
- Expectiminimax
- Extra cost involved in looking ahead

Expectiminimax value =

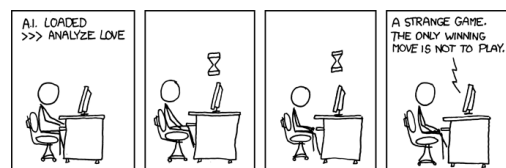
- Utility if n is a terminal state
- Max of successors Expectiminimax values if n is a MAX node
- Min of successors Expectiminimax values if n is a MIN node
- Weighted average of successors Expectiminimax values if n is a chance node



- Queensland Championships, 20-21 August 2011, UQ Centre
- Soccer
- Rescue
- Dance
- www.robocupjunior.org.au/qld

Summary

- MiniMax
- Alpha-beta pruning



xkcd.com