## Games

## Adversaries

- Consider the process of reasoning when an adversary is trying to defeat our efforts
- In game playing situations one searches down the tree of alternative moves while accounting for the opponent's actions
- Problem is more difficult because the opponent will try to choose paths that avoid a win for the machine

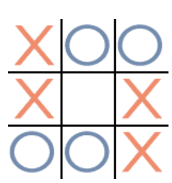
## Two-player games

- The object of a search is to find a path from the starting state to a goal state
- In one-player games such as puzzle and logic problems you get to choose every move
  - e.g. solving a maze
- In two-player games you alternate moves with another player
  - competitive games
  - each player has their own goal
  - search technique must be different

## Game trees

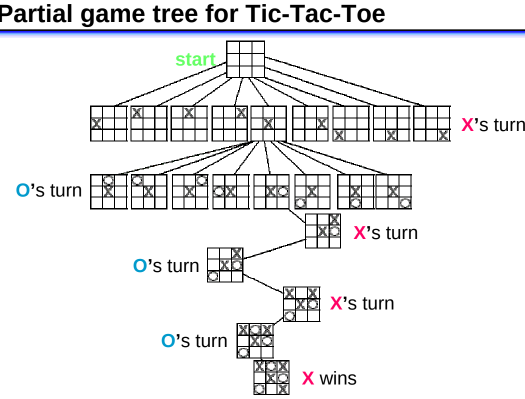
- A game tree is like a search tree in many ways …
  - nodes are search states, with full details about a position
    - characterize the arrangement of game pieces on the game board
  - edges between nodes correspond to moves
  - leaf nodes correspond to a set of goals
    - { win, lose, draw }
    - usually determined by a score for or against player
  - at each node it is one or other player's turn to move
- A game tree is not like a search tree because you have an opponent!

## Tic-Tac-Toe



- The first player is **X** and the second is **O**
- Object of game: get three of your symbol in a horizontal, vertical or diagonal row on a 3×3 game board
- **X** always goes first
- Players alternate placing **X**s and **O**s on the game board
- Game ends when a player has three in a row (a wins) or all nine squares are filled (a draw)

## Partial game tree for Tic-Tac-Toe



## Perfect information

- In a game with perfect information, both players know everything there is to know about the game position
  - no hidden information
    - opponents hand in card games
  - no random events
  - two players need not have same set of moves available
- Examples
  - Chess, Go, Checkers, Tic-Tac-Toe

## Payoffs

- Each game outcome has a *payoff,* which we can represent as a number
- In some games, the outcome is either a win or loss
  - we could use payoff values +1, -1
- In some games, you might also tie or draw
  - payoff 0
- In other games, outcomes may be other numbers
  - e.g. the amount of money you win at poker

## Problems with game trees

- Game trees are huge
  - Tic-Tac-Toe is 9! = 362,880
  - Checkers about $10^{40}$
  - Chess about $10^{120}$
  - Go is 361! $\approx 10^{750}$
- It is not good enough to find a route to a win
  - have to find a winning strategy
  - usually many different leaf nodes represent a win
  - much more of the tree needs to be explored

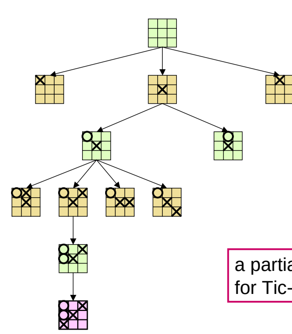
## Heuristics

- In a large game, you don't really know the payoffs
- A heuristic computes your best guess as to what the payoff will be for a given node
- Heuristics can incorporate whatever knowledge you can build into your program
- Make two key assumptions:
  - your opponent uses the same heuristic function
  - the more moves ahead you look, the better your heuristic function will work

## Evaluation functions

- It is usually impossible to solve games completely
  - Connect 4 has been solved
  - Checkers has not
- This means we cannot search entire game tree
  - we have to cut off search at a certain depth
    - like depth bounded depth first, lose completeness
- Instead we have to *estimate* cost of internal nodes
- We do this using a evaluation function
  - evaluation functions are heuristics
- Explore game tree using combination of evaluation function and search

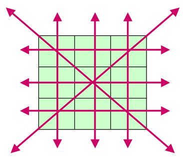
## Tic-Tac-Toe revisited

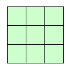


a partial game tree for Tic-Tac-Toe

## Evaluation function for Tic-Tac-Toe

- A simple evaluation function for Tic-Tac-Toe
  - count number of rows where **X** can win
  - subtract number of rows where **O** can win
- Value of evaluation function at start of game is zero
  - on an empty game board there are 8 possible winning rows for both **X** and **O**



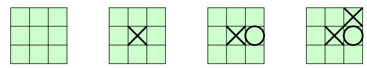


8-8 = 0

## Evaluating Tic-Tac-Toe

```
evalX = (number of rows where X can win) –
             (number of rows where O can win)
```

- After **X** moves in center, score for **X** is +4
- After **O** moves, score for **X** is +2
- After **X**'s next move, score for **X** is +3

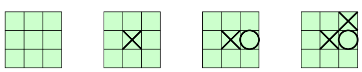


8-8 = 0    8-4 = 4    6-4 = 2    6-3 = 3

## Evaluating Tic-Tac-Toe

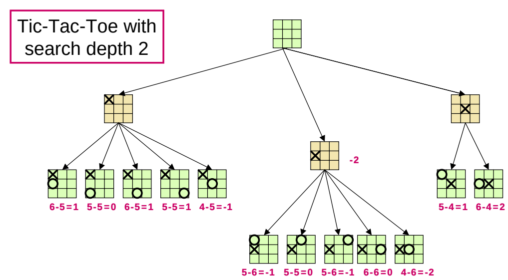
```
evalO = (number of rows where O can win) –
             (number of rows where X can win)
```

- After **X** moves in center, score for **O** is -4
- After **O** moves, score for **O** is +2
- After **X**'s next move, score for **O** is -3



8-8 = 0    4-8 = -4    4-6 = -2    3-6 = -3

## Search depth cutoff




Evaluations shown for **X**

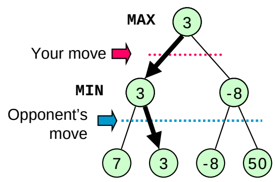
## Evaluation functions and search

- How do we use evaluation functions in our game tree?
- Idea is called minimaxing
- Call the two players MAX and MIN
  - MAX wants node with highest score
  - MIN wants leaf node with smallest score
- Always chose the move that will minimize the maximum damage that your opponent can do to you.

## Minimax search

- Assume that both players play perfectly
  - do not assume player will miss good moves or make mistakes
- Consider MIN's strategy
  - wants lowest possible score
  - must account for MAX
  - MIN's best strategy:
    - choose the move that minimizes the score that will result when MAX chooses the maximizing move
- MAX does the opposite

## Minimaxing

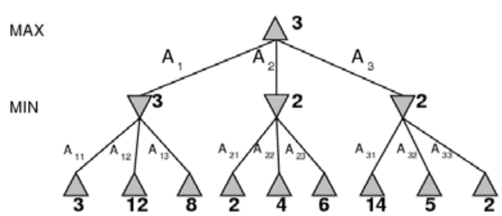



- Your opponent will choose smaller numbers
- If you move left, your opponent will choose 3
- If you move right, your opponent will choose -8
- Thus your choices are only 3 or -8
- You should move left

## Minimax procedure

- Evaluate positions at cutoff search depth and propagate information upwards in the tree
  - score of MAX nodes is the maximum of child nodes
  - score of MIN nodes is the minimum of child nodes
- Bottom-up propagation of scores eventually gives score for all possible moves from root node
- This gives us the best move to make

## Minimax




## Minimax is bad

- The problem with minimax is that it is inefficient
  - search to depth $d$ in the game tree
  - suppose each node has at most $b$ children
  - calculate the exact score at every node
  - in worst case we search $b^d$ nodes – exponential!
- However, many nodes are useless
  - there are some nodes where we don't need to know exact score because we will never take that path in the future

## Is there a good minimax?

- Yes! We just need to prune branches we do not to search from the tree
- Idea:
  - start propagating scores as soon as leaf nodes are generated
  - do not explore nodes which cannot affect the choice of move
    - that is, do not explore nodes that we can know are no better than the best found so far
- The method for pruning the search tree generated by minimax is called Alpha-beta

## Alpha-beta values

- At MAX node we store an alpha ($\alpha$ ) value
  - $\alpha$ is lower bound on the exact minimax score
  - with best play MAX can score at least $\alpha$
  - the true value might be > $\alpha$
  - if MIN can choose nodes with score < $\alpha$, then MIN's choice will never allow MAX to choose nodes with score > $\alpha$
- Similarly, at MIN nodes we store a beta ($\beta$) value
  - $\beta$ is upper bound on the exact minimax score
  - with best play MIN can score no more than $\beta$
  - the true value might be $\leq \beta$

## Alpha-beta pruning

- Two key points:
  - alpha values can *never decrease*
  - beta values can *never increase*
- Search can be discontinued at a node if:
  - It is a Max node and
    - the alpha value is ≥ the beta of any Min ancestor
    - this is *beta cutoff*
  - Or it is a Min node and
    - the beta value is ≤ the alpha of any Max ancestor
    - this is *alpha cutoff*

## The importance of cutoffs

- If you can search to the end of the game, you know exactly the path to follow to win
  - thus the further ahead you can search, the better
- If you can ignore large parts of the tree, you can search deeper on the other parts
  - the number of nodes at each turn grows exponentially
  - you want to prune braches as high in the tree as possible
- Exponential time savings possible