Adversarial Search

CS30
David Kauchak
Spring 2015

Some material borrowed from :
Sara Owsley Sood and others

Admin

- Assignment 10 out soon
  - May work in groups of up to 4 people
  - Due Sunday 4/26 (though, don't wait until the weekend to finish!)

A quick review of search

Problem solving via search:

- To define the state space, define three things:
  - is_goal
  - next_states
  - starting state

Uninformed search vs. informed search

- what's the difference?
- what are the techniques we've seen?
- pluses and minuses?

Why should we study games?

Clear success criteria

Important historically for AI

Fun ☺

Good application of search

- hard problems (chess 35^{100} states in search space, 10^{40} legal states)

Some real-world problems fit this model

- game theory (economics)
- multi-agent problems

Types of games

What are some of the games you've played?

Types of games: game properties

single-player vs. 2-player vs. multiplayer

Fully observable (perfect information) vs. partially observable

Discrete vs. continuous

real-time vs. turn-based

deterministic vs. non-deterministic (chance)

Strategic thinking $\stackrel{?}{=}$ intelligence

For reasons previously stated, two-player games have been a focus of AI since its inception…

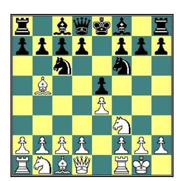
Begs the question: Is strategic thinking the same as intelligence?

Strategic thinking $\stackrel{?}{=}$ intelligence

Humans and computers have different relative strengths in these games:

humans

?



computers

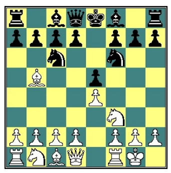
?

Strategic thinking $\stackrel{?}{=}$ intelligence

Humans and computers have different relative strengths in these games:

humans

good at evaluating the strength of a board for a player



computers

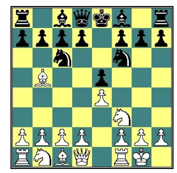
good at looking ahead in the game to find winning combinations of moves

Strategic thinking $\stackrel{?}{=}$ intelligence

How could you figure out how humans approach playing chess?

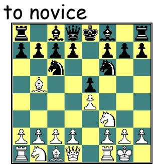
humans

good at evaluating the strength of a board for a player



How humans play games…

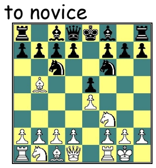
An experiment (by deGroot) was performed in which chess positions were shown to novice and expert players…



- experts could reconstruct these perfectly
- novice players did far worse…

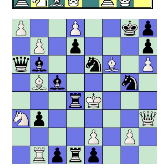
How humans play games…

An experiment (by deGroot) was performed in which chess positions were shown to novice and expert players…



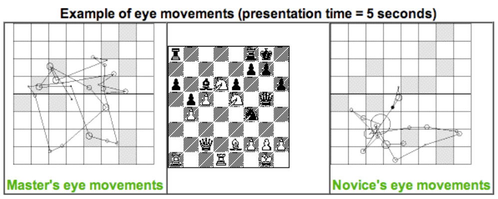
- experts could reconstruct these perfectly
- novice players did far worse…

Random chess positions (not legal ones) were then shown to the two groups



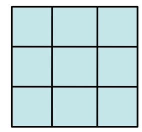
- experts and novices did just as badly at reconstructing them!

People are still working on this problem…



http://people.brunel.ac.uk/~hsstffg/frg-research/chess_expertise/

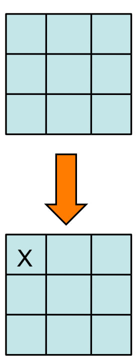
Tic Tac Toe as search



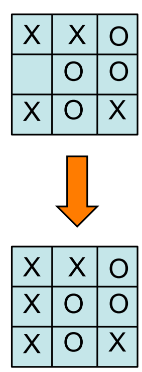
If we want to write a program to play tic tac toe, what question are we trying to answer?

Given a state (i.e. board configuration), what move should we make!

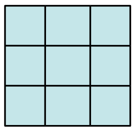
Tic Tac Toe as search



Tic Tac Toe as search

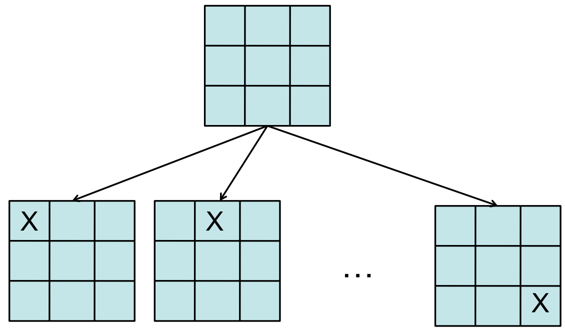


Tic Tac Toe as search

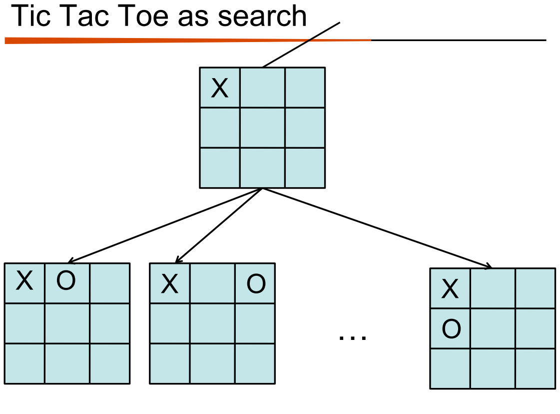


How can we pose this as a search problem?

Tic Tac Toe as search

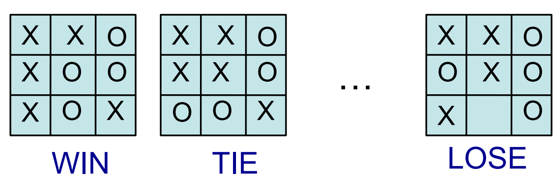


Tic Tac Toe as search



Tic Tac Toe as search

Eventually, we'll get to a leaf



How does this help us?

Try and make moves that move us towards a win, i.e. where there are leaves with a WIN.

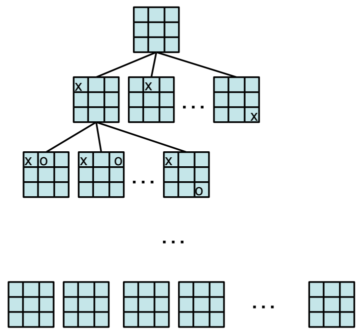
Tic Tac Toe

X's turn

O's turn

X's turn

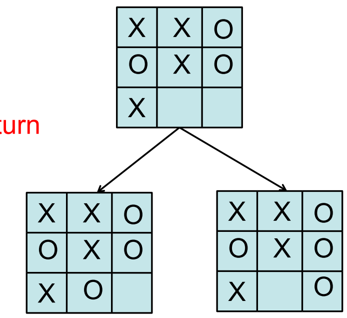
…



Problem: we don't know what O will do

I'm X, what will 'O' do?

O's turn

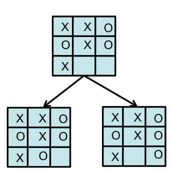


Minimizing risk

The computer doesn't know what move O (the opponent) will make

It can *assume*, though, that it will try and make the best move possible

Even if O actually makes a different move, we're no worse off. Why?



Optimal Strategy

An Optimal Strategy is one that is at least as good as any other, no matter what the opponent does

– If there's a way to force the win, it will

– Will only lose if there's no other option

Defining a scoring function

X	X	O
X	O	O
X	O	X

WIN
+1

X	X	O
X	X	O
O	O	X

TIE
0

…

X	X	O
O	X	O
X		O

LOSE
-1

Idea:
- define a function that gives us a "score" for how good each state is for us
- higher scores mean better for us

Defining a scoring function

Our (X) turn

X	X	O
	O	O
X	O	X

What should be the score of this state?

+1: we can get to a win

Defining a scoring function

Opponent's (O) turn

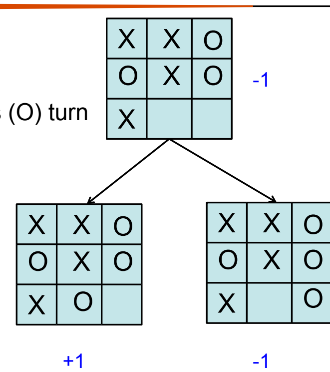
X	X	O
O	X	O
X		

What should be the score of this state?

-1: we can get to a win

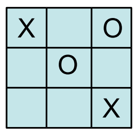
Defining a scoring function

Opponent's (O) turn



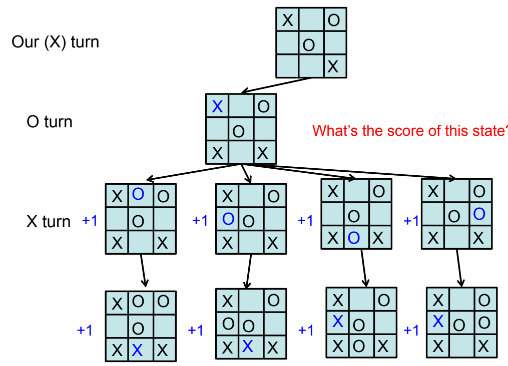
Defining a scoring function

Our (X) turn

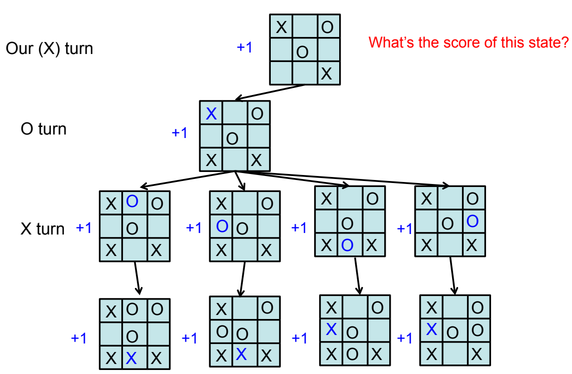


What should be the score of this state?

Defining a scoring function

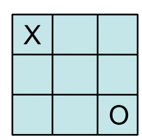


Defining a scoring function



Defining a scoring function

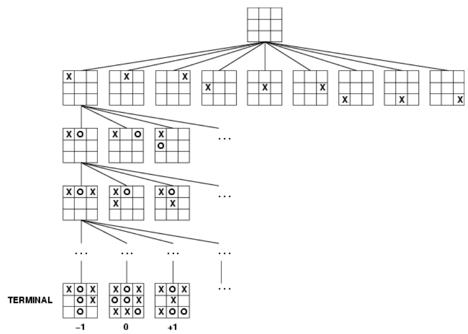
Our (X) turn



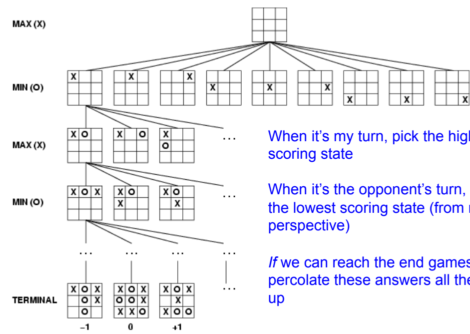
What should be the score of this state?

0: If we play perfectly and so does O, the best we can do is a tie (could do better if O makes a mistake)

How can X play optimally?



How can X play optimally?



When it's my turn, pick the highest scoring state

When it's the opponent's turn, assume the lowest scoring state (from my perspective)

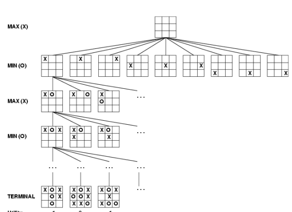
If we can reach the end games, we can percolate these answers all the way back up

How can X play optimally?

Start from the leaves and propagate the score up:

- if X's turn, pick the move that maximizes the utility
- if O's turn, pick the move that minimizes the utility

Is this optimal?

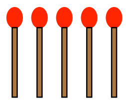


Minimax Algorithm: An Optimal Strategy

minimax(state) =
- if state is a terminal state
  Utility(state)
- if MY turn
  return the *maximum* of minimax(...)
  on all next states of state
- if OPPONENTS turn
  return the *minimum* of minimax(…)
  on all next states of state

- Uses recursion to compute the "value" of each state
- Proceeds to the leaves, then the values are "backed up" through the tree as the recursion unwinds
- What type of search is this?
- What does this assume about how MIN will play? What if this isn't true?

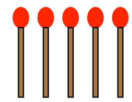
Baby Nim



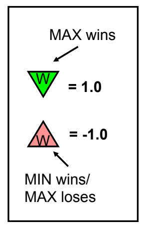
Take 1 or 2 at each turn
Goal: take the last match

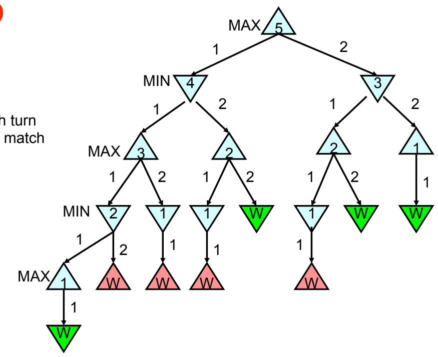
What move should I take?

Baby Nim

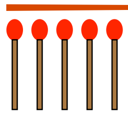


Take 1 or 2 at each turn
Goal: take the last match

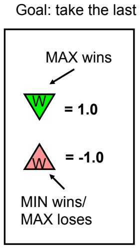


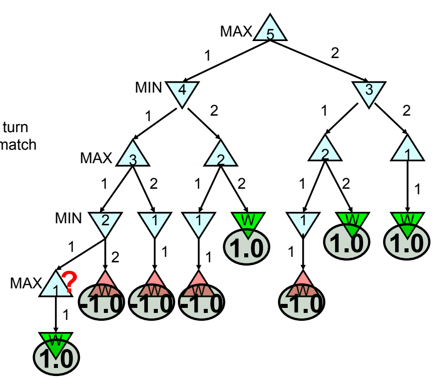


Baby Nim

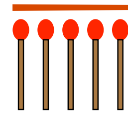


Take 1 or 2 at each turn
Goal: take the last match

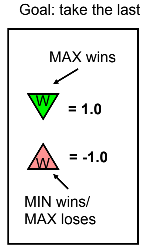


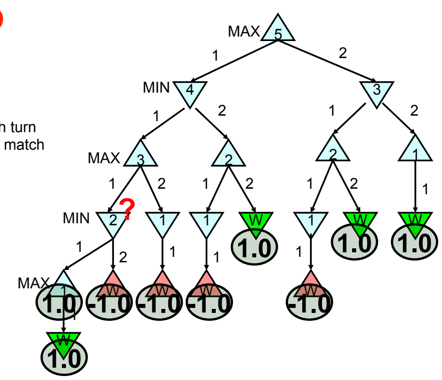


Baby Nim

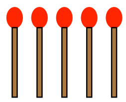


Take 1 or 2 at each turn
Goal: take the last match



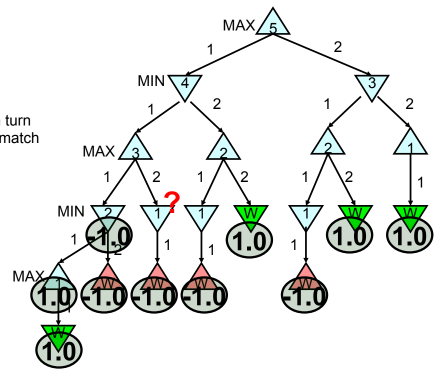


Baby Nim

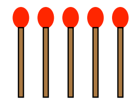


Take 1 or 2 at each turn
Goal: take the last match

MAX wins
= 1.0
= -1.0
MIN wins/
MAX loses

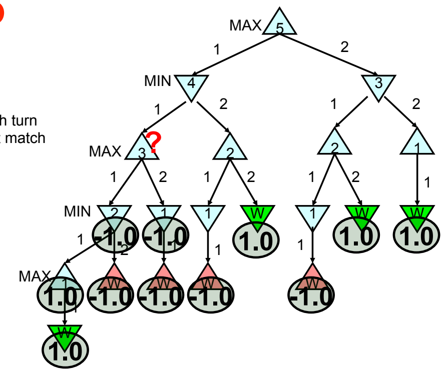


Baby Nim

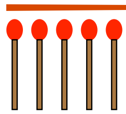


Take 1 or 2 at each turn
Goal: take the last match

MAX wins
= 1.0
= -1.0
MIN wins/
MAX loses

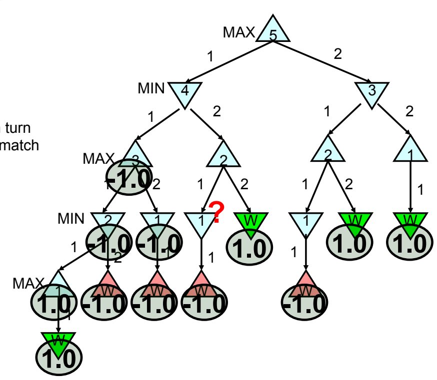


Baby Nim

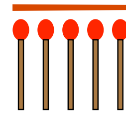


Take 1 or 2 at each turn
Goal: take the last match

MAX wins
= 1.0
= -1.0
MIN wins/
MAX loses

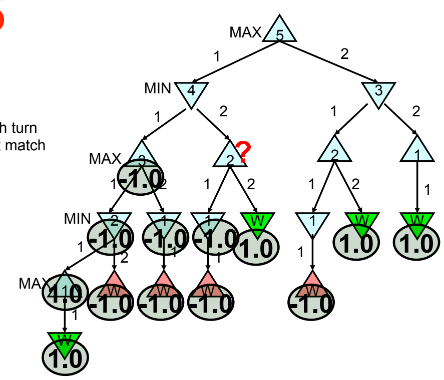


Baby Nim



Take 1 or 2 at each turn
Goal: take the last match

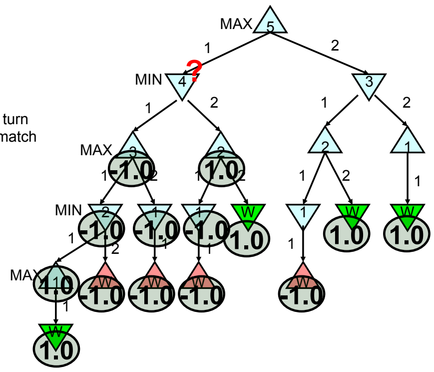
MAX wins
= 1.0
= -1.0
MIN wins/
MAX loses



Baby Nim

Take 1 or 2 at each turn
Goal: take the last match

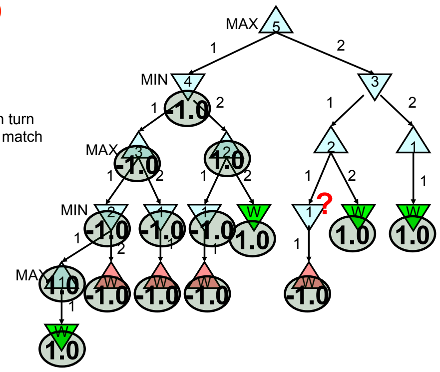
MAX wins
W = 1.0
W = -1.0
MIN wins/
MAX loses



Baby Nim

Take 1 or 2 at each turn
Goal: take the last match

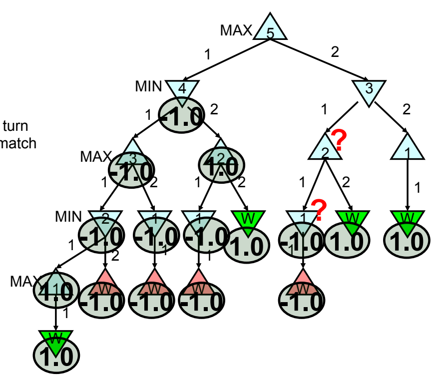
MAX wins
W = 1.0
W = -1.0
MIN wins/
MAX loses



Baby Nim

Take 1 or 2 at each turn
Goal: take the last match

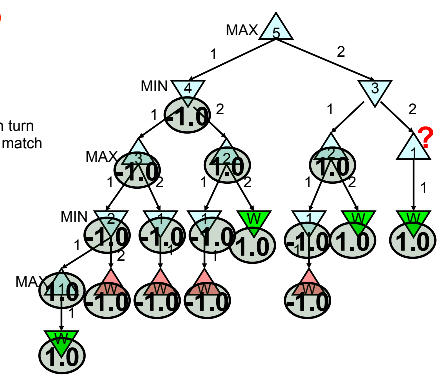
MAX wins
W = 1.0
W = -1.0
MIN wins/
MAX loses



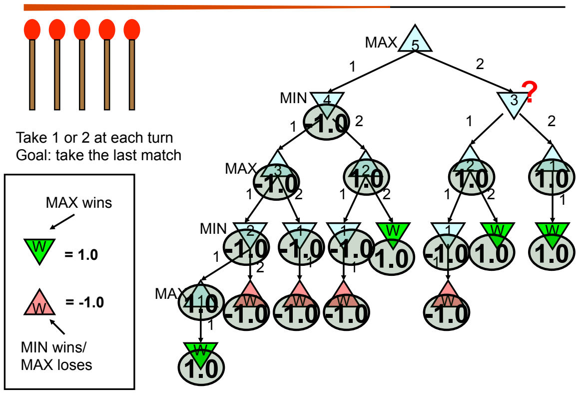
Baby Nim

Take 1 or 2 at each turn
Goal: take the last match

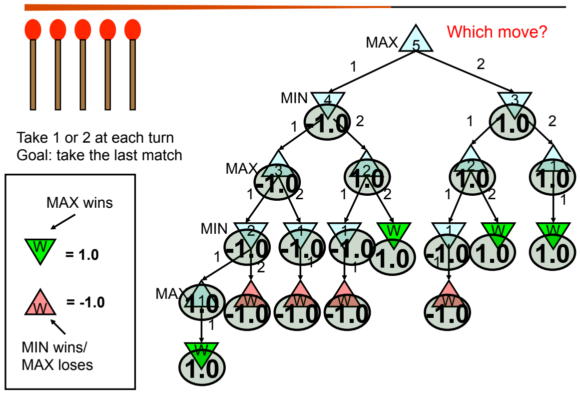
MAX wins
W = 1.0
W = -1.0
MIN wins/
MAX loses



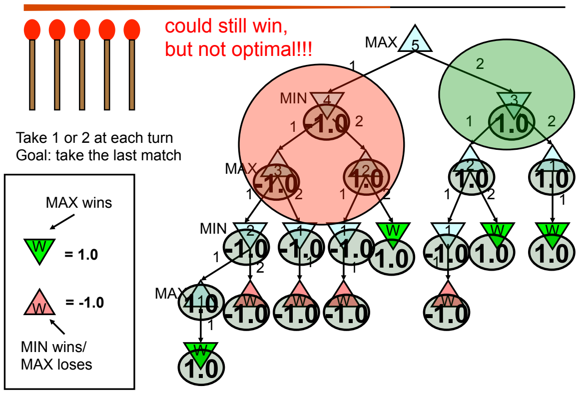
Baby Nim



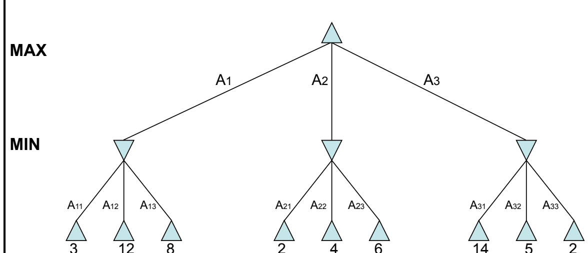
Baby Nim



Baby Nim

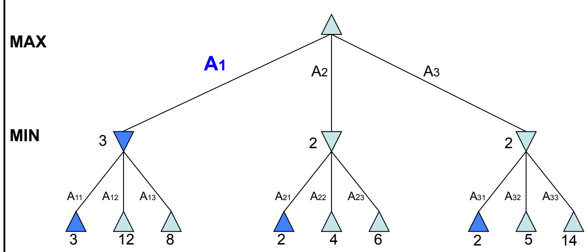


Minimax example 2



Which move should be made: A_1, A_2 or A_3?

Minimax example 2



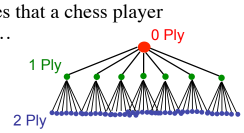
Properties of minimax

Minimax is optimal!

Are we done?

Games State Space Sizes

On average, there are ~35 possible moves that a chess player can make from any board configuration…



18 Ply!!

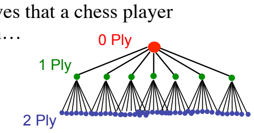
Hydra at home in the United Arab Emirates…

Branching Factor Estimates
for different two-player games

Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

Games State Space Sizes

On average, there are ~35 possible moves that a chess player can make from any board configuration…



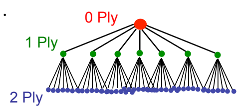
Branching Factor Estimates
for different two-player games

Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

Boundaries for qualitatively different games…

Games State Space Sizes

On average, there are ~35 possible moves that a chess player can make from any board configuration…



Branching Factor Estimates for different two-player games	
Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

Can search entire space

"solved" games

CHINOOK (2007) → Checkers

computer-dominated

Can't ☹

human-dominated

What do we do?

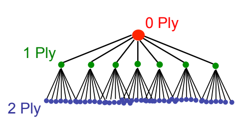
Alpha-Beta pruning

Games State Space Sizes

Pruning helps get a bit deeper

For many games, still can't search the entire tree

Now what?



Branching Factor Estimates for different two-player games	
Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

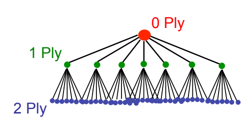
computer-dominated

Games State Space Sizes

Pruning helps get a bit deeper

For many games, still can't search the entire tree

Go as deep as you can:
- *estimate* the score/quality of the state (called an evaluation function)
- use that instead of the real score



Branching Factor Estimates for different two-player games	
Tic-tac-toe	4
Connect Four	7
Checkers	10
Othello	30
Chess	35
Go	300

computer-dominated

Tic Tac Toe evaluation functions

O	X	X
	O	

Ideas?

Example Tic Tac Toe EVAL

Tic Tac Toe
Assume MAX is using "X"

EVAL(*state*) =

if state is win for MAX:
+ ∞
if state is win for MIN:
- ∞
else:
(number of rows, columns and diagonals available to MAX) - (number of rows, columns and diagonals available to MIN)

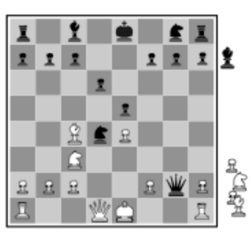
	X	O

= 6 - 4 = 2

O	X	X
	O	

= 4 - 3 = 1

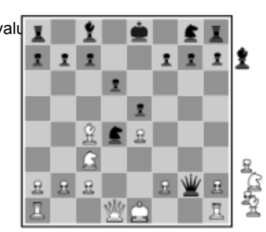
Chess evaluation functions



Ideas?

Chess EVAL

Assume each piece has the following valu
- pawn = 1;
- knight = 3;
- bishop = 3;
- rook = 5;
- queen = 9;



EVAL(*state*) =
sum of the value of white pieces –
sum of the value of black pieces

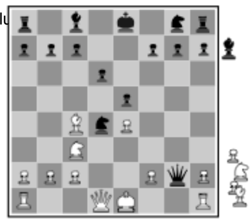
= 31 - 36 = -5

Chess EVAL

Assume each piece has the following values:

- pawn = 1;
- knight = 3;
- bishop = 3;
- rook = 5;
- queen = 9;

EVAL(*state*) =
sum of the value of white pieces –
sum of the value of black pieces



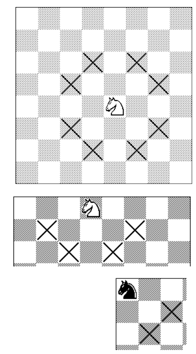
Any problems with this?

Chess EVAL

Ignores actual positions!

Actual heuristic functions are often a weighted combination of features

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + w_3 f_3(s) + \ldots$$



Chess EVAL

$$EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + w_3 f_3(s) + \ldots$$

number of pawns

number of attacked knights

1 if king has knighted, 0 otherwise

A feature can be any numerical information about the board

- as general as the number of pawns
- to specific board configurations

Deep Blue: 8000 features!

history/end-game tables

History

- keep track of the quality of moves from previous games
- use these instead of search

end-game tables

- do a reverse search of certain game configurations, for example all board configurations with king, rook and king
- tells you what to do in ***any*** configuration meeting this criterion
- if you ever see one of these during search, you lookup exactly what to do

end-game tables

Devastatingly good

Allows much deeper branching

- for example, if the end-game table encodes a 20-move finish and we can search up to 14
- can search up to depth 34

Stiller (1996) explored all end-games with 5 pieces

- one case check-mate required 262 moves!

Knoval (2006) explored all end-games with 6 pieces

- one case check-mate required 517 moves!

Traditional rules of chess require a capture or pawn move within 50 or it's a stalemate

Opening moves

At the very beginning, we're the farthest possible from any goal state

People are good with opening moves

Tons of books, etc. on opening moves

Most chess programs use a database of opening moves rather than search