**Artificial Intelligence**

Module 3: Search Strategies

## PART 3.5: Adversarial Search

Dr. Chandra Prakash
Assistant Professor
Department of Computer Science and Engineering

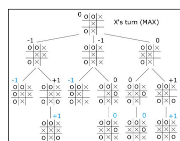
*(Slides adapted from StuartJ. Russell, B Ravindran, Mausam, Dan Klein and Pieter Abbeel)*

# Module 3: Search Strategies

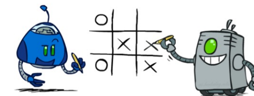

- PART 3.1: Search
- PART 3.2: Uninformed Search
- PART 3.3: Informed/Heuristic Search
- PART 3.4: Beyond Classical Search
  - Local Search
    - Generate-and-test
    - Hill climbing
    - Simulated Analing
  - Problem reduction
- PART 3.5: Adversarial Search
- PART 3.6: Constraint Satisfaction Problems

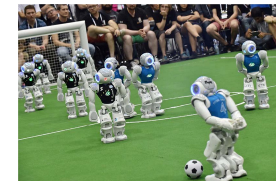
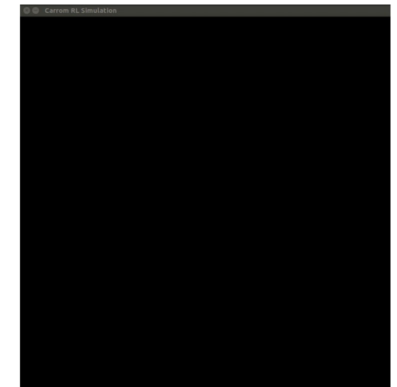

# Games you play ??

- Identify it is :
  - Single/ Multi-Player
  - Cooperative vs. competitive
  - Deterministic and non-deterministic
  - Probabilistic



# Play/Search with Other Agents



(Credits: Samiran Roy. Graphic source: https://github.com/samiranrl/Carrom_rl)

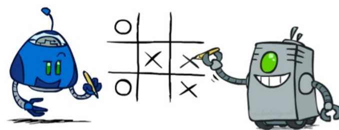
# PRISONER'S DILEMMA

# OVERVIEW OF GAMES

- Games are a form of **multi-agent** environment
  - What do other agents do and how do they affect our success?
- Cooperative vs. competitive multi-agent environments .
  - Competitive multi-agent environments give rise to adversarial search.
  - Adversarial Search Problem also know as GAMES
- Specifics:
  - Sequences of player's decisions we control
  - Decisions of other player(s) we do not control
    - **Opponent's behavior introduces uncertainty**
- **Contingency problem:** many possible opponent's moves must be "covered" by the solution
- Rational opponent – maximizes its own utility (payoff) function

# Game Playing

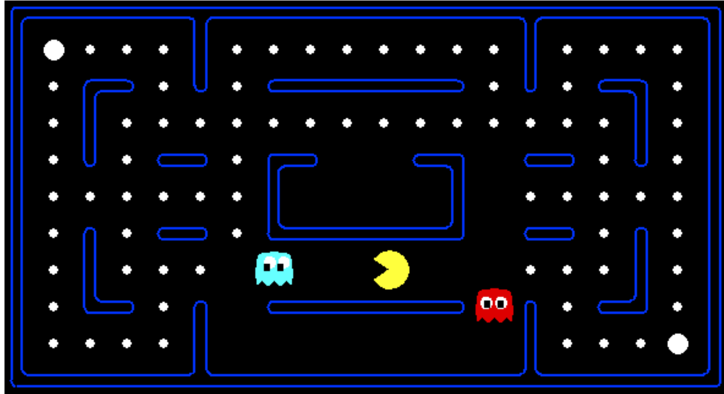
- **Why do AI researchers study game playing?**
  - Well defined rules
  - Easy to evaluate
  - It's a good reasoning problem, formal and nontrivial.
  - Direct comparison with humans and other computer programs is easy.
- **Adversarial search** in Game playing :
  - Examine the problems that arise when we try to plan ahead in a world where other agents are planning against us
- What Kinds of Games?
  - Mainly games of strategy with the following characteristic
    - Sequence of moves to play
    - Rules that specify possible moves
    - Rules that specify a payment for each move
    - Objective is to maximize your payment



# Search Vs. Games Problems

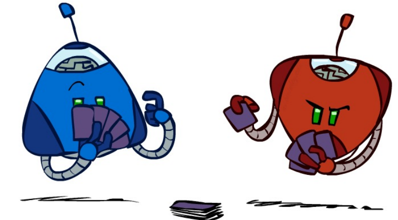
| Search Problems | GAME Problems |
|---|---|
| Specifying a move for every possible opponent reply | Unpredictable opponent |
| Unlikely to find goal, must approximate | Time limits |
| no adversary | adversary |
| Solution is (heuristic) method for finding goal | Solution is **strategy** (strategy specifies move for every possible opponent reply). |
| Heuristic techniques can find *optimal* solution | Optimality depends on opponent. Why?<br>Time limits force an *approximate* solution |
| **Evaluation function:** estimate of cost from start to goal through given node | **Evaluation function:** evaluate "goodness" of game position |
| **Examples:**<br>path planning, scheduling activities | **Examples:**<br>chess, checkers, Othello, backgammon |

# Video of Demo Mystery Pac-man



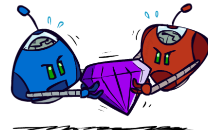
# Types of Games

- Many different kinds of games!
- Axes:
  – Deterministic or stochastic?
  – One, two, or more players?
  – Zero sum?
  – Perfect information (can you see the state)?
- Want algorithms for calculating a strategy (policy) which recommends a move from each state



# Types of Games

- General Games
  – Agents have independent utilities (values on outcomes)
  – Cooperation, indifference, competition, and more are all possible
    - We don't make AI to act in isolation, it should a) work around people and b) help people
    - That means that every AI agent needs to solve a game
- Common payoff games
  – Discussion: Use a technique you've learned so far to solve one!

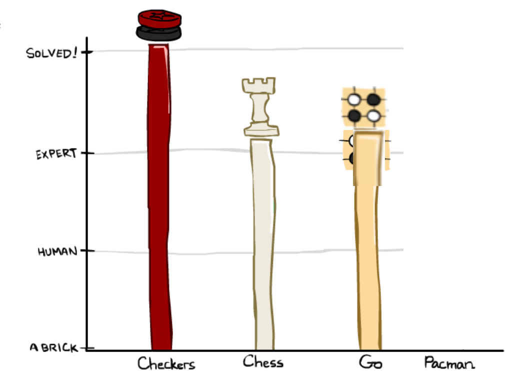


| | B chooses B1 | B chooses B2 | B chooses B3 |
|---|---|---|---|
| A chooses A1 | +3 | −2 | +2 |
| A chooses A2 | −1 | 0 | +4 |
| A chooses A3 | −4 | −3 | +1 |

- Zero-Sum Games
  – Agents have opposite utilities (values on outcomes)
  – Lets us think of a single value that one maximizes and the other minimizes
  – Adversarial, pure competition
  – Zero-sum describes a situation in which a participant's gain or loss is exactly balanced by the losses or gains of the other participant(s).
  – If the total gains of the participants are added up, and the total losses are subtracted, they will sum to zero.

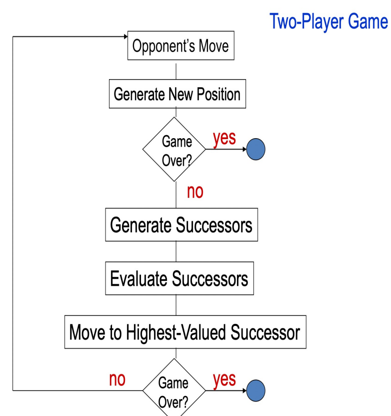
# Zero-Sum Game

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Go :2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search, learned evaluation function.**
- **Pacman**



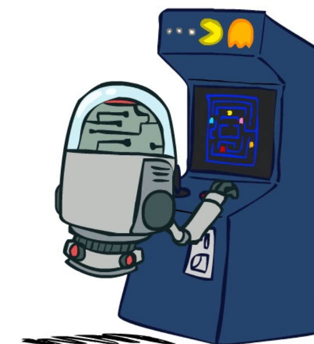
# Typical AI assumptions

- Two agents whose actions alternate
- Utility values for each agent are the opposite of the other
  - creates the adversarial situation
- Fully observable environments
- In game theory terms:
  - **Zero-sum games** of perfect information.



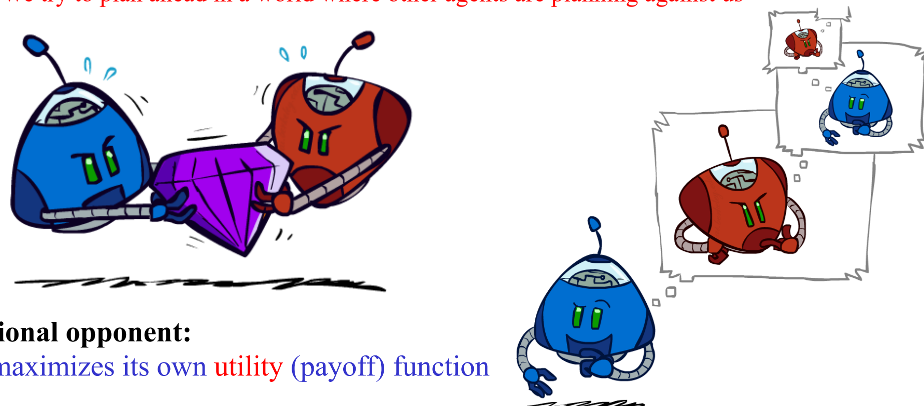

# Deterministic Games with Terminal Utilities

- Many possible formalizations, one is:
  - States:
    - S (start at $s_0$)
  - Players:
    - P={1...N} (usually take turns)
  - Actions:
    - A (may depend on player / state)
  - Transition Function:
    - $SxA \rightarrow S$
  - Terminal Test:
    - $S \rightarrow \{t,f\}$
  - Terminal Utilities:
    - $SxP \rightarrow R$
- Solution for a player is a policy: $S \rightarrow A$



# Adversarial Games

We try to plan ahead in a world where other agents are planning against us



**Rational opponent:**

- maximizes its own utility (payoff) function

# Games as Adversarial Search

- States:
  - board configurations
- Initial state:
  - the board position and which player will move
- Successor function:
  - returns list of (move, state) pairs, each indicating a legal move and the resulting state
- Terminal test:
  - determines when the game is over
- Utility function:
  - gives a numeric value in terminal states
  - (e.g., -1, 0, +1 for loss, tie, win)

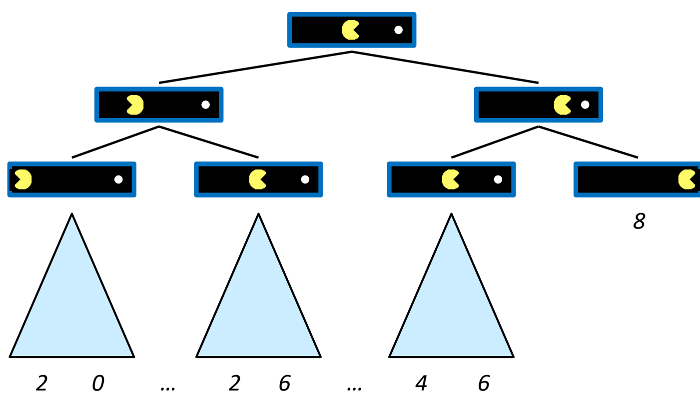
## More on Search

- Cost -> Utility
  - no longer minimizing cost!
  - agent now wants to maximize its score/utility!
- Search Tree
- Size of search trees
  - b = branching factor
  - d = number of moves by both players
  - Search tree is $O(b^d)$
  - Chess
    - b ~ 35
    - D ~100
      - search tree is ~ $10^{154}$ (!!)

## Game Trees

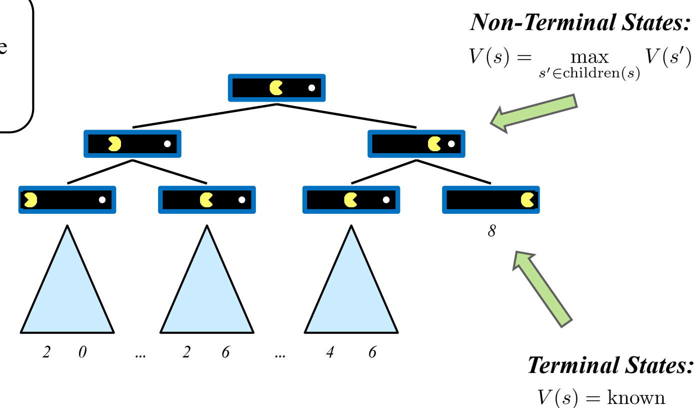
- A tree with three types of nodes,
  - Terminal nodes :
    - Terminal nodes have no children.
  - Min nodes and Max nodes.
  - The tree has alternating levels of Max and Min nodes, representing the turns of Player-1 and Player-2 in making moves
- All nodes represent some state of the game
- Terminal nodes are labeled with the payoff for Player-1.
  - It could be Boolean (such as WON or LOST).
  - In large games, where looking ahead up to the WON / LOST states is not feasible, the payoff at a terminal node may represent a heuristic cost representing the quality of the state of the game from Player-1's perspective
- The payoff at a Min node is the minimum among the payoffs of its successors
- The payoff at a Max node is the maximum among the payoffs of its successors
- If Player-1 aims to maximize its payoff, then it represents Max nodes, else it represents Min nodes.

## Single-Agent Trees




## Value of a State

**Value of a state:** The best achievable outcome (utility) from that state $V$



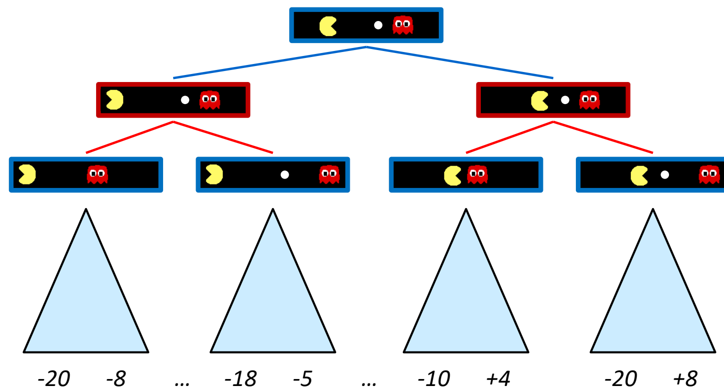

***Non-Terminal States:***

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

***Terminal States:***

$$V(s) = \text{known}$$

## Adversarial Game Trees



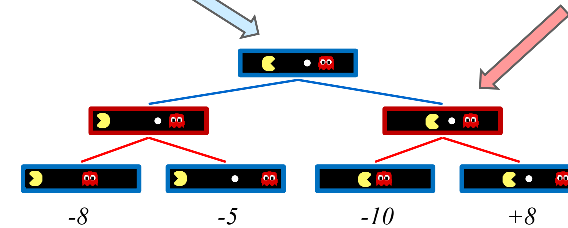

## Minimax Values

*States Under Agent's Control:*

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

*States Under Opponent's Control:*

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$




*Terminal States:*

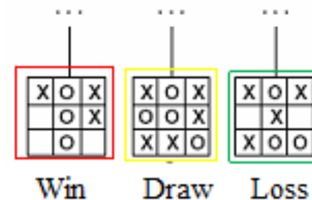
$V(s) = \text{known}$

## EXAMPLE OF AN ADVERSARIAL 2 PERSON GAME: TIC-TAC-TOE

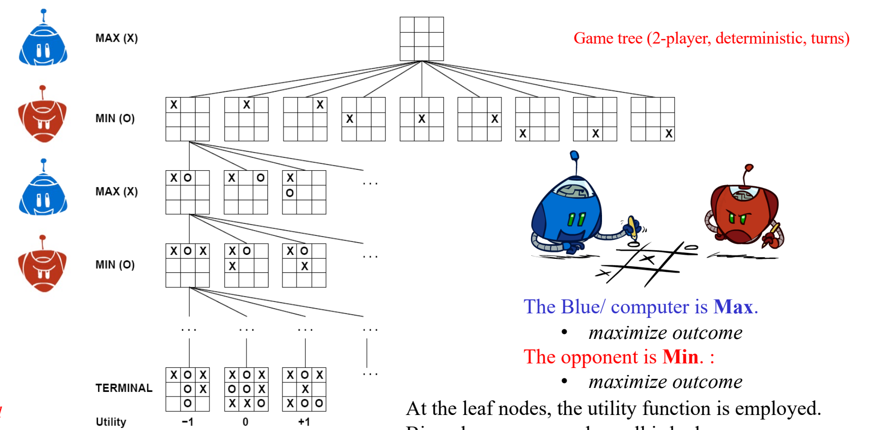
A two player game where minmax algorithm is applies is Tic–Tac-Toe.

In this game in order to win
- must fill a row, a column or diagonal (X or O).

**Generate a Game tree**




## Tic-Tac-Toe Game Tree



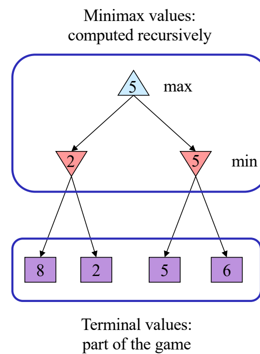

The Blue/ computer is **Max**.
- *maximize outcome*

The opponent is **Min**. :
- *maximize outcome*

At the leaf nodes, the utility function is employed. Big value means good, small is bad.

# Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary




# Mini-Max Terminology

- move: a move by both players
- ply: a half-move
- utility function: the function applied to leaf nodes
- backed-up value
  - of a max-position: the value of its largest successor
  - of a min-position: the value of its smallest successor
- minimax procedure: search down several levels; at the bottom level apply the utility function, back-up values all the way up to the root node, and that node selects the move.

# Minimax Implementation (Dispatch)

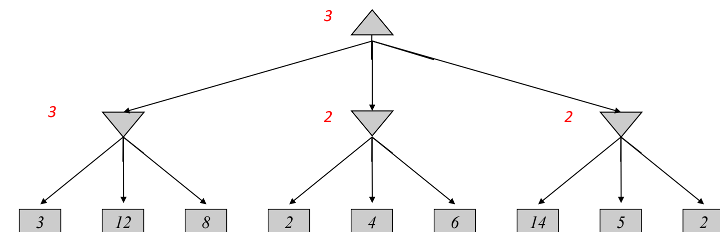
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

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

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

# Minimax Example

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:



## Pseudocode for Minimax Algorithm

***function*** *MINIMAX-DECISION(state)* ***returns*** *an action*
  ***inputs:*** *state, current state in game*
  *v←MAX-VALUE(state)*
  ***return*** *the action in SUCCESSORS(state) with value v*

---

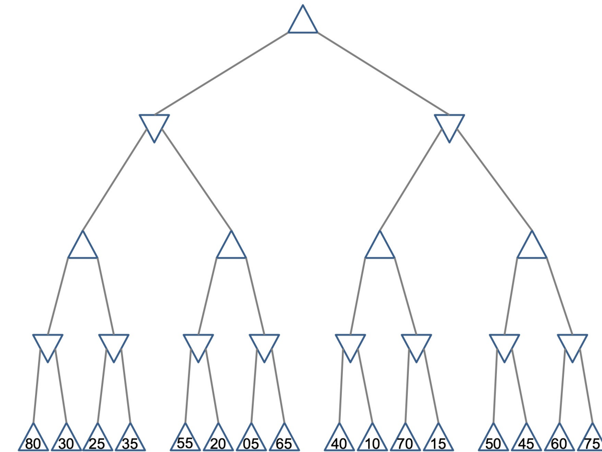
***function*** *MAX-VALUE(state)* ***returns*** *a utility value*
  ***if*** *TERMINAL-TEST(state)* ***then return*** *UTILITY(state)*
  *v ← -∞*
  ***for*** *a,s in SUCCESSORS(state)* ***do***
    *v ← MAX(v,MIN-VALUE(s))*
  ***return*** *v*

---

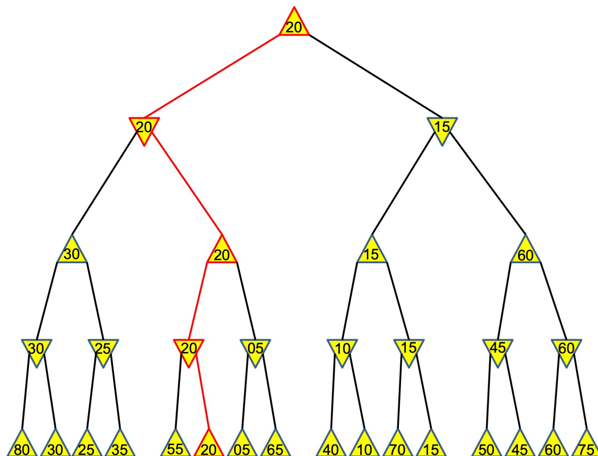
***function*** *MIN-VALUE(state)* ***returns*** *a utility value*
  ***if*** *TERMINAL-TEST(state)* ***then return*** *UTILITY(state)*
  *v ← ∞*
  ***for*** *a,s in SUCCESSORS(state)* ***do***
    *v ← MIN(v,MAX-VALUE(s))*
  ***return*** *v*

Minimax algorithm Adversarial analogue of DFS

## Exercise



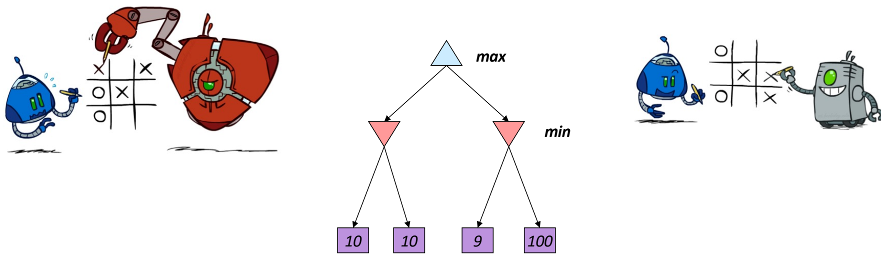

## Solution




## Minimax Strategy

- Why do we take the min value every other level of the tree?
- These nodes represent the opponent's choice of move.
- The computer assumes that the human will choose that move that is of least value to the computer.

## Minimax Properties

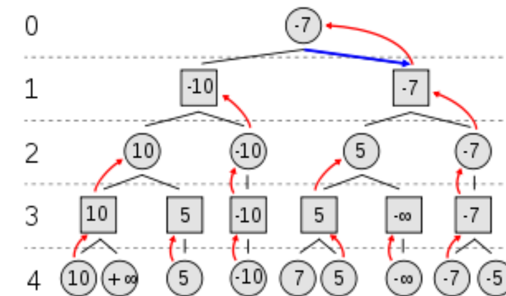



Optimal against a perfect player. Otherwise?

## Video of Demo Min vs. Exp (Min)

## Video of Demo Min vs. Exp (Exp)

## Example of Algorithm Execution

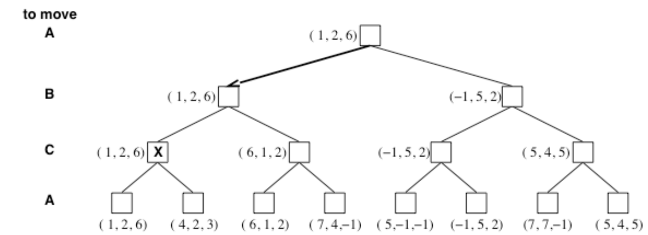


# Aspects of Multiplayer Games

- Upto previous slides assumes that each player operates to maximize only their own utility
- In practice, players make alliances
  - E.g, C strong, A and B both weak
  - May be best for A and B to attack C rather than each other
- If game is not zero-sum (i.e., utility(A) = - utility(B) then alliances can be useful even with 2 players
  - e.g., both cooperate to maximum the sum of the utilities

# Multiplayer games

- Games allow more than two players
- Single minimax values become vectors




# PROPERTIES OF MINIMAX

Complete depth-first exploration of the game tree

- Complete?
  - Yes (if tree is finite)
- Optimal?
  - Yes (against an optimal opponent)
  - No (does not exploit opponent weakness against suboptimal opponent)
- Time complexity?
  - **O($b^m$)** Max depth = m, b legal moves at each point
- Space complexity?
  - **O(bm)**
  - (depth-first exploration, for algorithm that generates all successors at once or O(m) for an algorithm that generates succesors one at a time )

Efficent similar to (exhaustive) DFS

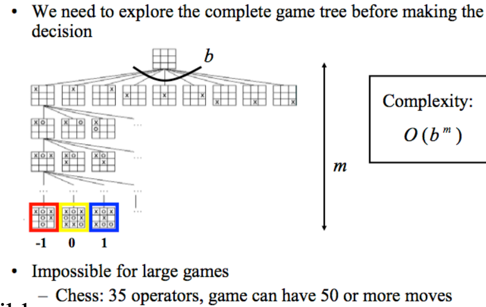
# PROPERTIES OF MINIMAX

- **Minimax advantages:**
  - Returns an optimal action, assuming perfect opponent play.
  - Minimax is the simplest possible (reasonable) game search algorithm.
- **Minimax disadvantages:**
  - It's completely infeasible in practice.
    - When the search tree is too large, we need to limit the search depth and apply an evaluation function to the cut-off states.

# Good Enough?

- Chess:
  - branching factor b≈35
  - game length m≈100
  - search space $b^m \approx 35^{100} \approx 10^{154}$
- The Universe:
  - number of atoms $\approx 10^{78}$
  - age $\approx 10^{18}$ seconds
  - $10^8$ moves/sec x $10^{78}$ x $10^{18}$ = $10^{104}$
- Exact solution completely infeasible
  - for "reasonable“ games exact solution completely infeasible

- We need to explore the complete game tree before making the decision




- Impossible for large games
  - Chess: 35 operators, game can have 50 or more moves

Many nodes are useless : There are some nodes where we don’t need to know exact score because we will never take path in the future

# Solution to the complexity problem

**Two solutions:**

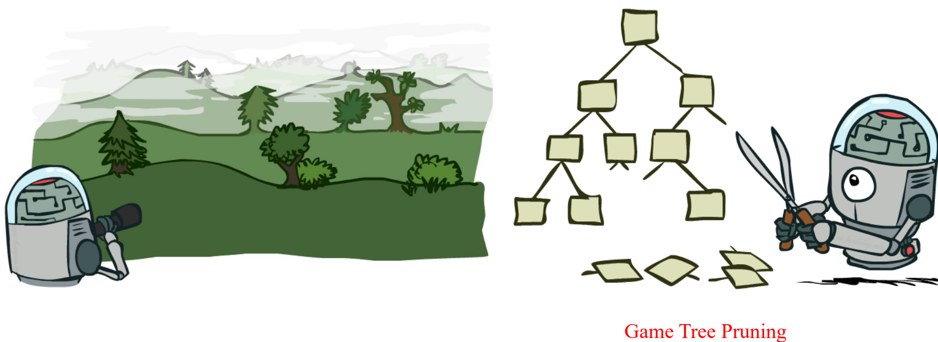
1. **Dynamic pruning of redundant branches** of the search tree
   - identify a provably suboptimal branch of the search tree before it is fully explored
   - Eliminate the suboptimal branch

   **Procedure: Alpha-Beta pruning**

2. **Early cutoff of the search tree**
   - uses imperfect minimax value estimate of non-terminal states (positions)

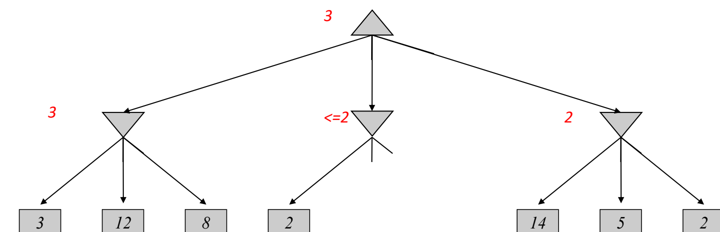


# Resource Limits



Game Tree Pruning

# Minimax Example

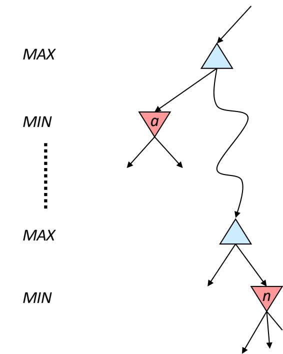


## Is there a good Min-Max ?

- Yes !
- We just need to prune branches that are not required in searching
- Idea:
  - Start propagating scores as soon as leaf nodes are generated
  - Do not explore nodes which cannot affect the choice of move
- The method for pruning the search tree generated by minimax is called Alpha-Beta

## Alpha-Beta Pruning

- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node $n$
  - We're looping over $n$'s children
  - $n$'s estimate of the childrens' min is dropping
  - Who cares about $n$'s value? MAX
  - Let $a$ be the best value that MAX can get at any choice point along the current path from the root
  - If $n$ becomes worse than $a$, MAX will avoid it, so we can stop considering $n$'s other children (it's already bad enough that it won't be played)
- MAX version is symmetric




## Alpha (α) Beta (β) values

- **Computing alpha-beta values**
  - α value is a lower-bound on the actual value of a Max node, maximum across seen children
  - β value is an upper-bound on actual value of a Min node, minimum across seen children
- **Propagation**
  - Update α ,β values by propagating upwards values of terminal nodes
  - Update α ,β values down to allow pruning
- **Two key points:**
  - α value can never decrease
  - β value can never increase
- **Search can be discontinued at a node if:**
  - It is a Max node and
    $\alpha \geq \beta$ , it is beta cutoff
  - It is a Min node and
    $\beta \leq \alpha$ , it is alpha cutoff

## Alpha-Beta Implementation

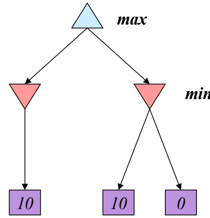
α: MAX's best option on path to root
β: MIN's best option on path to root

```
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
```

```
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
```

## Alpha-Beta Pruning Properties

- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…
- This is a simple example of metareasoning (computing about what to compute)



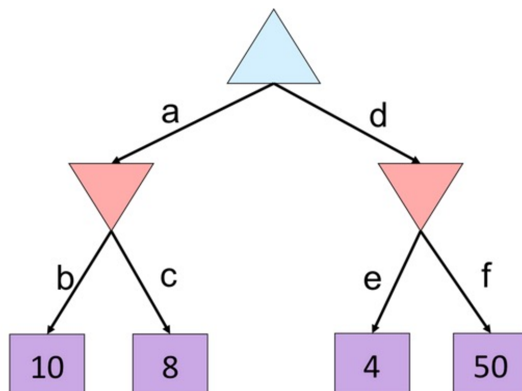

## Pseudocode for Alpha-Beta Algorithm

***function*** *ALPHA-BETA-SEARCH(state)* ***returns*** *an action*
  ***inputs:*** *state, current state in game*
  *v←MAX-VALUE(state, - ∞ , +∞)*
  ***return*** *the action in SUCCESSORS(state) with value v*

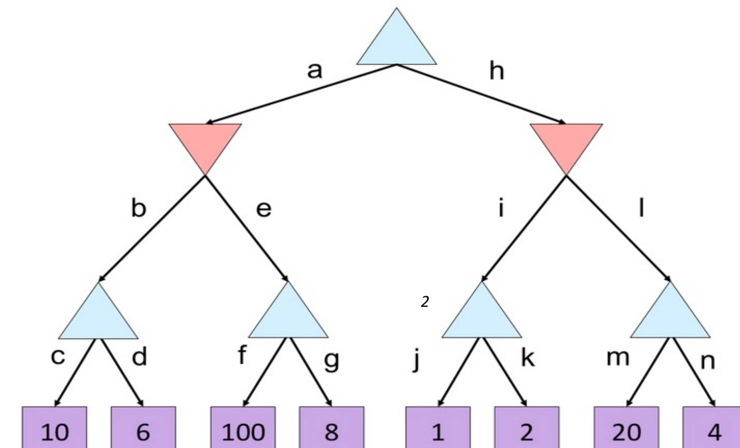
---

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

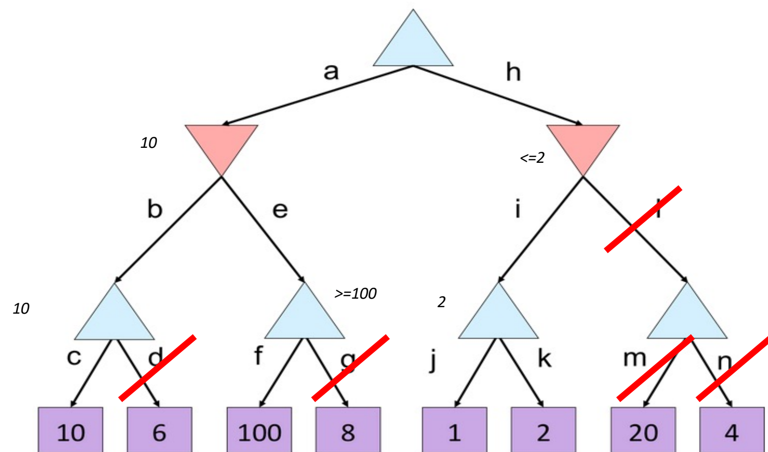
## Alpha-Beta Quiz



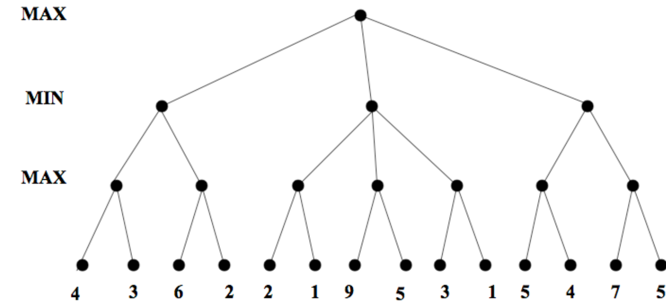

## Alpha-Beta Quiz 2



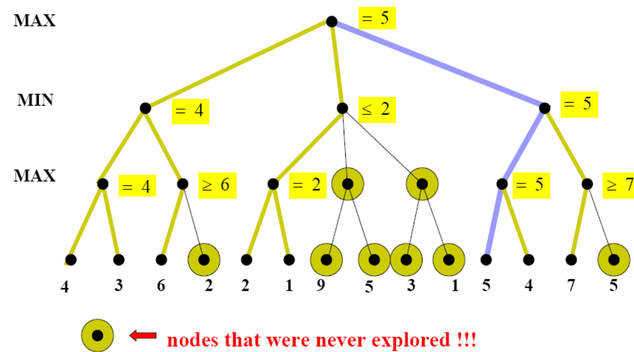
## Alpha-Beta Quiz 2




## Alpha beta pruning. Example




## Solution : Alpha Beta




## Properties of α-β

- Pruning does not affect final result. This means that it gets the exact same result as does full minimax.
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  → doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

## Good Enough?

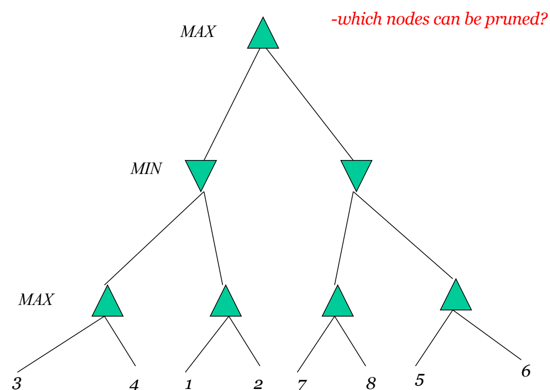
- Chess:
  - branching factor $b \approx 35$
  - game length $m \approx 100$
  - search space $b^{m/2} \approx 35^{50} \approx 10^{77}$
- The Universe:
  - number of atoms $\approx 10^{78}$
  - age $\approx 10^{18}$ seconds
  - $10^{8}$ moves/sec x $10^{78}$ x $10^{18} = 10^{104}$

The universe can play chess - can we?

## Effectiveness of Alpha-Beta Search

- Worst-Case
  - branches are ordered so that no pruning takes place. In this case alpha-beta gives no improvement over exhaustive search
- Best-Case
  - each player's best move is the left-most alternative (i.e., evaluated first)
  - in practice, performance is closer to best rather than worst-case
- In practice often get $O(b^{(d/2)})$ rather than $O(b^d)$
  - this is the same as having a branching factor of sqrt(b),
    - since $(\mathrm{sqrt}(b))^d = b^{(d/2)}$
    - i.e., we have effectively gone from b to square root of b
  - e.g., in chess go from b ~ 35 to b ~ 6
    - this permits much deeper search in the same amount of time
    - Typically twice as deep.

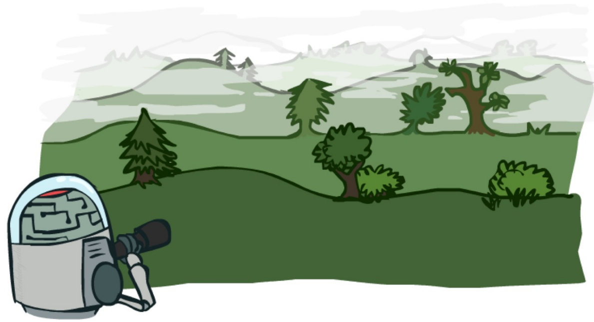
## Example




## Final Comments about Alpha-Beta Pruning

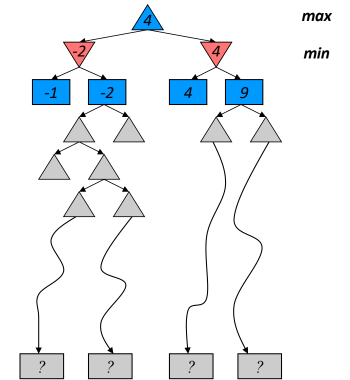
- Pruning does not affect final results
- Entire subtrees can be pruned.
- Good move *ordering* improves effectiveness of pruning
- Repeated states are again possible.
  - Store them in memory = transposition table

# Resource Limits



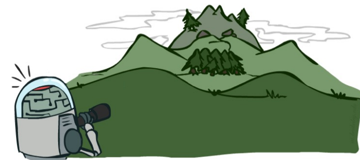
# Resource Limits

- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm




# Depth Matters

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation



*[Demo: depth limited (L6D4, L6D5)]*

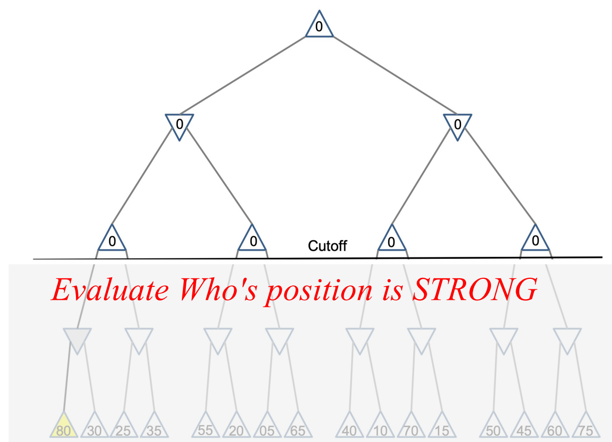
# Video of Demo Limited Depth (2)

## Video of Demo Limited Depth (10)

## Cutting off Search

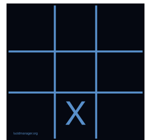
- We cannot search till leaf
- MinimaxCutoff is identical to MinimaxValue except
  1. Terminal? is replaced by Cutoff?
  2. Utility is replaced by Eval
- Does it work in practice?
  - $b^m$ = 106, b=35 →m=4
- 4-ply lookahead is a hopeless chess player!
  - 4-ply ≈ human novice
  - 8-ply ≈ typical PC, human master
  - 12-ply ≈ Deep Blue, Kasparov

## Cutting off Search




## Evaluation Functions

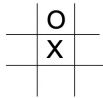
- Evaluation functions score non-terminals in depth-limited search

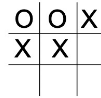



- Let p be a position in the game
- Define the utility function f(p) by
  - f(p) =
    - largest positive number if p is a win for computer
    - smallest negative number if p is a win for opponent
    - RCDC – RCDO
  - where RCDC is number of rows, columns and diagonals in which computer could still win
  - and RCDO is number of rows, columns and diagonals in which opponent could still win.

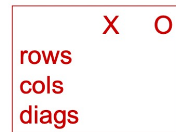
## Sample Evaluations

- X = Computer; O = Opponent



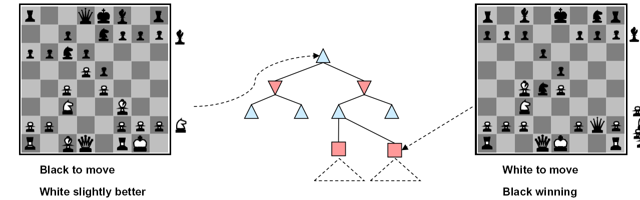




- X: 6 Win
- O: 4 win




## Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search




- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- e.g. $w_1$=9 with
  - $f_1(s)$ = (num white queens – num black queens), etc.

## Example: Samuel's Checker-Playing Program

- It uses a linear evaluation function

  $f(n) = w_1 f_1(n) + w_2 f_2(n) + ... + w_m f_m(n)$
- For example: f = 6 K + 4 M + U
  - K = King Advantage, M = Man Advantage
  - U = Undenied Mobility Advantage (number of moves that Max where Min has no jump moves)
- In learning mode
  - Computer acts as 2 players: A and B
  - A adjusts its coefficients after every move
  - B uses the static utility function
  - If A wins, its function is given to B

## Samuel's Checker Player

- How does A change its function?
- Coefficent replacement
  - $\Delta$(node) = backed-up value(node) – initial value(node)
    - if $\Delta > 0$ then terms that contributed positively are given more weight and terms that contributed negatively get less weight
    - if $\Delta < 0$ then terms that contributed negatively are given more weight and terms that contributed positively get less weight

## Chess: Rich history of cumulative ideas

- Minimax search, evaluation function learning (1950).
- Alpha-Beta search (1966).
- Transposition Tables (1967).
- Iterative deepening DFS (1975).
- End game data bases ,singular extensions(1977, 1980)
- Parallel search and evaluation(1983 ,1985)
- Circuitry (1987)

## (Durdarshi) दूरदर्शी / horizon effect

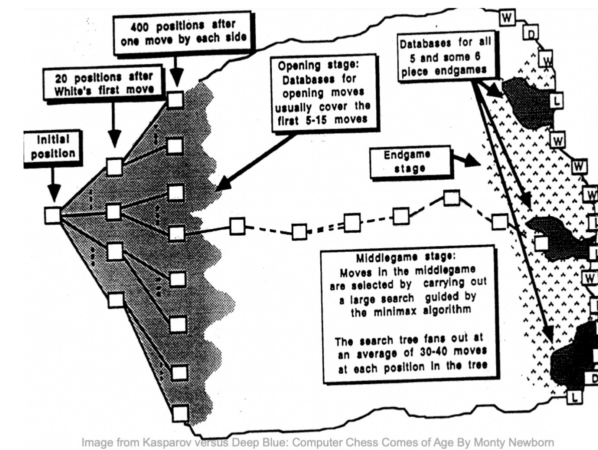
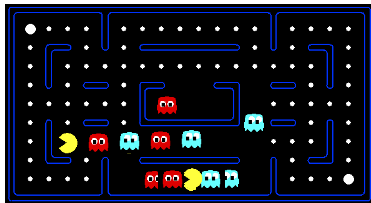


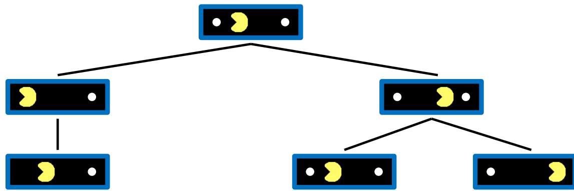
Image from Kasparov versus Deep Blue: Computer Chess Comes of Age By Monty Newborn

## Evaluation for Pacman



## Video of Demo Thrashing (d=2)

## Why Pacman Starves



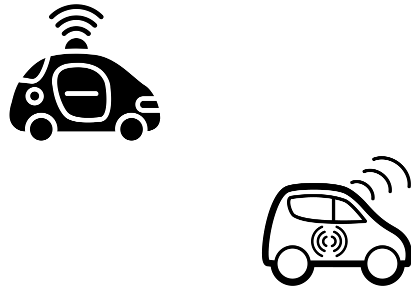
- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

## Video of Demo Thrashing -- Fixed (d=2)

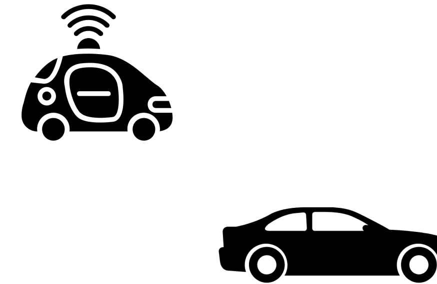
## Video of Demo Smart Ghosts (Coordination)

## Video of Demo Smart Ghosts (Coordination) – Zoomed In

## Agents Getting Along with Other Agents

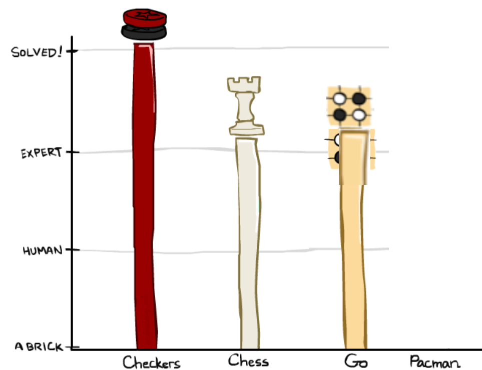


## Agents Getting Along with Humans



## Status of AI Game

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Go :2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search, learned evaluation function.**
- **Pacman**

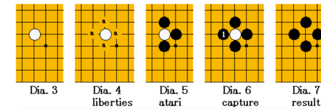



## Imperfect Information

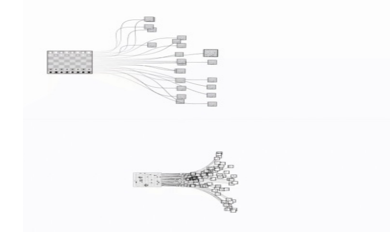
- E.g. card games, where opponents' initial cards are unknown
- Idea: For all deals consistent with what you can see
  - –compute the minimax value of available actions for each of possible deals
  - –compute the expected value over all deals

# Status of AI Game Players

- Tic Tac Toe
  - Tied for best player in world
- Othello
  - Computer better than any human
  - Human champions now refuse to play computer
- Scrabble
  - Maven beat world champions Joel Sherman and Matt Graham
- Backgammon
  - 1992, Tesauro combines 3-ply search & neural networks (with 160 hidden units) yielding top-3 player
- Bridge
  - Gib ranked among top players in the world
- Poker
  - 2015, Heads-up limit hold'em poker is solved
- Checkers
  - 1994, Chinook ended 40-year reign of human champion Marion Tinsley
- Chess
  - 1997, Deep Blue beat human champion Gary Kasparov in sixgame match
  - Deep Blue searches 200M positions/second, up to 40 ply
  - Now looking at other applications (molecular dynamics, drug synthesis)
- Go
  - 2016, Deepmind's AlphaGodefeated Lee Sedol & 2017 defeated Ke Jie




# Go Story



| Game size | Board size N | $3^N$ | Percent legal | legal game positions (A094777)[11] |
|---|---|---|---|---|
| 1×1 | 1 | 3 | 33% | 1 |
| 2×2 | 4 | 81 | 70% | 57 |
| 3×3 | 9 | 19,683 | 64% | 12,675 |
| 4×4 | 16 | 43,046,721 | 56% | 24,318,165 |
| 5×5 | 25 | $8.47 \times 10^{11}$ | 49% | $4.1 \times 10^{11}$ |
| 9×9 | 81 | $4.4 \times 10^{38}$ | 23.4% | $1.039 \times 10^{38}$ |
| 13×13 | 169 | $4.3 \times 10^{80}$ | 8.66% | $3.72497923 \times 10^{79}$ |
| 19×19 | 361 | $1.74 \times 10^{172}$ | 1.196% | $2.08168199382 \times 10^{170}$ |

Till 2015 : human champions refused to compete against computers, because software used to be too bad
AlphaGo (2016)

Mastering the game of Go with Deep Neural Networks & Tree Search
Silver et al, 2016
AlphaGo

Mastering the game of Go without Human Knowledge
Silver et al, 2017
AlphaGo Zero

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play
Silver et al, 2018
AlphaZero

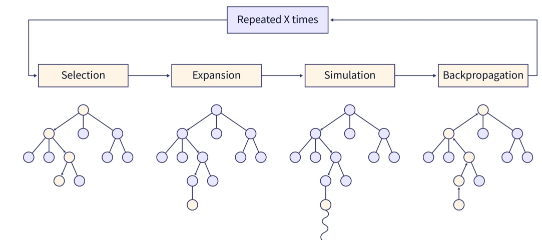
Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model
Schrittwieser et al, 2020
**MuZero**

**MuZero** with Self-competition for Rate Control in VP9 Video Compression
Mandhane et al, 2022

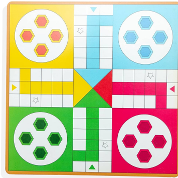
https://www.youtube.com/watch?v=c8SLNEpFSrs

# Other Game Methods

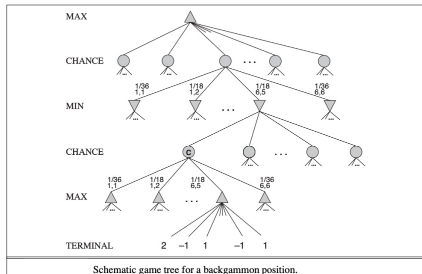
- **Monte Carlo Tree Search (MCTS)**
  - Evaluates states not by applying a heuristic function, but by playing out the game all the way to the end and using a rules of that game to see who won.
  - The value of the state is estimated as the average utility over a number of **simulations** of complete games starting from the state.
  - Four steps
    - Selection
    - Expansion
    - Simulation
    - Back-propagation



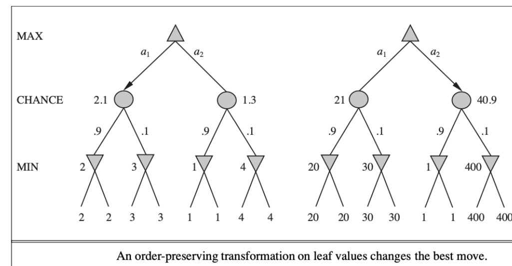
# Other Game Methods



- **Stochastic Games**
  - Many unpredictable external events can put us into unforeseen situations.
  - Many games mirror this unpredictability by including a random element, such as the throwing of dice
    - Backgammon is a typical game that combines luck and skill. Dice are rolled at the beginning of a player's turn to determine the legal moves.




Schematic game tree for a backgammon position.




An order-preserving transformation on leaf values changes the best move.

# Summary

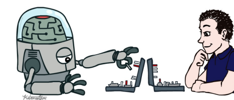
- Games are fun to work on!
- They illustrate several important points about AI.
- Perfection is unattainable →must approximate.
- Game playing programs have shown the world what AI can do.
- Incorporate Heuristics in Game Trees
- Perform Best First Search in Game Trees
- Multi-Player Games for more than two players
- Team Games – Cooperation and Competition
- Probabilistic Games
- Real Life Situations
  - Economics
  - Reactive Control Systems
  - Autonomous Systems

# Next

- Module 3: Search Strategies
  - PART 3.1: Search
  - PART 3.2: Uninformed Search
  - PART 3.3: Informed/Heuristic Search
  - PART 3.4: Beyond Classical Search
    - Local Search
      - Generate-and-test
      - Hill climbing
      - Simulated Analing
    - Problem reduction
  - PART 3.5: Adversarial Search
  - PART 3.6: Constraint Satisfaction Problems

# References

- ***Artificial Intelligence** by Elaine Rich & Kevin Knight*, Third Ed, Tata McGraw Hill
- ***Artificial Intelligence and Expert System** by Patterson*
- http://www.cs.rmit.edu.au/AI-Search/Product/
- http://aima.cs.berkeley.edu/demos.html (for more demos)
- ***Artificial Intelligence and Expert System** by Patterson*
- Slides adapted from CS188 Instructor: Anca Dragan, University of California, Berkeley
- Slides adapted from CS60045 ARTIFICIAL INTELLIGENCE



*(some slides adapted from http://aima.cs.berkeley.edu/)*