

# Artificial Intelligence

CS 165A

May 4, 2023

Instructor: Prof. Yu-Xiang Wang

T  
o  
d  
a  
y

- Games and minimax search
- Midterm review

# Recap: Informed Search

- How to use a heuristic function to improve search
  - Greedy Best-first search + Uniform-cost search = A\* Search
- When is A\* search optimal?
  - h is Admissible (optimistic) for Tree Search
  - h is Consistent for Graph Search
- Choosing heuristic functions
  - A good heuristic function can reduce time/space cost of search by orders of magnitude.
  - Good heuristic function may take longer to evaluate.

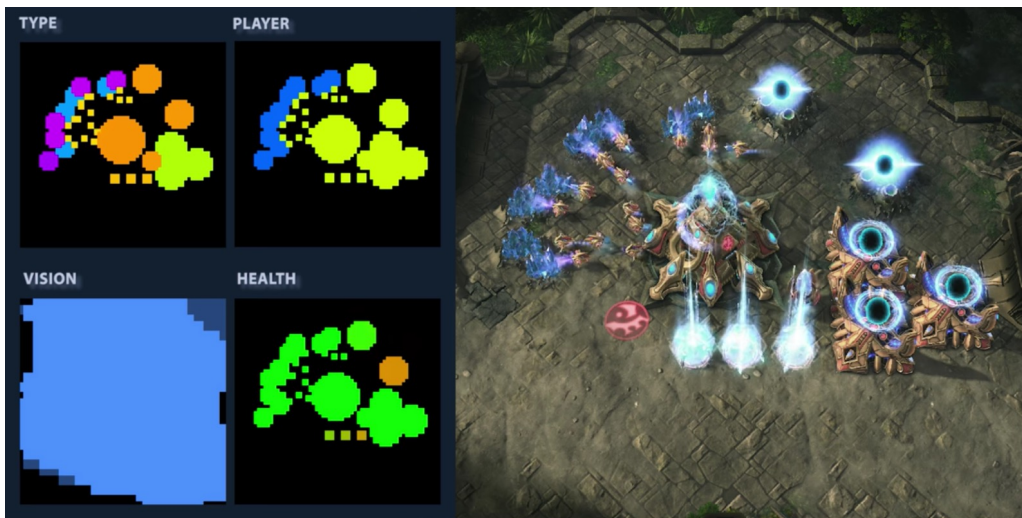
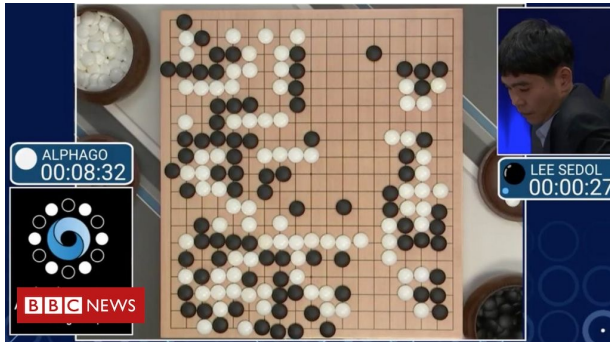
# This lecture

- Games and Adversarial Search
- Midterm Review

# Games and Adversarial Search



- Games: problem setup
- Minimax search
- Alpha-beta pruning



# Illustrative example of a simple game (1 min discussion)



## Example: game 1

You choose one of the three bins.  
I choose a number from that bin.  
Your goal is to maximize the chosen number.

① adversary  $\rightarrow$  B  
② 0.5 chance of either number  
③ Mom  $\rightarrow$  A C

0

**A**

-50    50

$$0.5 \cdot 1 + 0.5 \cdot 3 = 2$$

**B**

1    3

$$-5 \cdot 0.5 + 15 \cdot 0.5 = 5$$

**C**

-5    15

(Example taken from Liang and Sadigh)

# Game as a search problem

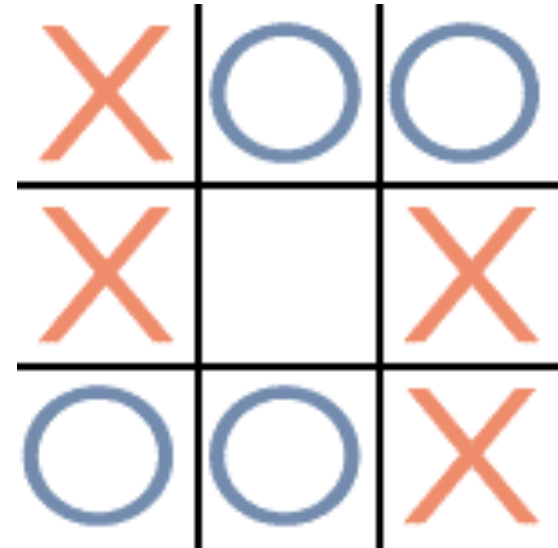
- $S_0$  The initial state
- PLAYER(s): Returns which player has the move
- ACTIONS(s): Returns the legal moves.
- RESULT(s, a): Output the state we transition to.
- TERMINAL-TEST(s): Returns True if the game is over.
- UTILITY(s,p): The payoff of player p at terminal state s.

# Two-player, Turn-based, Perfect information, Deterministic, Zero-Sum Game

- Two-player: Tic-Tac-Toe, Chess, Go!
- Turn-based: The players take turns in round-robin fashion.
- Perfect information: The State is known to everyone
- Deterministic: Nothing is random
- Zero-sum: The total payoff for all players is a **constant**.
  - *The 8-puzzle is a one-player, perfect info, deterministic, zero-sum game.*
  - *How about Rock-Paper-Scissors?*
  - *How about Monopoly?*
  - *How about Starcraft?*

# Tic-Tac-Toe

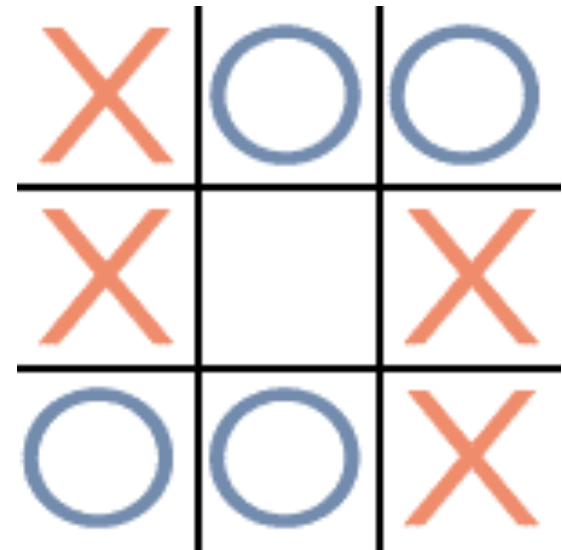
- The first player is **X** and the second is **O**
- Object of game: get three of your symbol in a horizontal, vertical or diagonal row on a 3x3 game board
- **X** always goes first
- Players alternate placing **Xs** and **O**s on the game board
- Game ends when a player has three in a row (a wins) or all nine squares are filled (a draw)





# 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 3x3 game board
- **X** always goes first
- Players alternate placing **Xs** 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)



What's the state, action, transition, payoff for Tic-Tac-Toe?



# 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!

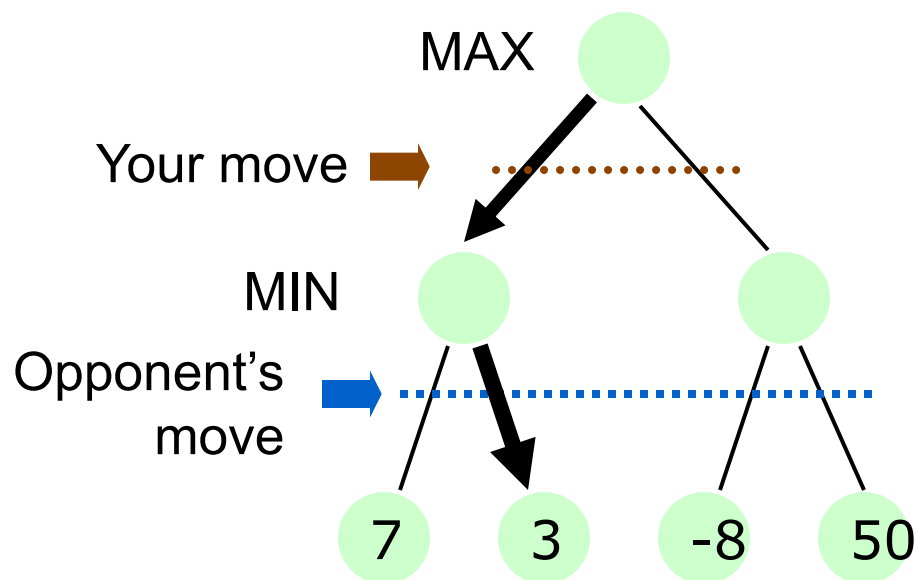
# Two players: MIN and MAX

- In a zero-sum game:
  - payoff to Player 1 = - payoff to Player 2
- The goal of Player 1 is to maximizing her payoff.
- The goal of Player 2 is to maximizing her payoff as well
  - Equivalent to minimizing Player 1's payoff.

# Minimax search

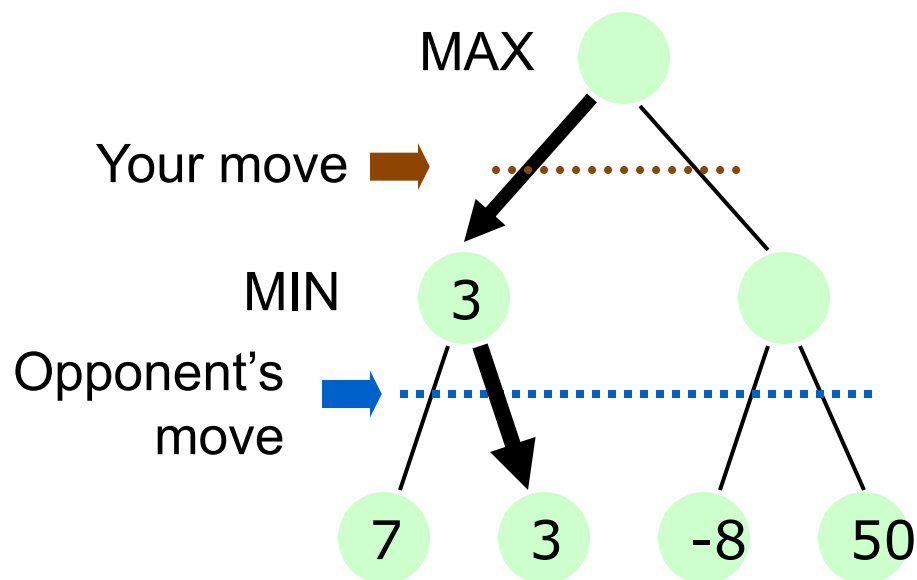
- Assume that both players play perfectly
  - do not assume player will miss good moves or make mistakes
- Score(s): The score that MAX will get towards the end if both player play perfectly from s onwards.
- Consider MIN's strategy
  - MIN's best strategy:
    - choose the move that **minimizes** the score that will result when MAX chooses the **maximizing** move
  - MAX does the opposite

# Minimaxing



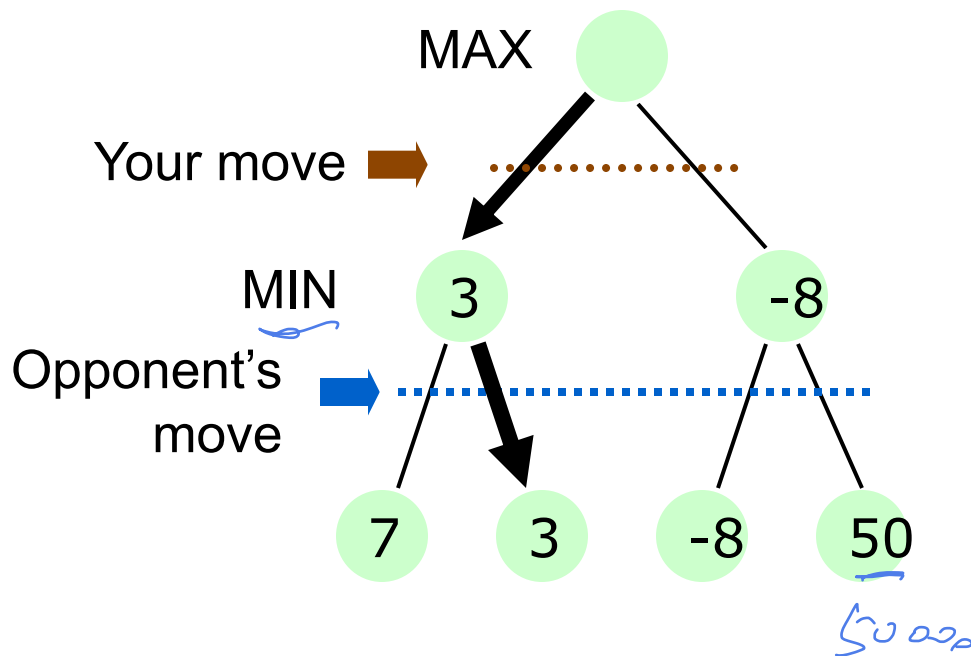
- Your opponent will choose smaller numbers
- If you move left, your opponent will choose 3
- If you move right, your opponent will choose -8
- Thus your choices are only 3 or -8
- You should move left

# 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

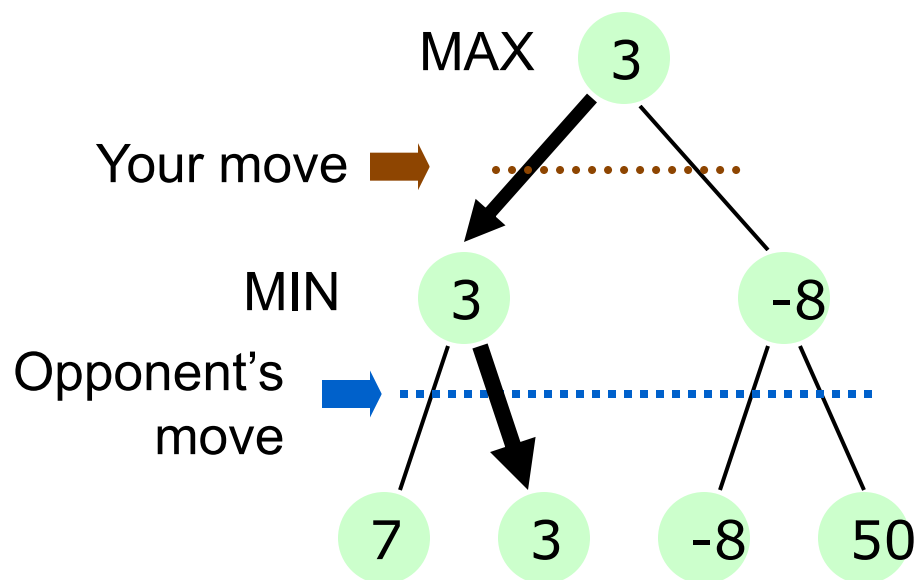
# Minimaxing



- 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

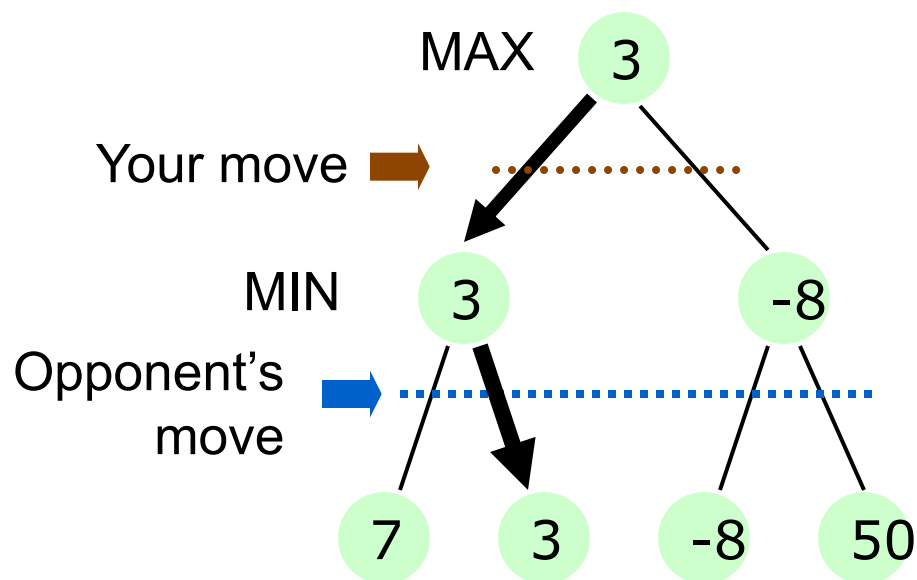


# Minimaxing



- 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

# Minimaxing

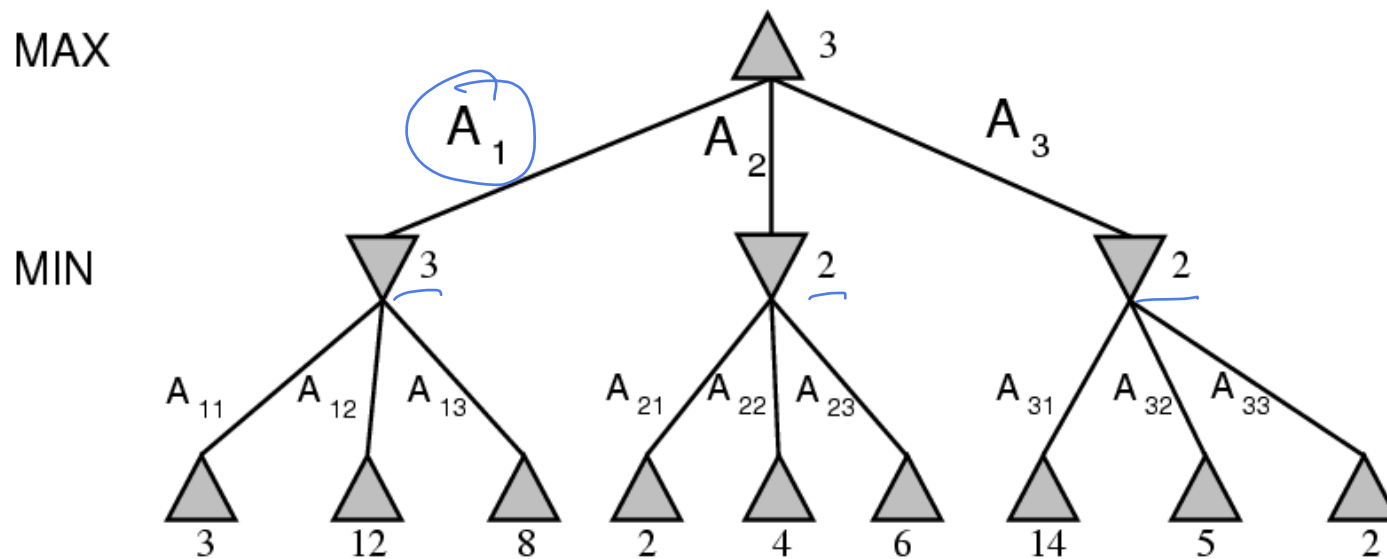


- Your opponent will choose smaller numbers
- If you move left, your opponent will choose 3
- If you move right, your opponent will choose -8
- Thus your choices are only 3 or -8
- You should move left

Each move is called a “ply”. One round is K-pplies for a K-player game.

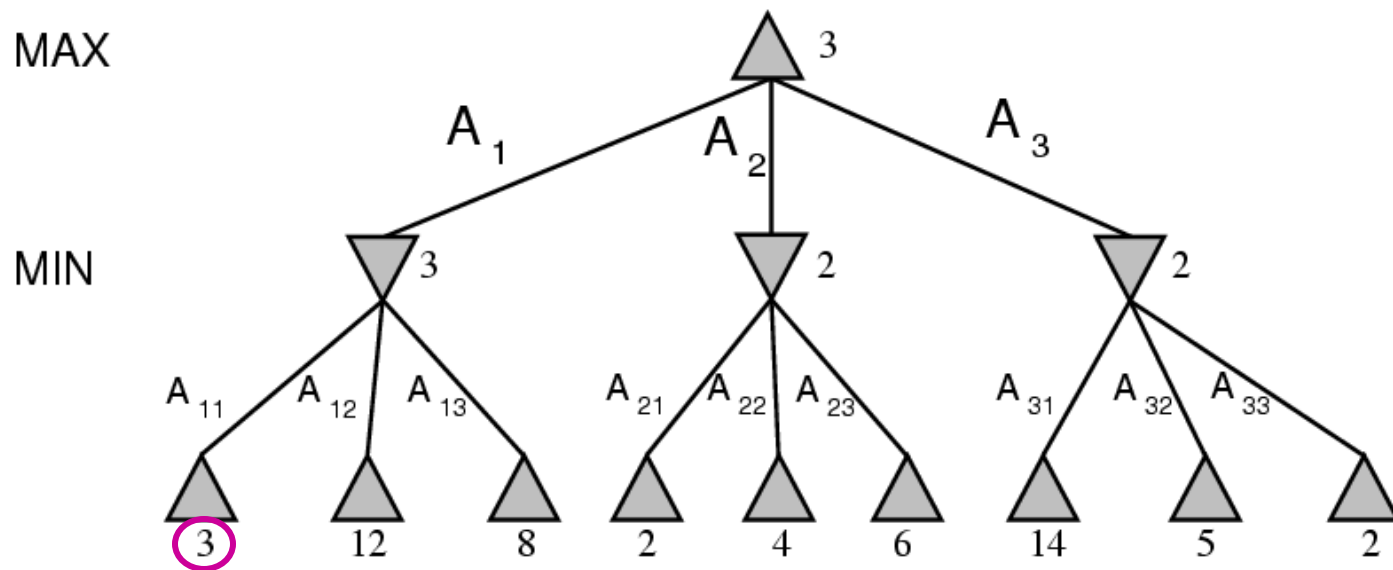
# Minimax example

Which move to choose?



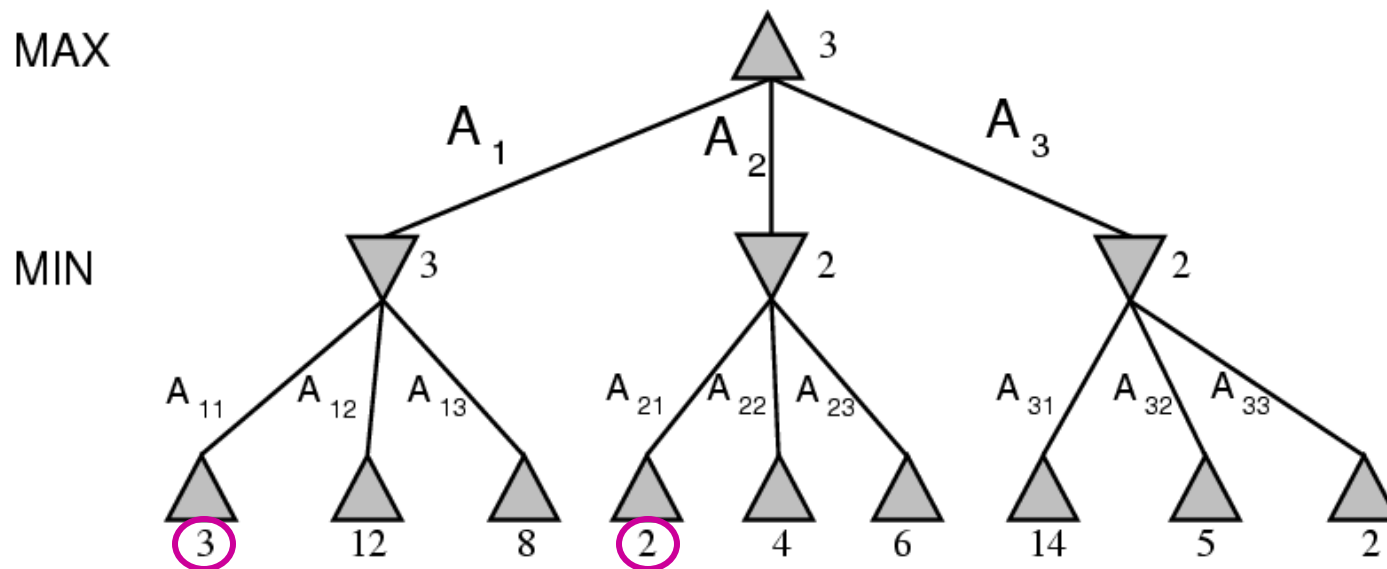
# Minimax example

Which move to choose?



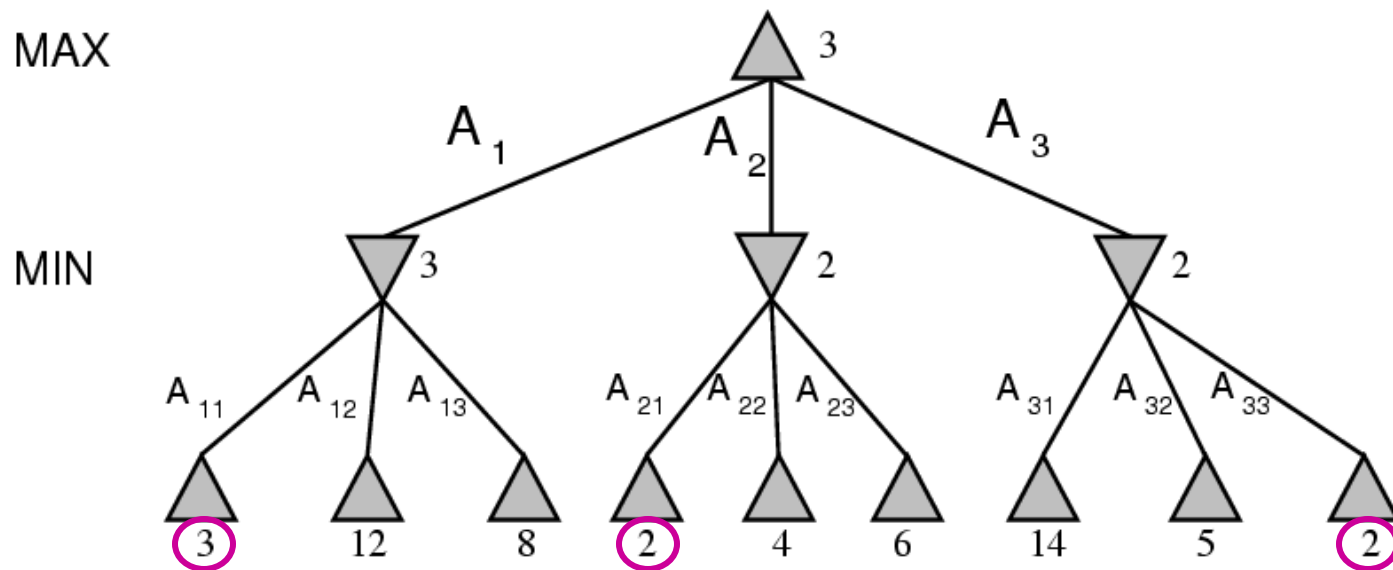
# Minimax example

Which move to choose?



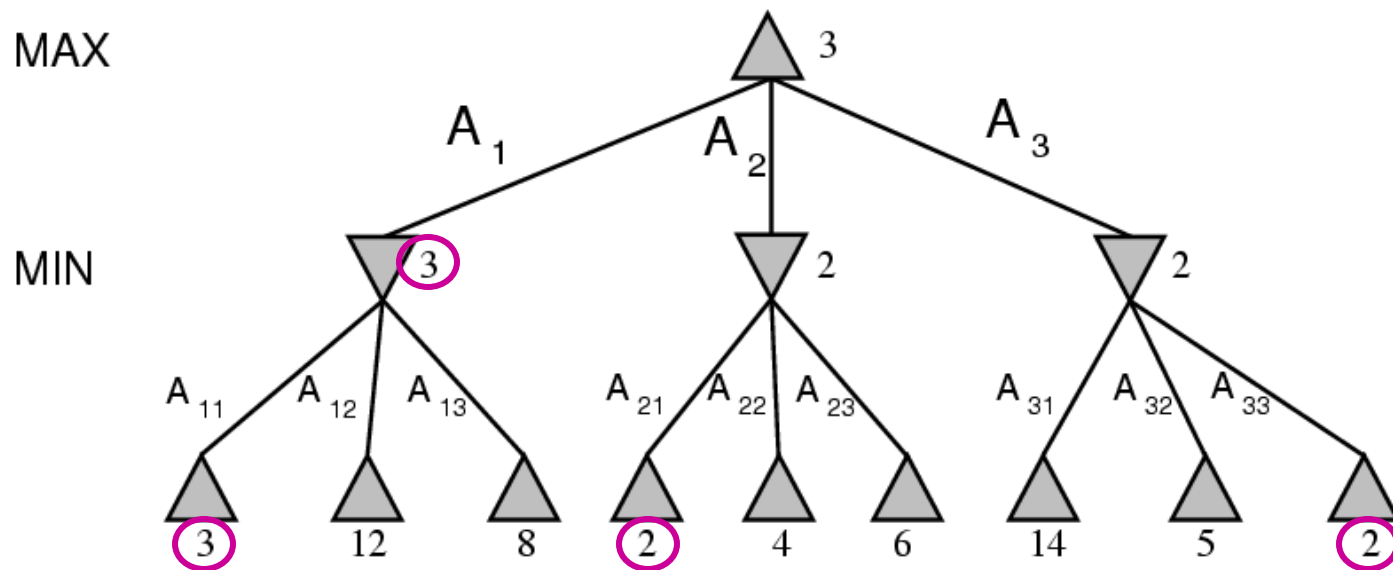
# Minimax example

Which move to choose?



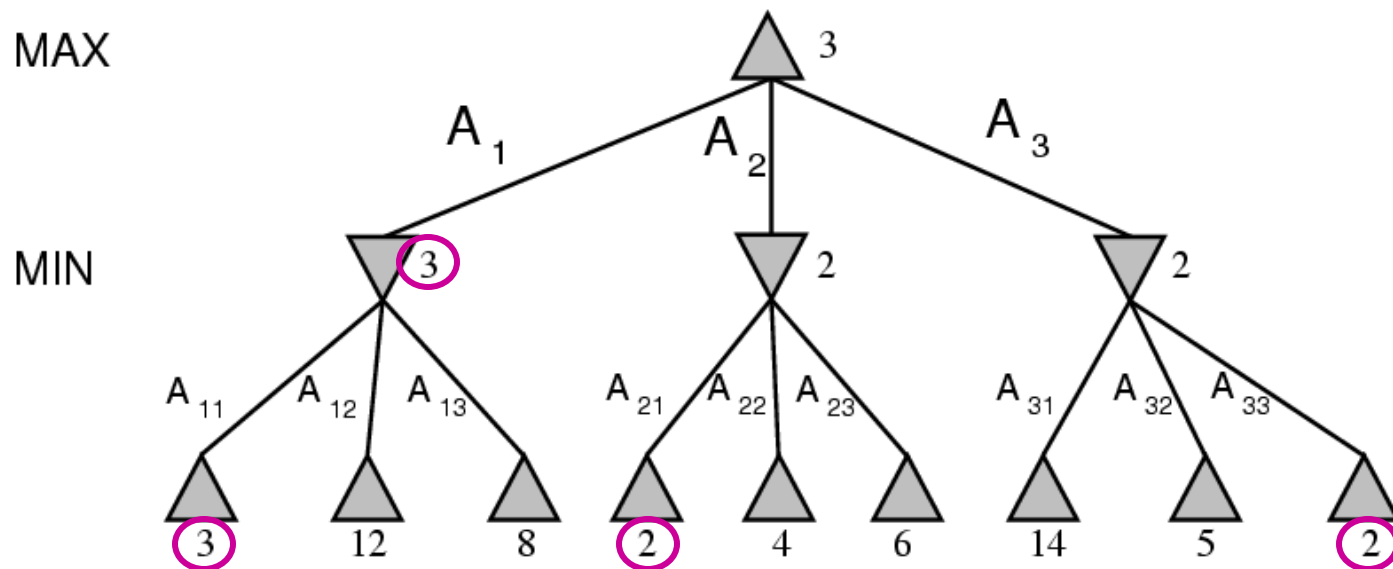
# Minimax example

Which move to choose?



# Minimax example

Which move to choose?

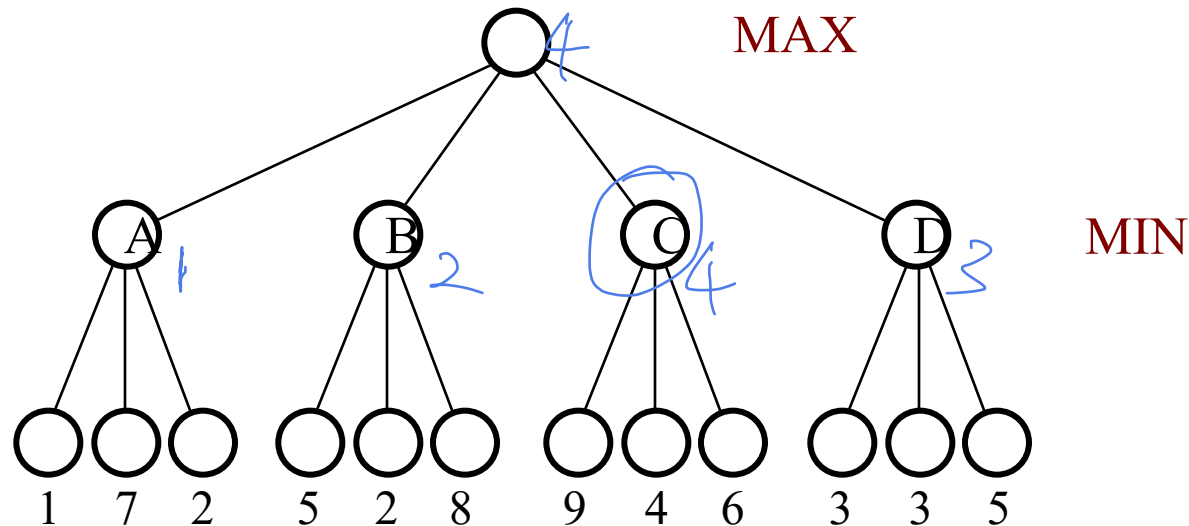


The **minimax decision** is move  $A_1$



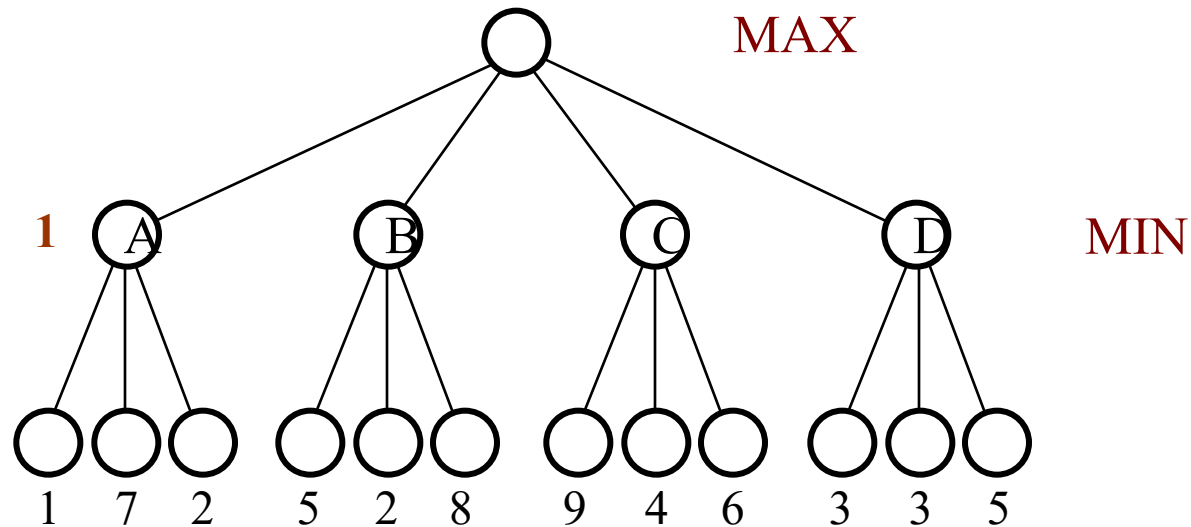
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



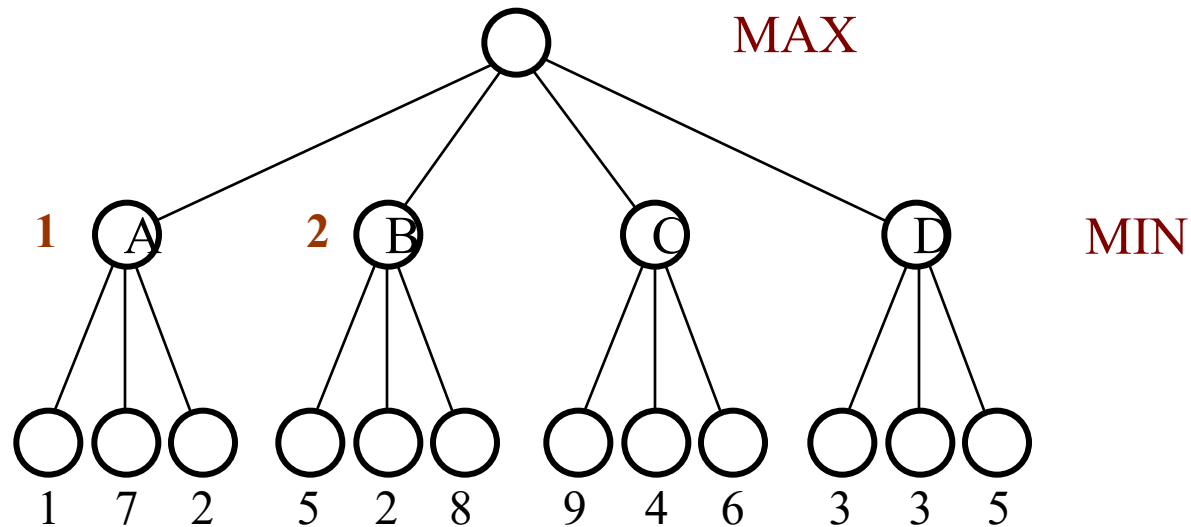
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



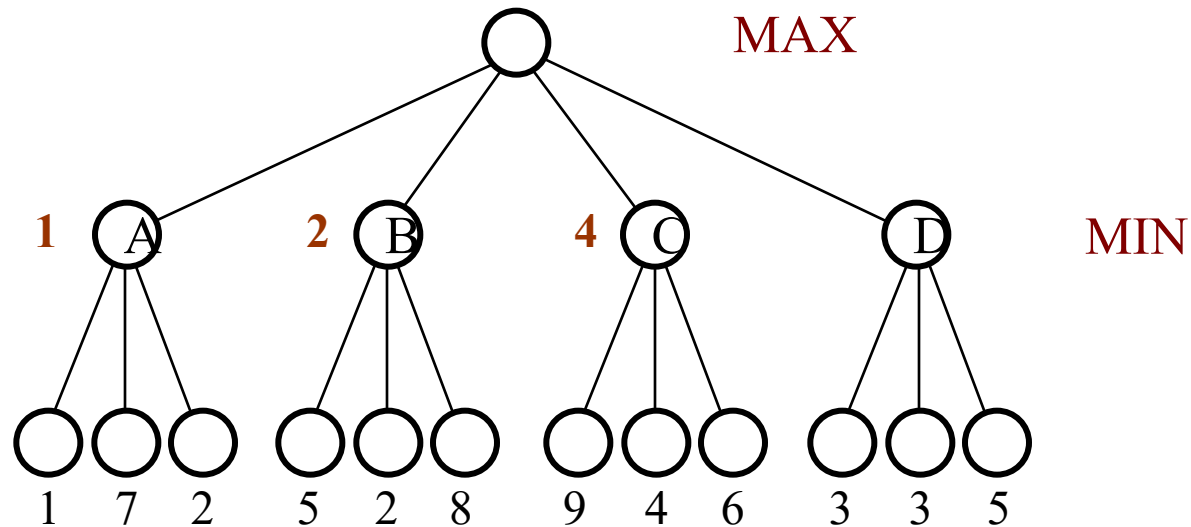
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



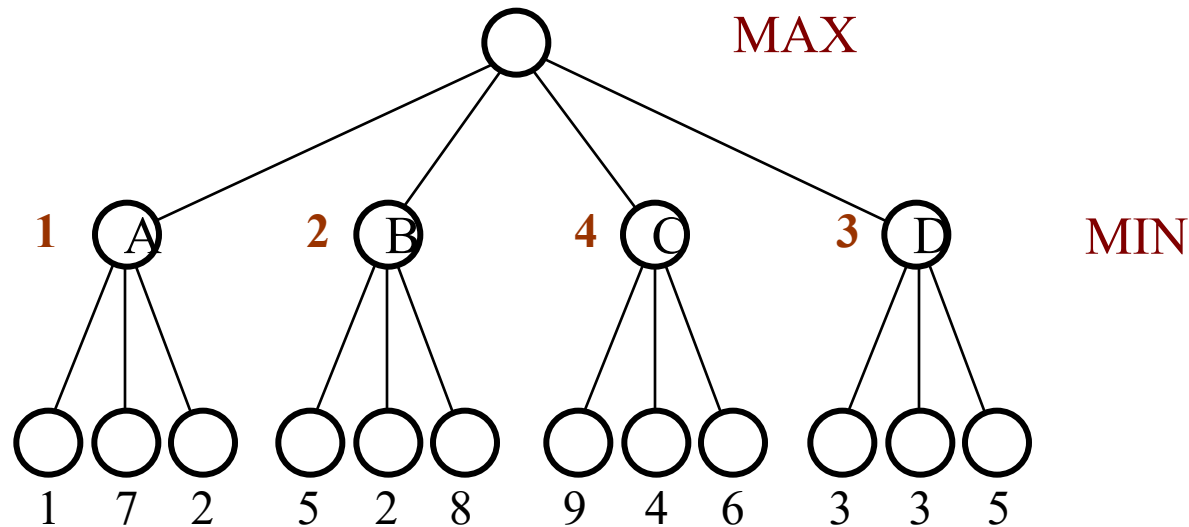
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



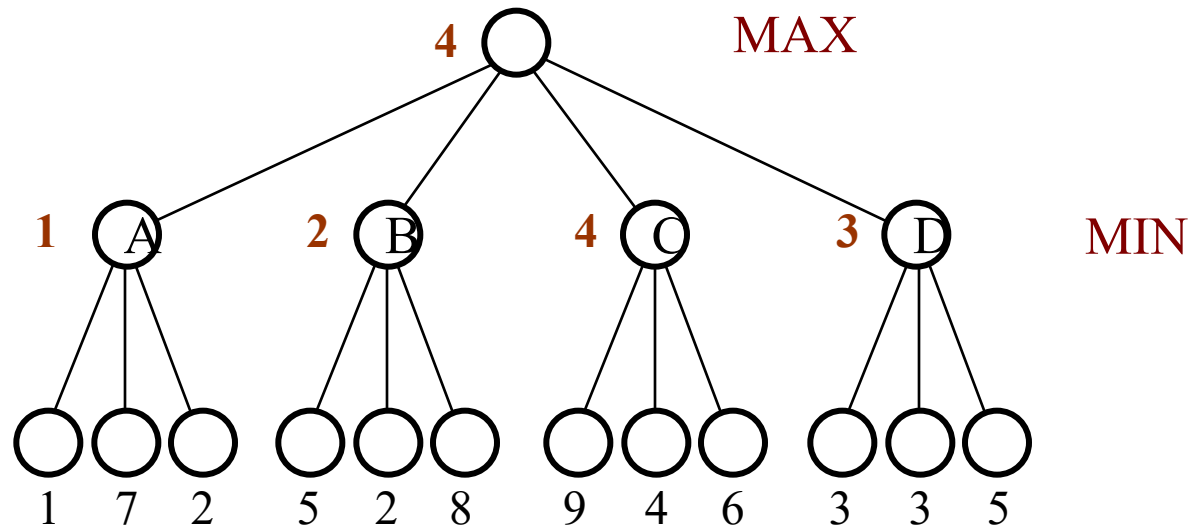
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



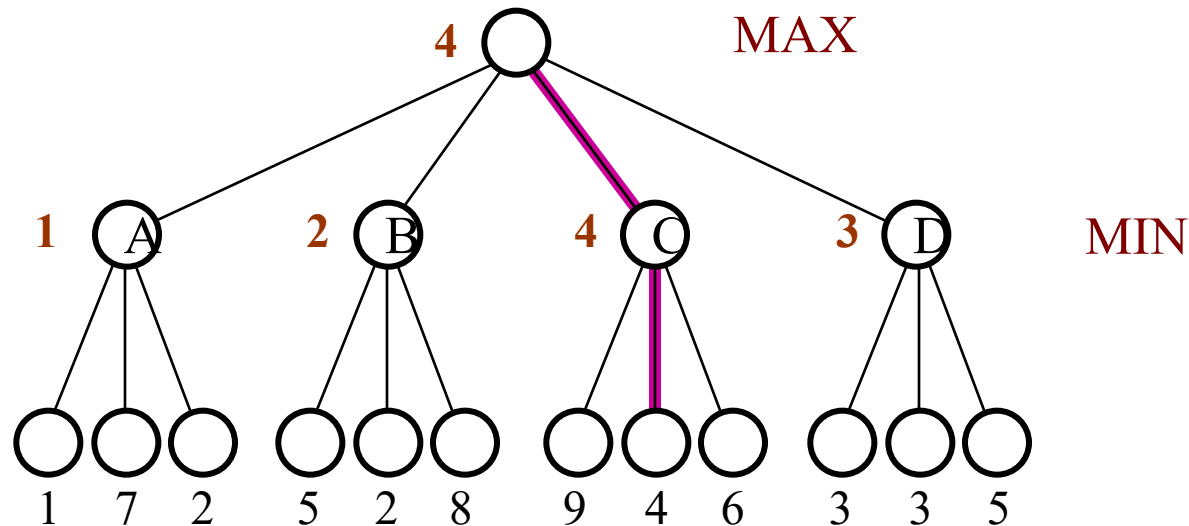
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



# Minimax search



# Minimax search

- The *minimax decision* maximizes the utility under the assumption that the opponent seeks to minimize it (if it uses the same evaluation function)

# Minimax search

- The *minimax decision* maximizes the utility under the assumption that the opponent seeks to minimize it (if it uses the same evaluation function)
- Generate the tree of minimax values
  - Then choose best (maximum) move
  - Don't need to keep all values around
    - Good memory property

# Minimax search

- The *minimax decision* maximizes the utility under the assumption that the opponent seeks to minimize it (if it uses the same evaluation function)
- Generate the tree of minimax values
  - Then choose best (maximum) move
  - Don't need to keep all values around
    - Good memory property
- Depth-first search is used to implement minimax
  - Expand all the way down to leaf nodes
  - Recursive implementation

# Minimax properties

- Optimal? Yes, against an optimal opponent, **if** the tree is finite
- Complete? Yes, **if** the tree is finite
- Time complexity? Exponential:  **$O(b^m)$**
- Space complexity? Polynomial:  **$O(bm)$**

# But this could take forever...

- Exact search is intractable
  - Tic-Tac-Toe is  $9! = 362,880$
  - For chess,  $b \approx 35$  and  $m \approx 100$  for “reasonable” games
  - Go is  $361! \approx 10^{750}$

# But this could take forever...

- Exact search is intractable
  - Tic-Tac-Toe is  $9! = 362,880$
  - For chess,  $b \approx 35$  and  $m \approx 100$  for “reasonable” games
  - Go is  $361! \approx 10^{750}$
- Idea 1: Pruning

# But this could take forever...

- Exact search is intractable
  - Tic-Tac-Toe is  $9! = 362,880$
  - For chess,  $b \approx 35$  and  $m \approx 100$  for “reasonable” games
  - Go is  $361! \approx 10^{750}$
- Idea 1: Pruning
- Idea 2: Cut off early and use a heuristic function

# Pruning

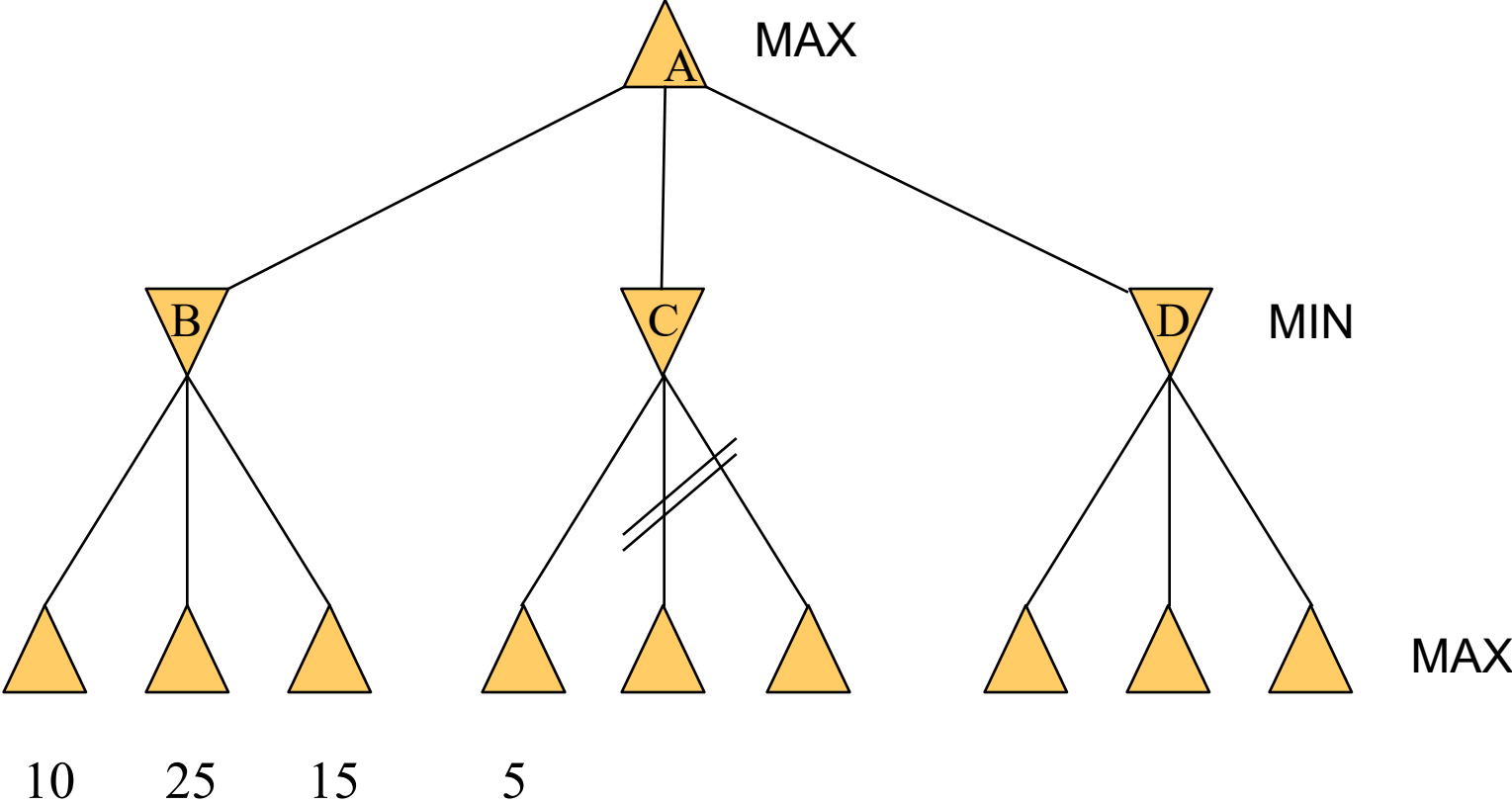
- What's really needed is “smarter,” more efficient search
  - Don't expand “dead-end” nodes!
- **Pruning** – eliminating a branch of the search tree from consideration



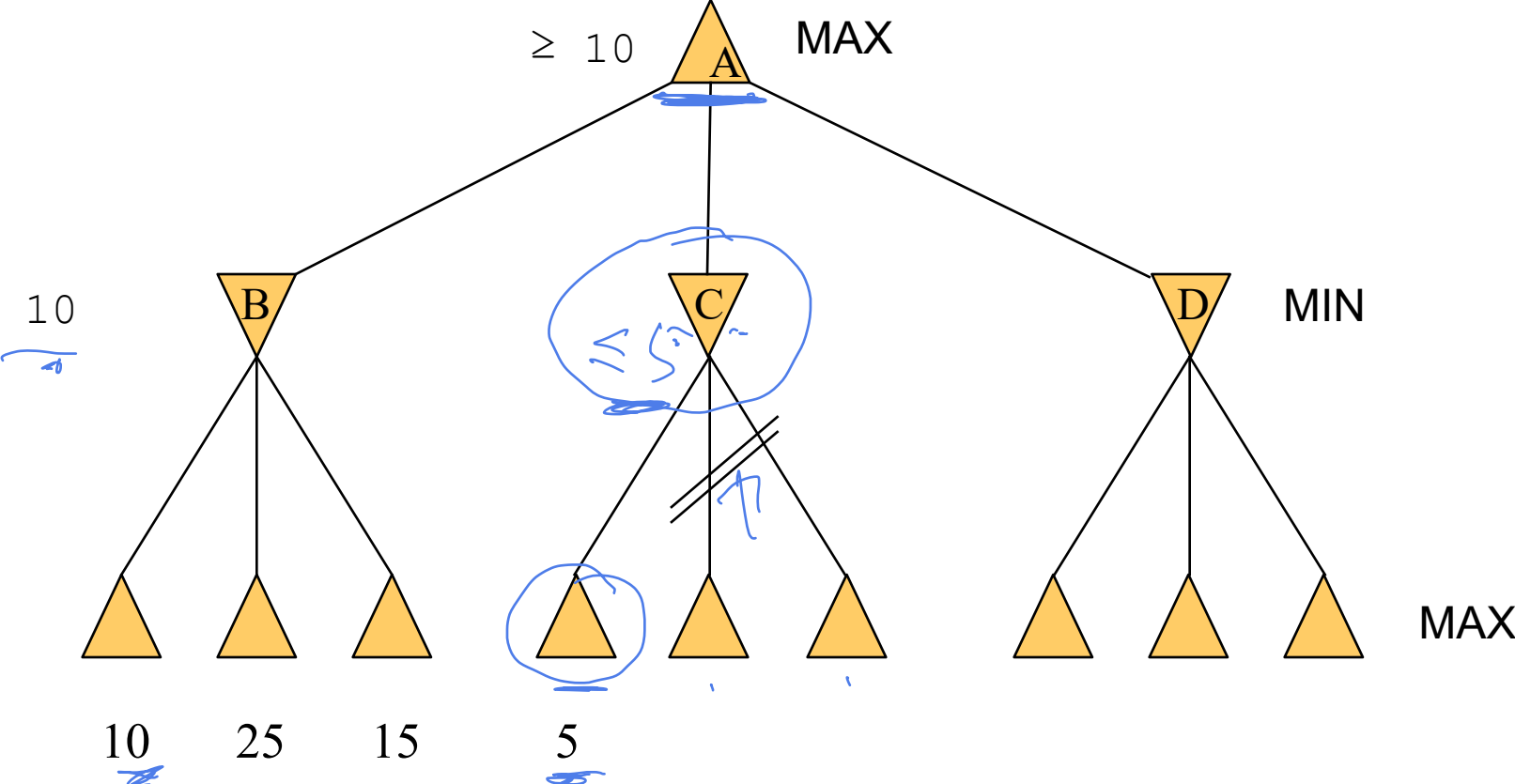
# Pruning

- What's really needed is “smarter,” more efficient search
  - Don't expand “dead-end” nodes!
- **Pruning** – eliminating a branch of the search tree from consideration
- **Alpha-beta pruning**, applied to a minimax tree, returns the same “best” move, while pruning away unnecessary branches
  - Many fewer nodes might be expanded
  - Hence, smaller effective branching factor
  - ...and deeper search
  - ...and better performance
    - Remember, minimax is *depth-first* search

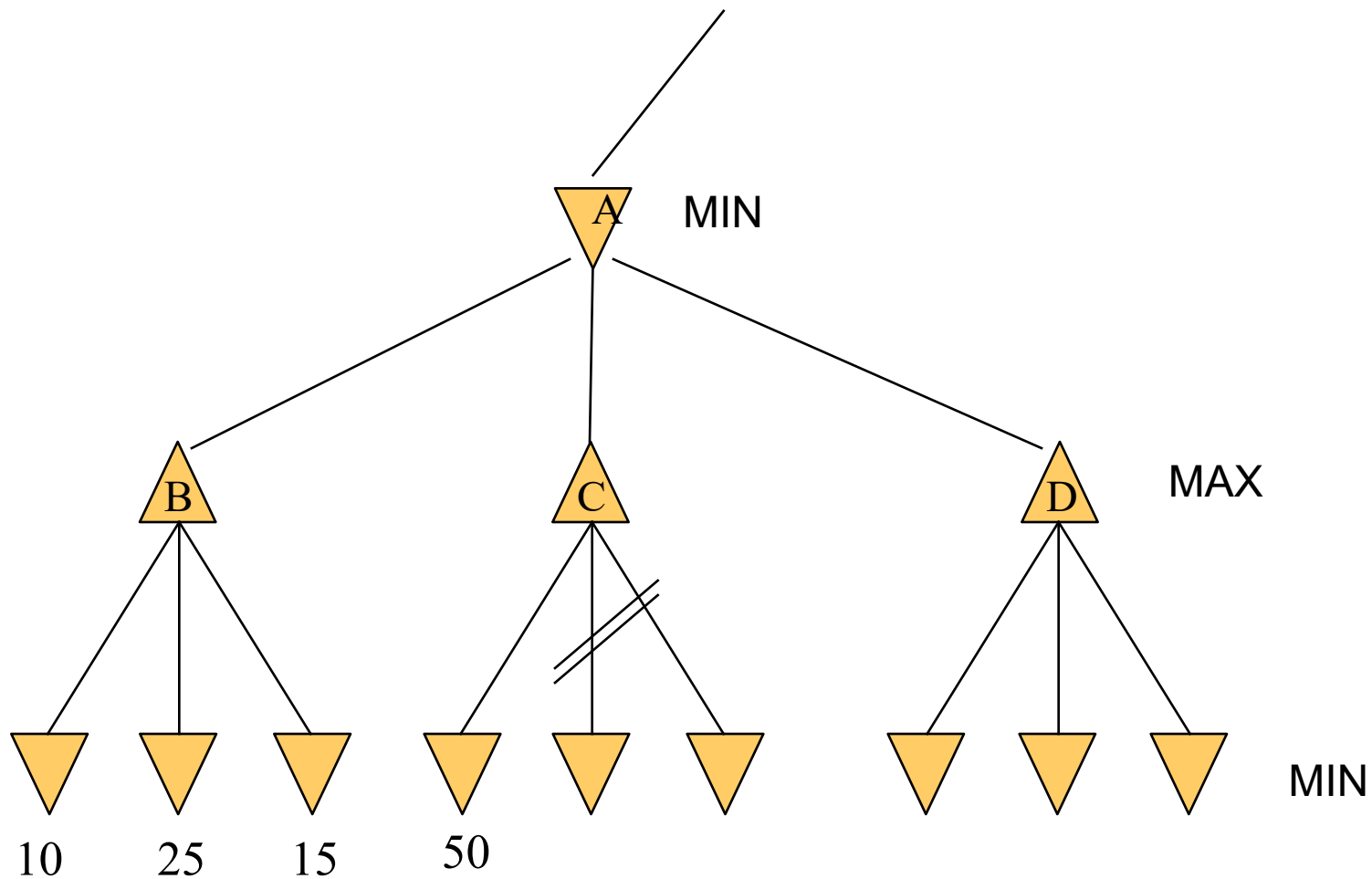
# Alpha pruning



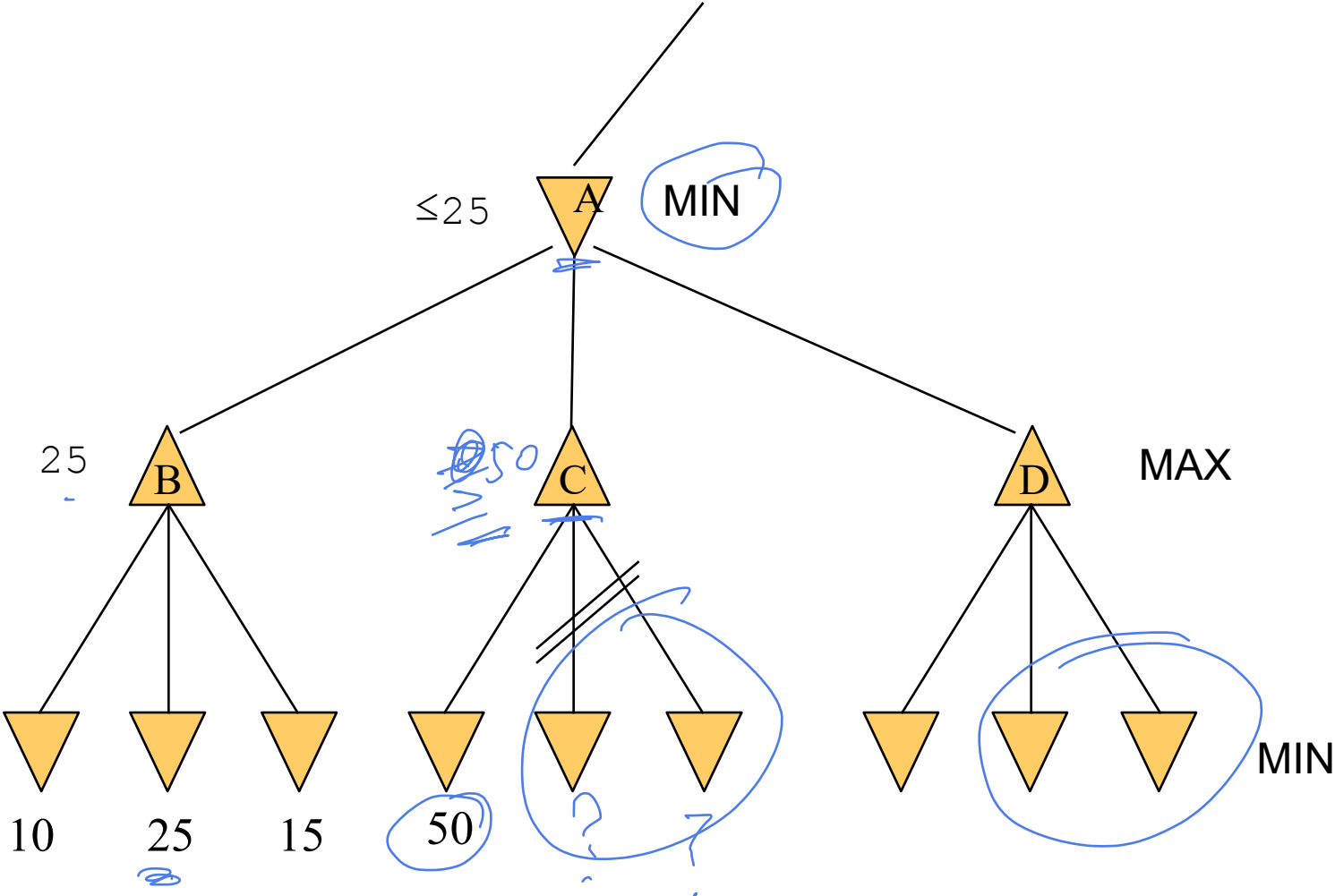
# Alpha pruning



# Beta pruning



# Beta pruning



# Improvements via alpha/beta pruning

# Improvements via alpha/beta pruning

- Depends on the ordering of expansion

# Improvements via alpha/beta pruning

- Depends on the ordering of expansion
- Perfect ordering  $O(b^{m/2})$



# Improvements via alpha/beta pruning

- Depends on the ordering of expansion
- Perfect ordering  $O(b^{\underline{m/2}})$
- Random ordering  $O(b^{3m/4})$

# Improvements via alpha/beta pruning

- Depends on the ordering of expansion
- Perfect ordering  $O(b^{m/2})$
- Random ordering  $O(b^{3m/4})$
- For specific games like Chess, you can get to almost perfect ordering.

# Heuristic (Evaluation function)

- It is usually impossible to solve games completely
- Rather, cut the search off early and apply a heuristic evaluation function to the leaves
  - $h(s)$  estimates the expected utility of the game from a given position (node/state)  $s$
  - like depth bounded depth first, lose completeness
  - Explore game tree using combination of evaluation function and search

# Heuristic (Evaluation function)

- It is usually impossible to solve games completely
- Rather, cut the search off early and apply a heuristic evaluation function to the leaves
  - $h(s)$  estimates the expected utility of the game from a given position (node/state)  $s$
  - like depth bounded depth first, lose completeness
  - Explore game tree using combination of evaluation function and search
- The performance of a game-playing program depends on the quality (and speed!) of its evaluation function

# Heuristics (Evaluation function)

- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1f_1(s) + w_2f_2(s) + \dots + w_df_d(s)$
  - *weights* • *features* [dot product]

# Heuristics (Evaluation function)

- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1f_1(s) + w_2f_2(s) + \dots + w_df_d(s)$
  - *weights* · *features* [dot product]
- For example, in chess
  - $W = \{ 1, 3, 3, 5, 8 \}$
  - $F = \{ \# \text{ pawns advantage, \# bishops advantage, \# knights advantage, \# rooks advantage, \# queens advantage } \}$

# Heuristics (Evaluation function)

- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1f_1(s) + w_2f_2(s) + \dots + w_df_d(s)$
  - *weights* · *features* [dot product]
- For example, in chess
  - $W = \{ 1, 3, 3, 5, 8 \}$
  - $F = \{ \# \text{ pawns advantage, } \# \text{ bishops advantage, } \# \text{ knights advantage, } \# \text{ rooks advantage, } \# \text{ queens advantage} \}$
  - Is this what Deep Blue used?

# Heuristics (Evaluation function)

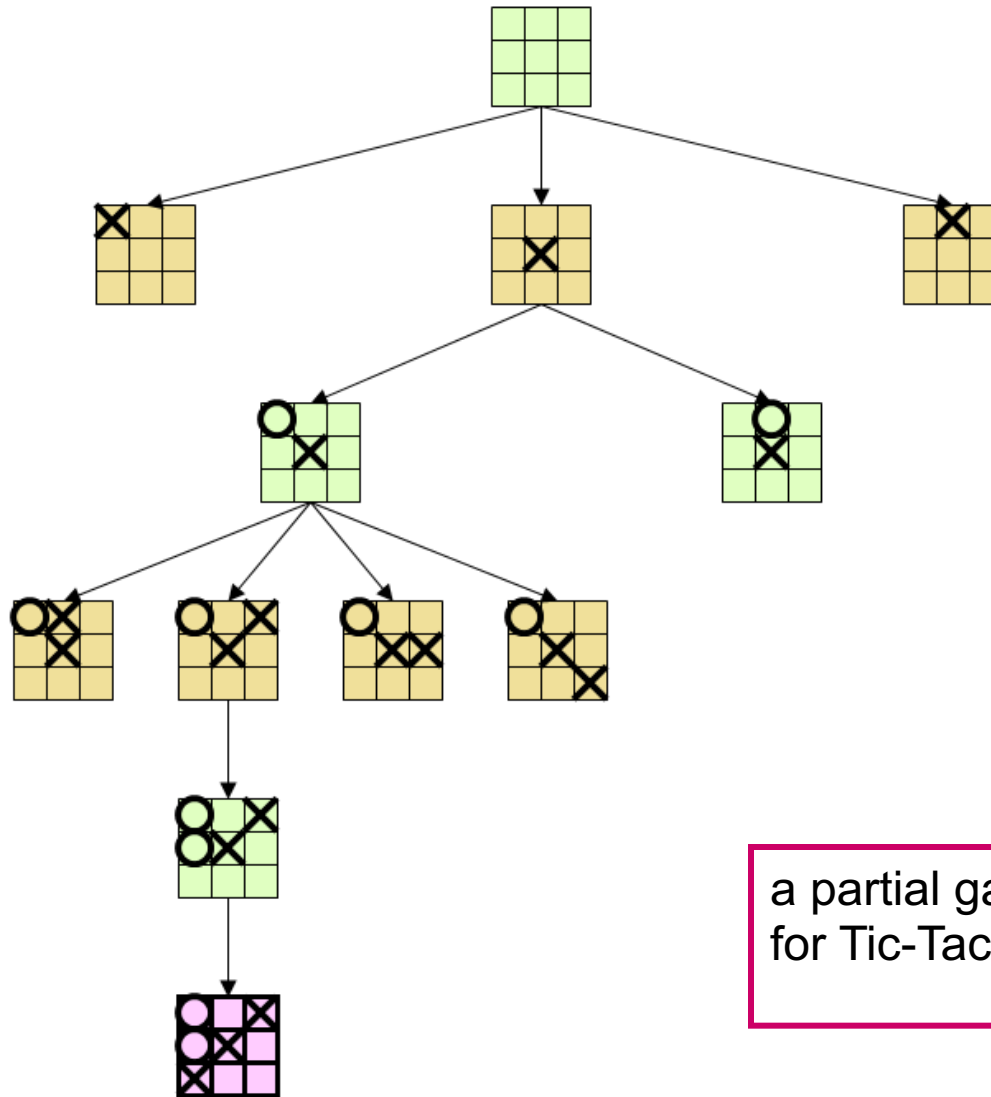
- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1f_1(s) + w_2f_2(s) + \dots + w_df_d(s)$
  - *weights* · *features* [dot product]
- For example, in chess
  - $W = \{ 1, 3, 3, 5, 8 \}$
  - $F = \{ \# \text{ pawns advantage, } \# \text{ bishops advantage, } \# \text{ knights advantage, } \# \text{ rooks advantage, } \# \text{ queens advantage} \}$
  - Is this what Deep Blue used?
  - What are some problems with this?



# Heuristics (Evaluation function)

- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1f_1(s) + w_2f_2(s) + \dots + w_df_d(s)$
  - *weights* · *features* [dot product]
- For example, in chess
  - $W = \{ 1, 3, 3, 5, 8 \}$
  - $F = \{ \# \text{ pawns advantage, } \# \text{ bishops advantage, } \# \text{ knights advantage, } \# \text{ rooks advantage, } \# \text{ queens advantage} \}$
  - Is this what Deep Blue used?
  - What are some problems with this?
- More complex evaluation functions may involve learning
  - Adjusting weights based on outcomes
  - Perhaps non-linear functions
  - How to choose the *features*?

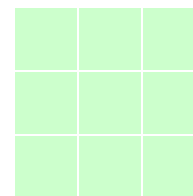
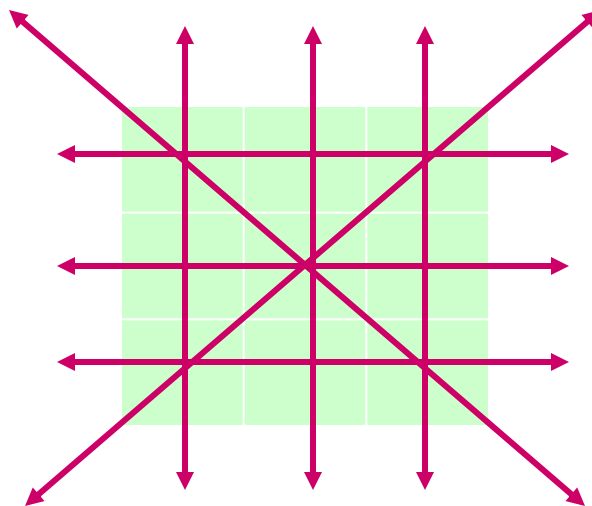
# Tic-Tac-Toe revisited



a partial game tree  
for Tic-Tac-Toe

# Evaluation function for Tic-Tac-Toe

- A simple evaluation function for Tic-Tac-Toe
  - count the number of rows where **X** can win
  - subtract the 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**



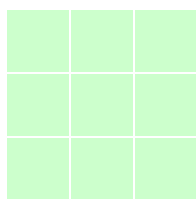
$$8-8 = 0$$

↑ ↑

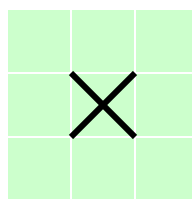
# Evaluating Tic-Tac-Toe

$\text{eval}_X = (\text{number of rows where } X \text{ can win}) - (\text{number of rows where } O \text{ can win})$

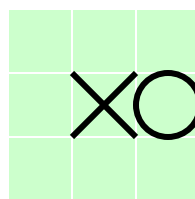
- After **X** moves in center, score for **X** is +4
- After **O** moves, score for **X** is +2
- After **X**'s next move, score for **X** is +4



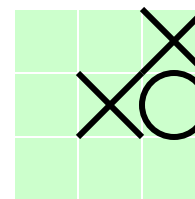
$$8-8 = 0$$



$$8-4 = 4$$



$$6-4 = 2$$

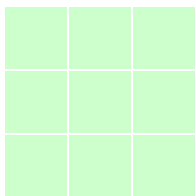


$$6-2 = 4$$

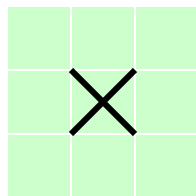
# Evaluating Tic-Tac-Toe

$\text{evalO} = (\text{number of rows where } \textcircled{O} \text{ can win}) - (\text{number of rows where } \text{X} \text{ can win})$

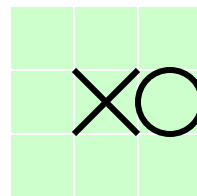
- After **X** moves in center, score for **O** is -4
- After **O** moves, score for **O** is +2
- After **X**'s next move, score for **O** is -4



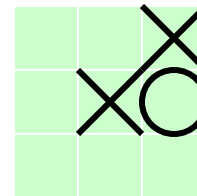
$$8-8 = 0$$



$$4-8 = -4$$



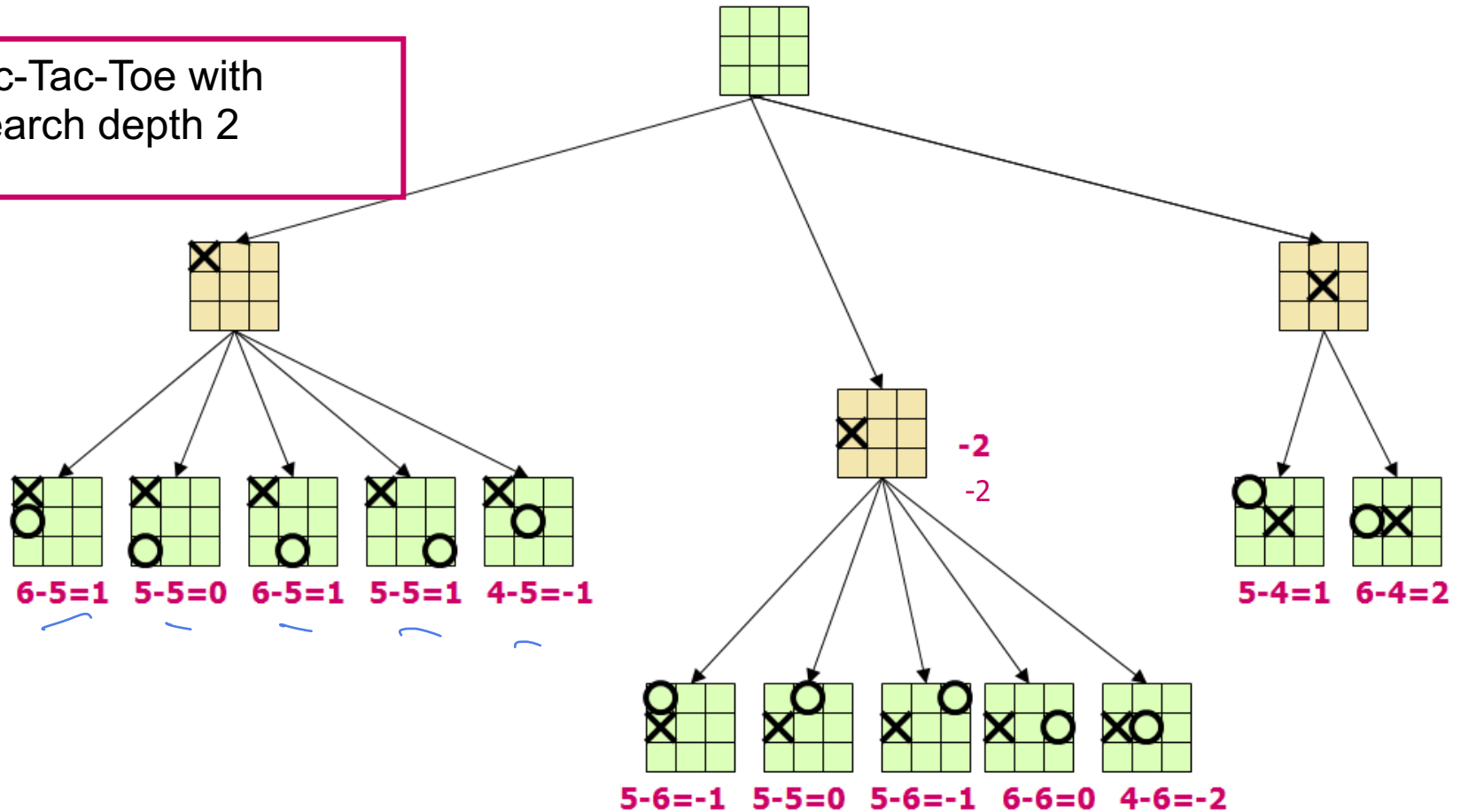
$$4-6 = -2$$



$$2-6 = -4$$

# Search depth cutoff

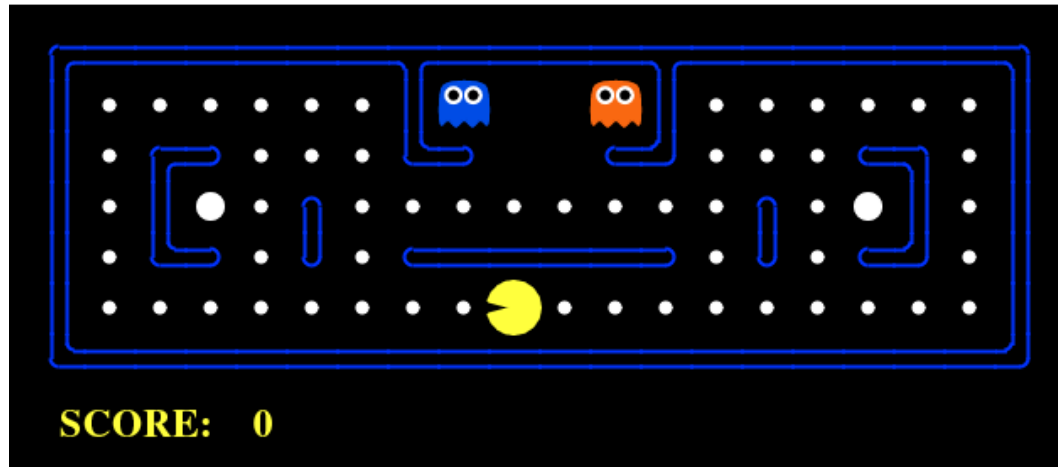
Tic-Tac-Toe with search depth 2



Evaluations shown for X

# Coding Project 2

- PACMAN with ghosts



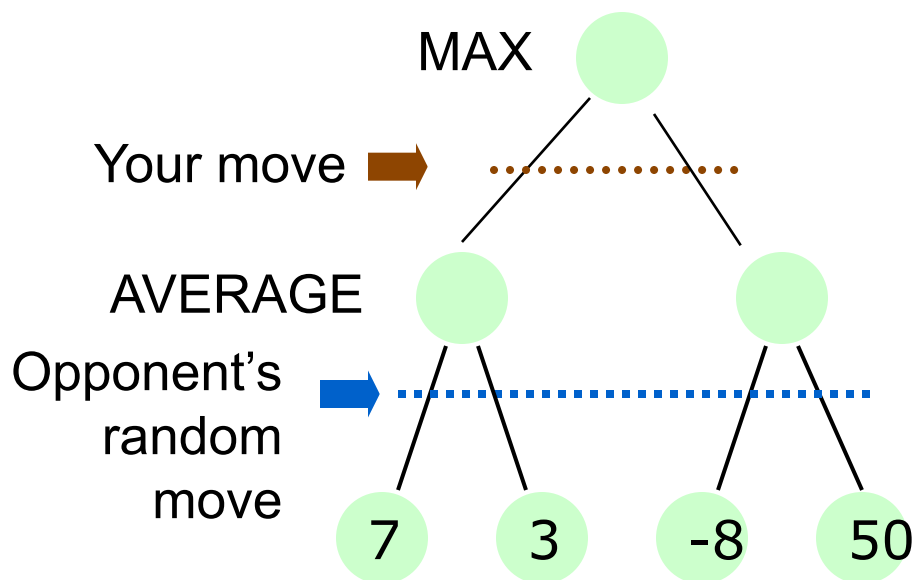
- You will generalize minimax search to multiple opponents in Q7
- You will exploit the benign opponents in Q8

# Expectimax: Playing against a benign opponent

- Sometimes your opponents are not clever.
  - They behave randomly.
  - You can take advantage of that by modeling your opponent.
- Example of game of chance:
  - Slot machines
  - Tetris

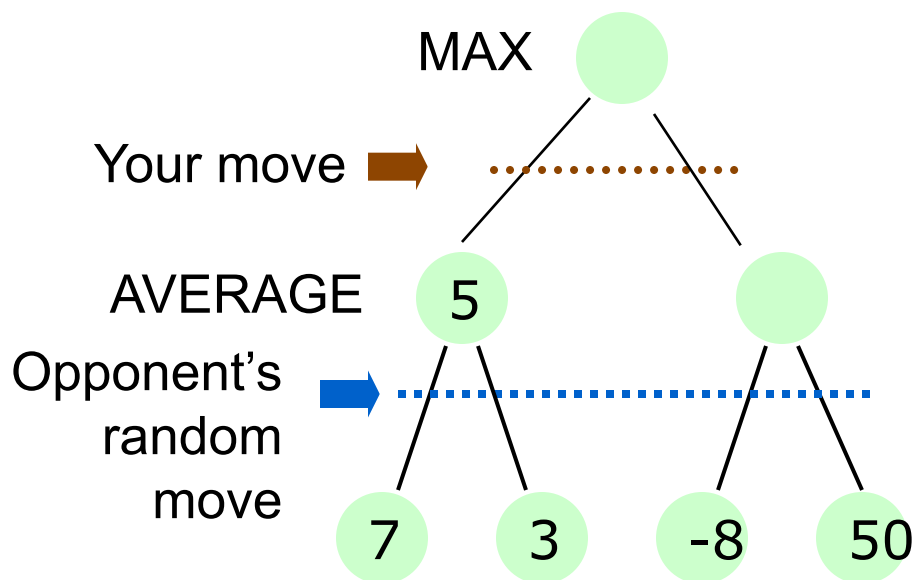


# Expectimax example



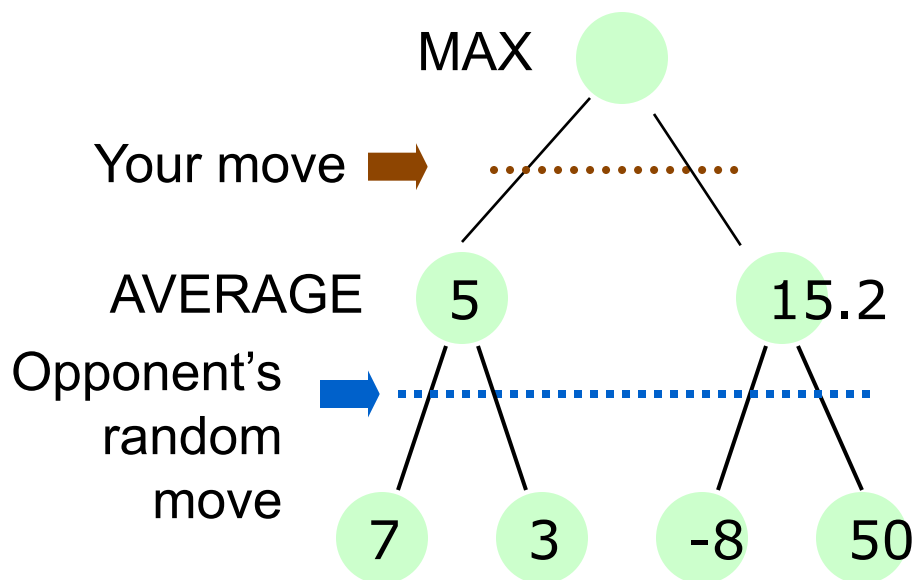
- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

# Expectimax example



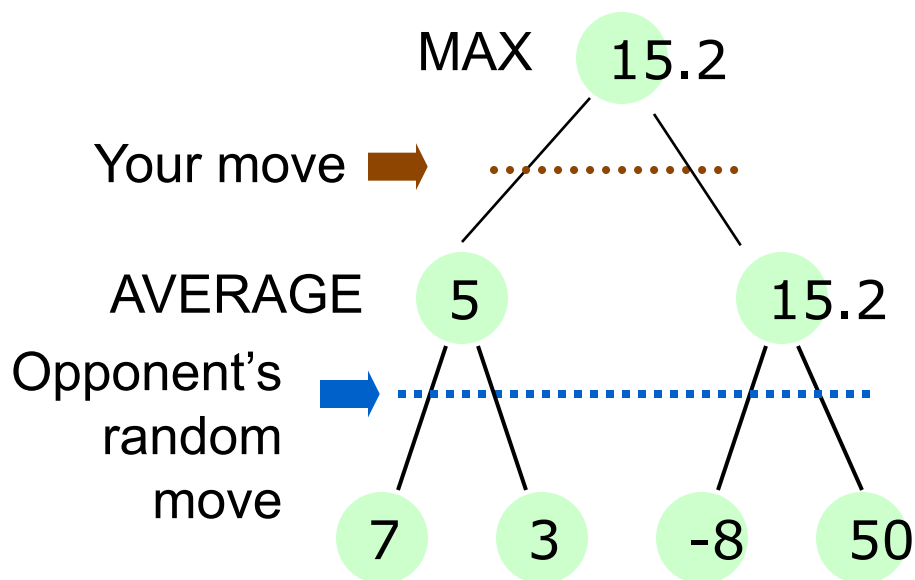
- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

# Expectimax example



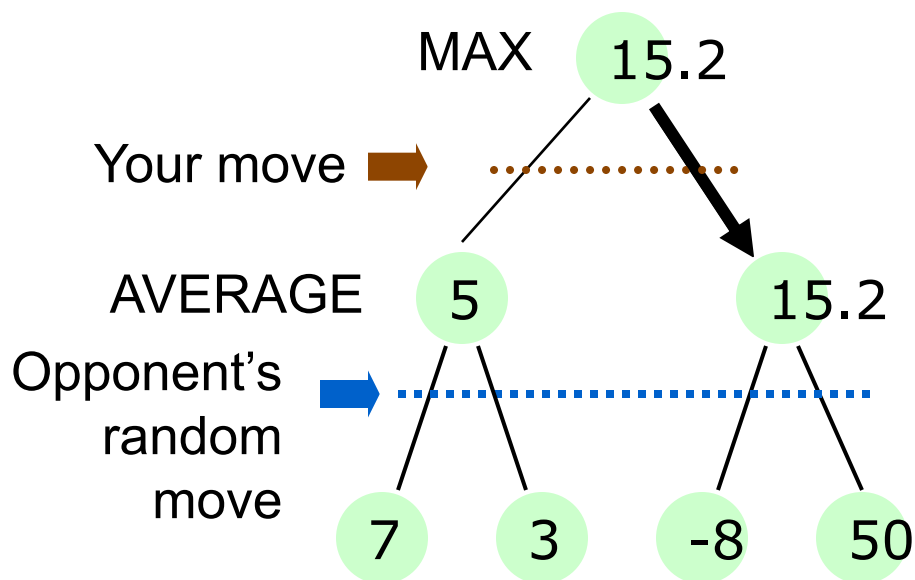
- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

# Expectimax example



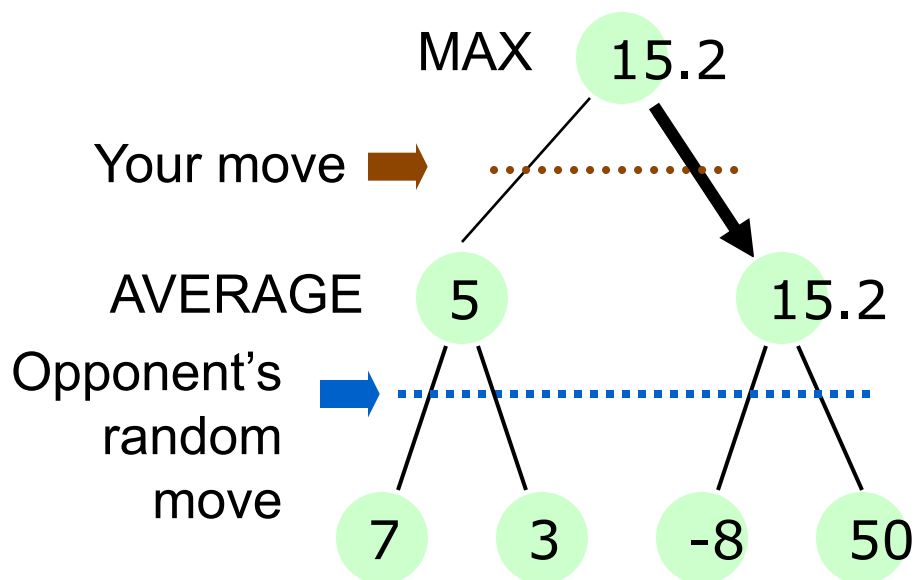
- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

# Expectimax example



- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

# Expectimax example



- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

**Note: pruning becomes tricky in expectimax... think about why.**

# Summary of game playing

- Minimax search
- Game tree
- Alpha-beta pruning
- Early stop with an evaluation function
- Expectimax

# More reading / resources about game playing

- Required reading: AIMA 5.1-5.3
- Stochastic game / Expectiminimax: AIMA 5.5
  - Backgammon. TD-Gammon
  - Blackjack, Poker
- Famous game AI: Read AIMA Ch. 5 “Historical notes”
  - Deep blue
  - TD Gammon
  - AlphaGo / AlphaZero
- AlphaGo: <https://www.nature.com/articles/nature16961>



# About the upcoming midterm

- **Topics:** Cover up to A\*-Search and HW2. (not including Games / Adversarial search)
- **Types of questions:**
  - True/False, MCQ, BayesNet reading, simple calculations / derivations. Similar to quiz questions.
- **What am I testing you on:**
  - Test of basic concepts.
  - Test of simple mechanical calculations.
  - Test of your ability to “generalize”

# About the upcoming midterm

- **Topics:** Cover up to A\*-Search and HW2. (not including Games / Adversarial search)
- **Types of questions:**
  - True/False, MCQ, BayesNet reading, simple calculations / derivations. Similar to quiz questions.
- **What am I testing you on:**
  - Test of basic concepts.
  - Test of simple mechanical calculations.
  - Test of your ability to “generalize”
- **Tough questions? Don't panic and try your best.**
  - Bonus questions may not be difficult (some are), but you are expected to finish non-bonus questions first.

# About the upcoming midterm

- **Date:** Next Tuesday in class
- **Duration:** 1 hour 15 minutes
- **Format:** Open book (but no digital devices)
  - No calculator needed.

# How to prepare for the midterm?

- Read relevant chapters from AIMA
- Lecture videos from a previous year will be made available
  - Slight differences on the syllabus
- Make sure your concepts on how the algorithms work are clear
  - Solve problems on Ed Quiz
  - Solve problems from HW0, HW1, HW2
- Participate in discussions on Ed

# Time to pause and take a look back!

Week	Topic	
1	Course Overview & Intelligent Agents	
2	Machine Learning	}
	Machine Learning	
2	Machine Learning	}
	Probabilistic Graphical Models	
3	Probabilistic Graphical Models	}
	Search: Problem solving with search	
4	Search: Search algorithms	}
	Search: Minimax search and game playing	
5	Midterm Review	
	<b>Midterm</b>	
6	RL: Intro / Markov Decision Processes	}
	RL: Solving MDPs	
7	RL: Bandits and Exploration	}
	RL: Reinforcement Learning Algorithms	
8	RL: Reinforcement Learning Algorithms	}
	Logic: Propositional Logic	
9	Logic: First order Logic	}
	Responsible AI	
10	Responsible AI	}
	Final Review	
11	Final Exam	

Machine Learning

Probabilistic Reasoning

Search

Reinforcement Learning

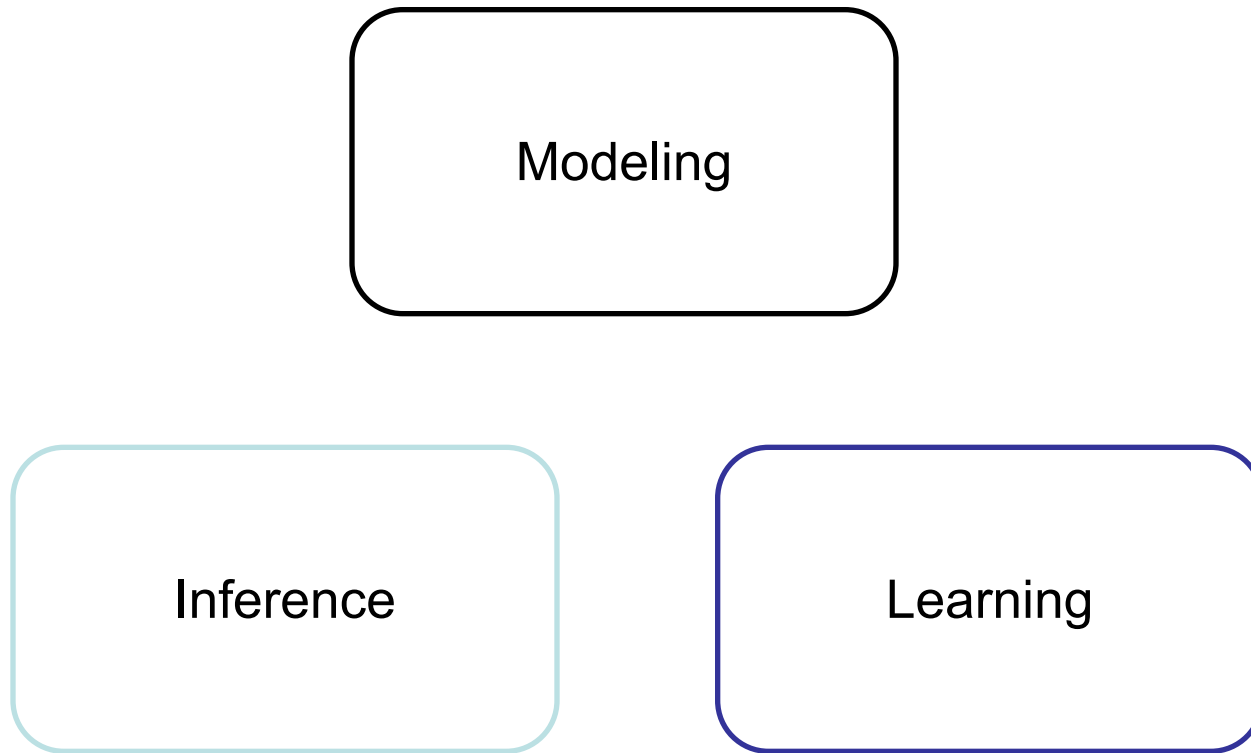
Logic

Responsible AI

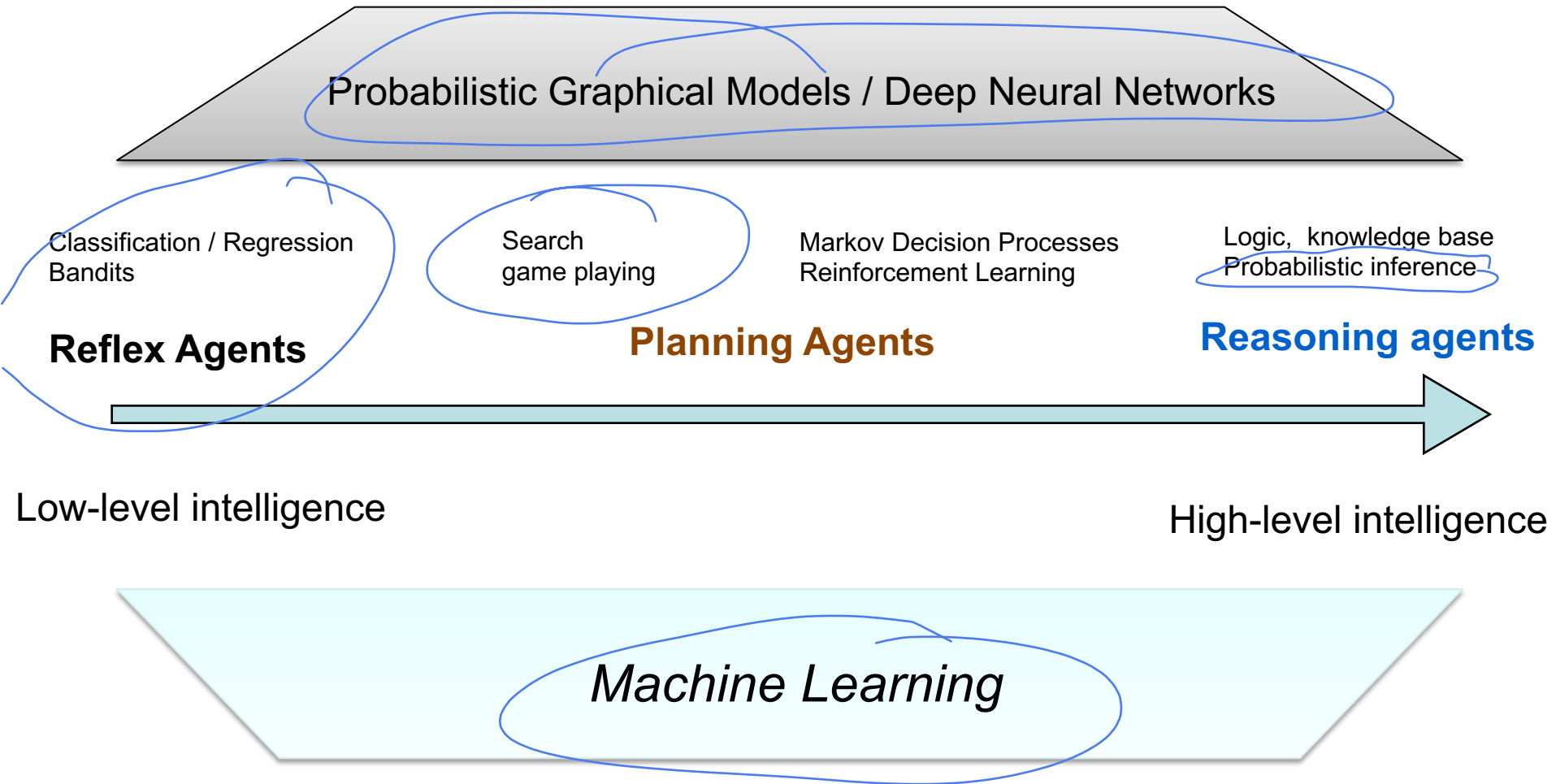
# Lecture 1: AI Overview & Agents

- AI for problem solving
- Rational agents
- Examples of AI in the real world
- Modelling-Inference-Learning Paradigm

# New paradigm: Modeling-Inference-Learning



# Structure of the course





# Modeling-Learning-Inference Paradigm

---

	<b>Modeling</b>	<b>Learning</b>	<b>Inference</b>
Classifier agent (Spam filter)	Feature engineering Hypothesis class	Minimize Error rate	Prediction on new data points
Probabilistic Inference agent (Sherlock)	Joint distribution Draw edges in BN Conditional independences	Fitting the CPTs to Data	Marginalization (conceptually easy)
Search agents	State-Space- diagram	Environment given (learn edge weights) <b>(Learn heuristics?)</b>	<b>Uninformed Search / A* Search etc...</b>

# PEAS Description of Task Environment

	<b>P</b>	<b>E</b>	<b>A</b>	<b>S</b>
Classifier agent (Spam filter)	Error rate	New example arrives	Classify	Observe the feature vector
Probabilistic Inference agent (Sherlock)	Error in the computed $P(Y X)$ Time/space complexity	Joint distribution via PGM	Probabilistic (Bayesian) Inference	Observe evidence
Search agents	Completeness, Optimality, Time / Space complexity	State-space diagram	Choose a sequence of operators	Observe next node

# Lecture 2-4: Machine Learning

- Machine learning
  - Types of machine learning models
  - Focus on Supervised Learning ---- classifier agents.
- What is a feature vector?
  - Feature engineering, feature extraction
- Hypothesis class and free parameters
  - How many are there? How to evaluate a classifier?
  - Error? On training data or on new data?
  - Overfitting, underfitting?
- How to learn (optimize)?
  - Surrogate losses, Gradient Descent, SGD

# Key concepts and learning goals

- Linear classifier
  - What is the score function? How does a linear classifier classify?
  - What are the (learnable) “free parameters of a linear classifier”?
- What is a decision boundary? How to analytically describe the decision boundary of a linear classifier?
- How to evaluate a classifier?
  - Training error, Validation error, Test error
  - Overfitting, Underfitting, Generalization
  - What is the difference between error and (surrogate) losses?

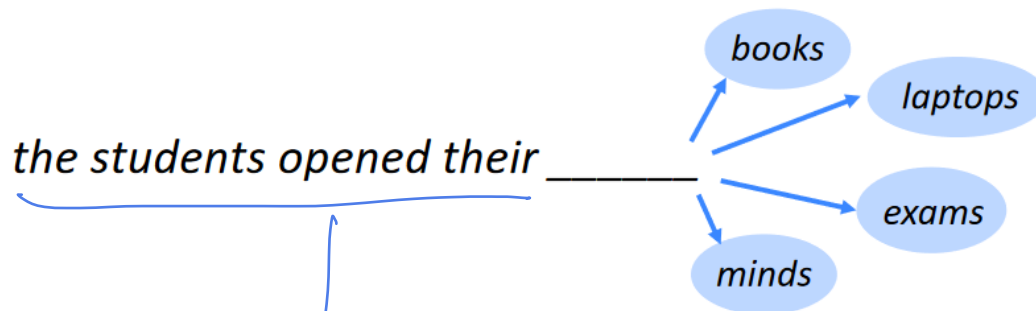


# Training a classifier

- How does Gradient Descent work?
- How does Stochastic Gradient Decent work?
- What is a “learning rate” or “step size” parameter for GD and SGD?
- How to calculate the gradient of a loss function?
  - We worked out “logistic loss”
  - We worked out “cross entropy” loss in Project 1

# Example: Language model / ChatGPT

- Formulate “Next Word prediction” as a classification problem.



- Feature space?  $\phi(\text{prompt}) \in \mathbb{R}^d$
- Label space?  $\text{Vocabulary}_t$
- Hypothesis class?  $f_{\theta}(\phi(\text{prompt})) \rightarrow \text{Vocab}$
- Loss function?  $\sum_{a \in \mathcal{Y}} -\log \hat{P}_{\theta}(x) \cdot \mathbb{1}(y=a)$
- Method for training?  $\longrightarrow$
- How to evaluate?

# Lecture 5-7: Probabilities and BayesNet

- Modelling the world with a joint probability distribution
  - (You can focus on discrete random variables!)
- Key concepts:
  - Probability distribution, p.m.f., Random variables, events
  - Joint / conditional / marginal probabilities
  - Independence and conditional independence
  - Bayesian Networks and how they allow more efficient representation of a joint probability distribution

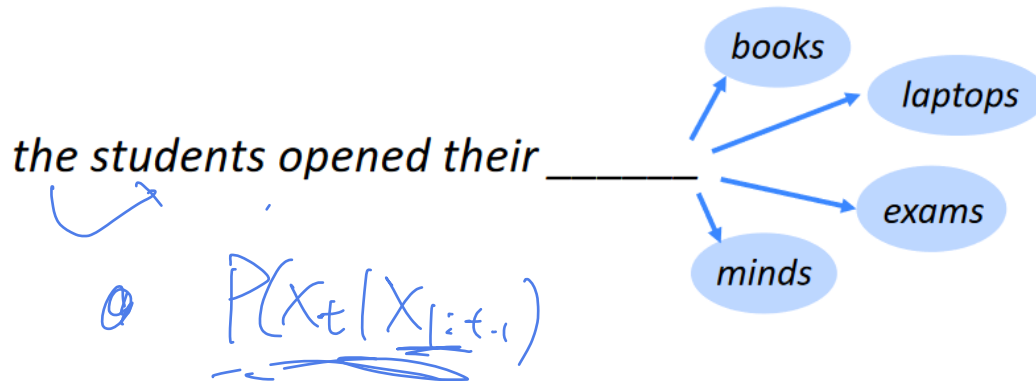
# Key learning goals

- Factorization of the joint distribution into CPTs according to a graph
  - Count number of independent numbers to unique specify a CPT
- Express  $P(\text{Query} \mid \text{Evidence})$  using only CPTs.
- Read off conditional independences from the graph
  - d-separation
  - Bayes ball algorithm
  - Markov Blanket



# Example: Language models / ChatGPT

- Maybe there isn't a clear cut label, we should quantify uncertainties



- Model text as a joint distribution. What are the random variables?

$$P(X_1, X_2, X_3, \dots, X_T)$$

- How do they depend on each other?

- How efficient is the representation?

Varab<sup>T</sup>

- What do we do to make inference once we have the joint distribution?

# Lecture 7-9: Search

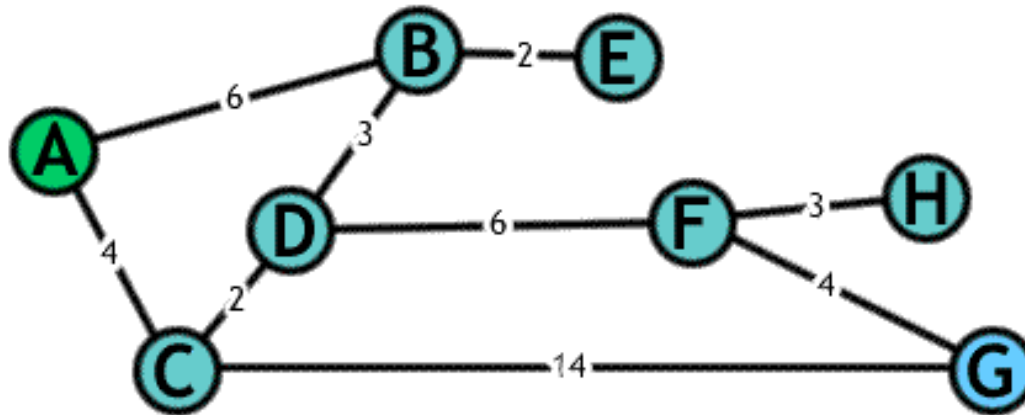
- Problem solving by search
  - Abstraction, problem formulation
  - State-space diagram
  - Count the number of states, number of actions.
- Search algorithms
  - General search strategy, data-structure used
  - Space / time complexity
  - Four evaluation criteria
- Heuristic search
  - When does it work?

# Problem Formulation and Search

- Problem formulation
  - State-space description  $\langle \{S\}, S_0, \{S_G\}, \{O\}, \{g\} \rangle$ 
    - **S**: Possible states
    - **S<sub>0</sub>**: Initial state of the agent
    - **S<sub>G</sub>**: Goal state(s)
      - Or equivalently, a goal test **G(S)**
    - **O**: Operators  $O: \{S\} \Rightarrow \{S\}$ 
      - Describes the possible actions of the agent
    - **g**: Path cost function, assigns a cost to a path/action
- At any given time, which possible action **O<sub>i</sub>** is best?
  - Depends on the goal, the path cost function, the future sequence of actions....
- Agent's strategy: Formulate, Search, and Execute
  - This is *offline* problem solving

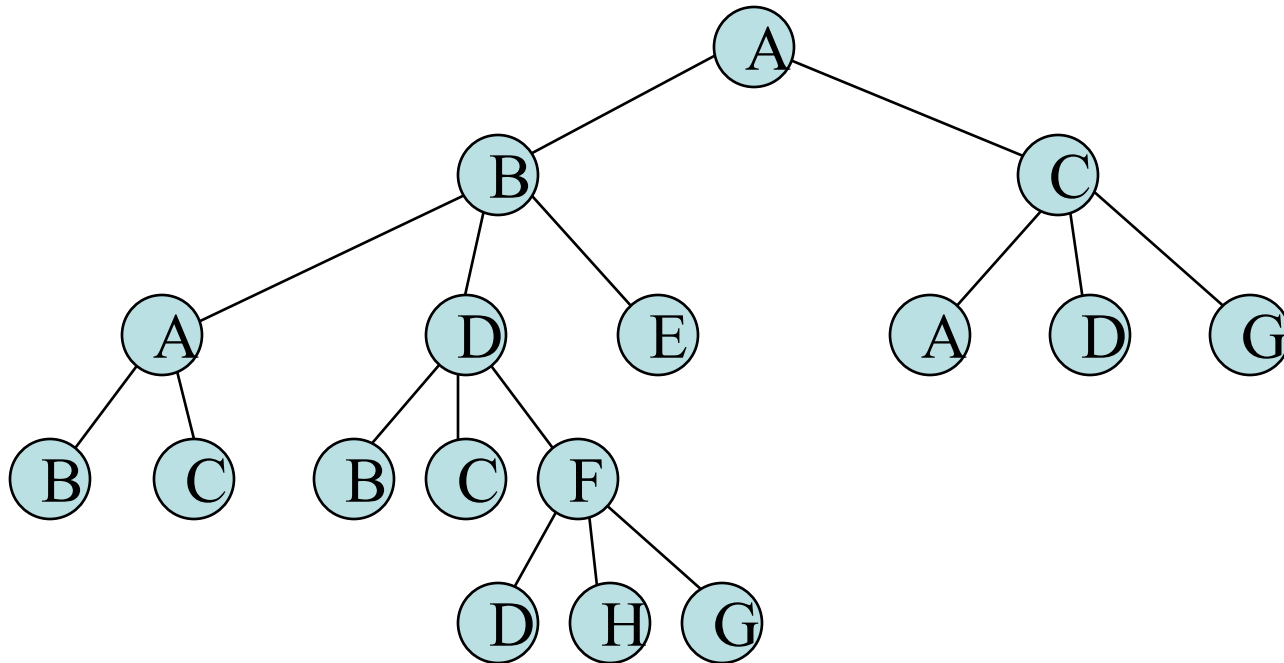
# State-Space Diagrams

- State-space description can be represented by a state-space diagram, which shows
  - States (incl. initial and goal)
  - Operators/actions (state transitions)
  - Path costs



# State Space vs. Search Tree (cont.)

Search tree (partially expanded)



# General Search Algorithm

- Uses a queue (a list) and a **queuing function** to implement a *search strategy*
  - **Queuing-Fn**(*queue*, *elements*) inserts a set of elements into the queue and determines the order of node expansion

**function** **GENERAL-SEARCH**(*problem*, **QUEUING-FN**) **returns** a solution or failure

*nodes* ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[*problem*]))

**loop do**

**if** *nodes* is empty **then return** failure

*node* ← REMOVE-FRONT(*nodes*)

**if** GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **then return** *node*

*nodes* ← **QUEUING-FN**(*nodes*, EXPAND(*node*, OPERATORS[*problem*]))

**end**

# Summary table of uninformed search

Criteria	BFS	Uniform-cost	DFS	Depth-limited	IDS	Bidirectional
Complete?	Yes <sup>#</sup>	Yes <sup>#&amp;</sup>	No	No	Yes <sup>#</sup>	Yes <sup>#+</sup>
Time	$O(b^d)$	$O(b^{1+[C^*/e]})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+[C^*/e]})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes <sup>\$</sup>	Yes	No	No	Yes <sup>\$</sup>	Yes <sup>\$+</sup>

$b$ : Branching factor

$d$ : Depth of the shallowest goal

$l$ : Depth limit

$m$ : Maximum depth of search tree

$e$ : The lower bound of the step cost

<sup>#</sup>: Complete if  $b$  is finite

<sup>&</sup>: Complete if step cost  $\geq e$

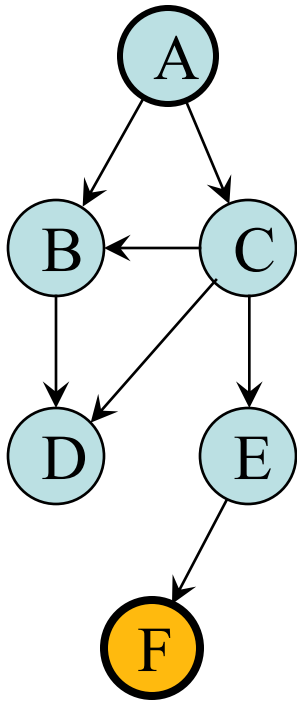
<sup>\$</sup>: Optimal if all step costs are identical

<sup>+</sup>: If both direction use BFS

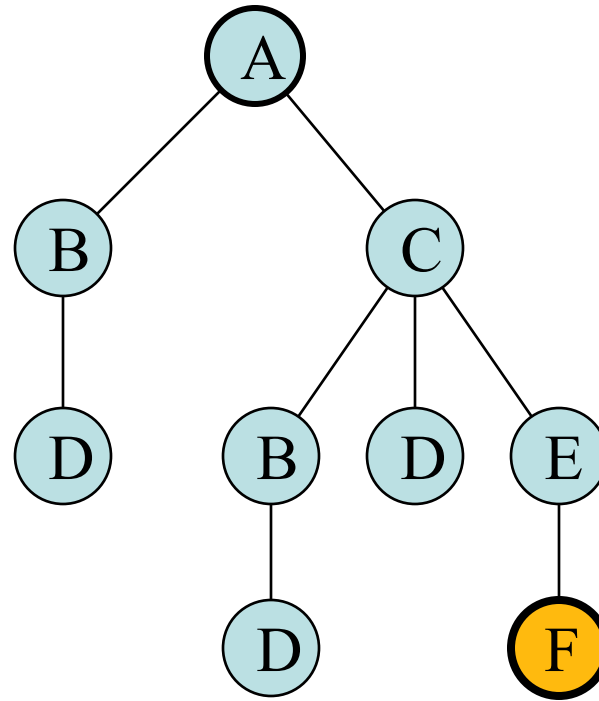
(Section 3.4.6 in the AIMA book.)

# Example

State space graph



Search tree



Queue

- (A)
- (B C)
- (C D)
- (D B D E)
- (B D E)
- (D E D)
- (E D)
- (D F)
- (F)
- ( )



# Key concepts and learning goals

- Formulate a problem as a search problem
  - State, Operators, State-space diagram
- Search algorithms:
  - Know the specific “Queue” data structure
  - Know how to work out the sequence of nodes to expand / keep track of the “Frontier”
  - Time / space complexity
  - Tree search vs Graph search
- Heuristics
  - What is an “admissible” heuristic? What is a “consistent” heuristic? What do they ensure?

# Example: Language Models / ChatGPT

- We don't just generate one word, we want to generate a long paragraph all together.

Model: Default (GPT-3.5)



Students in 165A is working hard to prepare their midterm coming next Tuesday. They \_\_\_



may be studying diligently, reviewing their notes and textbook, practicing problems, and seeking help from their instructor or classmates if they encounter difficulties. They may also be managing their time effectively to ensure they have enough time to prepare adequately for the midterm. Additionally, they may be engaging in self-care activities, such as getting enough sleep, eating well, and taking breaks to avoid burnout and stay focused. Overall, the students in 165A are likely doing their best to prepare for the upcoming midterm.



# Example: Language Models / ChatGPT

- Given an already trained language model, we have
  - i.e., we have access to  $P(\text{Next Word} \mid \text{Sentence up to this point})$
  - How to get “Generated Text”  $X_{1:T}$  when we have  $X_{\{-N:0\}}$ ?  
*Prompt*
- Naïve approach: Greedy next word prediction
  - $X_t = \underset{x}{\operatorname{argmax}} P(X_t = x \mid \text{Prompt}, X_{1:t-1})$
  - Or  $X_t = \text{classifier.predict}(\text{feature}(\text{Prompt}, X_{1:t-1}))$
- Might not be the best quality when we evaluate the generated text as a whole. Maybe we want

$$X_{1:T} = \underset{x}{\operatorname{argmax}} P(X_{1:T} = x_{1:T} \mid \text{Prompt},)$$

# Example: Language Models / ChatGPT

- Surprise. Surprise. This is a search problem.
- How to formulate it as a search problem?
  - Assume we have access to  $P(X_t | X_{1:t})$
  - The goal is to maximize  $P(\text{Generated Text} | \text{Prompt})$  over “Generated Text”.

$$P(X_{1:T} | X_{-N:0}) = P(X_1 | X_{-N:0}) P(X_2 | X_{-N:1}) P(X_3 | X_{-N:2}) \dots P(X_T | X_{-N:T-1})$$

# Example: Language Models / ChatGPT

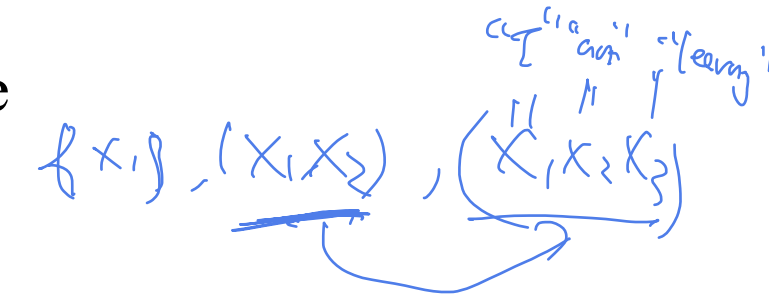
- Surprise. Surprise. This is a search problem.
- How to formulate it as a search problem?
  - Assume we have access to  $P(X_t | X_{1:t})$
  - The goal is to maximize  $P(\text{Generated Text} | \text{Prompt})$  over “Generated Text”.

“Decoding”

$$P(X_{1:T} | X_{-N:0}) = P(X_1 | X_{-N:0}) P(X_2 | X_{-N:1}) P(X_3 | X_{-N:2}) \dots P(X_T | X_{-N:T-1})$$

- What are the

- States
- Operators
- Goal test
- Path cost

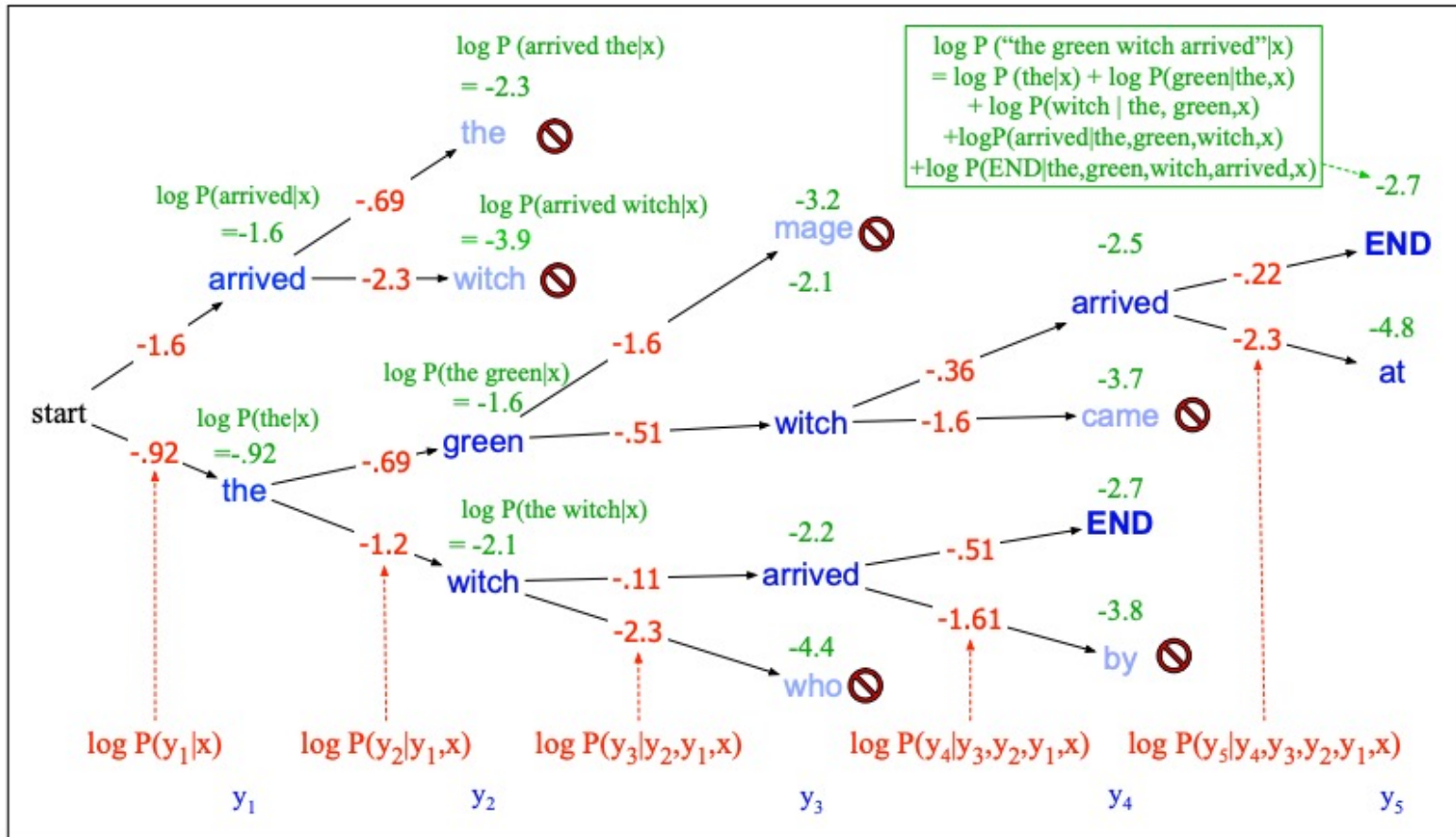


“Beam Search”

⟨END⟩

$$\log P(X_{1:t} | X_{-N:0}) = \log P(x_1 | X_{-N:0}) + \log P(x_2 | X_{-N:1}) + \dots + \log P(x_t | X_{-N:t-1})$$

# A search tree for maximum likelihood text generation



**Figure 10.10** Scoring for beam search decoding with a beam width of  $k = 2$ . We maintain the log probability of each hypothesis in the beam by incrementally adding the logprob of generating each next token. Only the top  $k$  paths are extended to the next step.

# What are the branching factors and depth for the task? How large are the state space?

- Branching factor?
- Search depth?
- Time complexity
- Space complexity
- Which search algorithm should you use?

# What are some general principles behind AI?

Week	Topic	
1	Course Overview & Intelligent Agents	
2	Machine Learning	}
	Machine Learning	
2	Machine Learning	}
	Probabilistic Graphical Models	
3	Probabilistic Graphical Models	}
	Search: Problem solving with search	
4	Search: Search algorithms	}
	Search: Minimax search and game playing	
5	Midterm Review	
	<b>Midterm</b>	
6	RL: Intro / Markov Decision Processes	}
	RL: Solving MDPs	
7	RL: Bandits and Exploration	}
	RL: Reinforcement Learning Algorithms	
8	RL: Reinforcement Learning Algorithms	}
	Logic: Propositional Logic	
9	Logic: First order Logic	}
	Responsible AI	
10	Responsible AI	}
	Final Review	
11	Final Exam	

Machine Learning

Probabilistic Reasoning

Search

Reinforcement Learning

Logic

Responsible AI



# Optimization perspective of AI

# Optimization perspective of AI

- A rational agent  $\max_{a_1, \dots, a_T} \text{Utility}(a_1, \dots, a_T)$ 
  - **Modelling tools:** Features / Hypothesis class, PGM, State-space abstraction, agent categorization
  - **Constraints:** Computation, Data, Storage

# Optimization perspective of AI

- A rational agent  $\max_{a_1, \dots, a_T} \text{Utility}(a_1, \dots, a_T)$ 
  - **Modelling tools:** Features / Hypothesis class, PGM, State-space abstraction, agent categorization
  - **Constraints:** Computation, Data, Storage
- Discrete optimization  $\min_{p \in \text{Paths}} \text{Distance}(p)$ 
  - **Algorithmic tool:** Search / Dynamic programming

# Optimization perspective of AI

- A rational agent  $\max_{a_1, \dots, a_T} \text{Utility}(a_1, \dots, a_T)$ 
  - **Modelling tools:** Features / Hypothesis class, PGM, State-space abstraction, agent categorization
  - **Constraints:** Computation, Data, Storage
- Discrete optimization  $\min_{p \in \text{Paths}} \text{Distance}(p)$ 
  - **Algorithmic tool:** Search / Dynamic programming
- Continuous optimization  $\min_{\theta \in \mathbb{R}^d} \text{TrainingError}(\theta)$ 
  - **Algorithmic tool:** Gradient descent / Stochastic gradient descent

# Different objectives to optimize in AI

- PGM:
  - MLE: Maximize the log-likelihood function
  - Classifier / decision: max the posterior distribution
- Search and planning:
  - Find valid solutions with smallest path cost.
- Machine Learning
  - Empirical Risk Minimization (ERM):

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \ell(\theta, (x_i, y_i))$$

A lot of these problems are computationally / statistically hard, but so what?

# A lot of these problems are computationally / statistically hard, but so what?

- Get help from human:
  - Use model
  - Use abstractions at the right level
  - Use features
  - Use heuristic functions

# A lot of these problems are computationally / statistically hard, but so what?

- Get help from human:
  - Use model
  - Use abstractions at the right level
  - Use features
  - Use heuristic functions
- Get help from mathematics and computer science theory:
  - Solve an approx. solution
  - Approximate inference of a PGM
  - More iterations with less accuracy per SGD
  - A\* Search



Good luck with your midterm!