# Lecture 27
# Exam 3 Review

Mark Hasegawa-Johnson

May 3, 2021

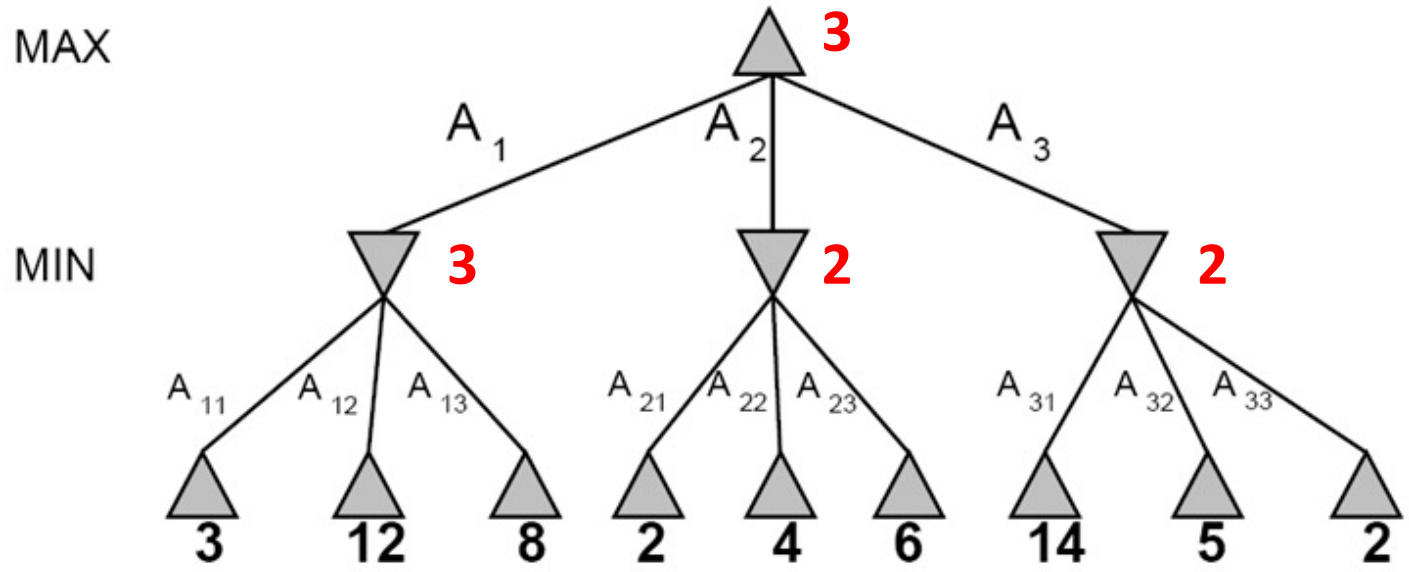
# Outline: Exam 3 will be multiple-choice, on prairielearn, including 24 problems

- 4 problems: Midterm 1 material
- 4 problems: Midterm 2 material
- 4-5 problems: minimax, alpha-beta, expectiminimax
- 3-4 problems: game theory
- 5-7 problems: MDP, reinforcement learning
- 1 problem: AI fairness (definitions of fairness; causal graphs)
- 1 problem: computer vision (similar triangles; convolution)

# Outline: Exam 3 will be multiple-choice, on prairielearn, including 24 problems

- 4 problems: Midterm 1 material
- 4 problems: Midterm 2 material
- 4-5 problems: minimax, alpha-beta, expectiminimax
- 3-4 problems: game theory
- 5-7 problems: MDP, reinforcement learning
- 1 problem: AI fairness (definitions of fairness; causal graphs)
- 1 problem: computer vision (similar triangles; convolution)

# Computing the minimax value of a node



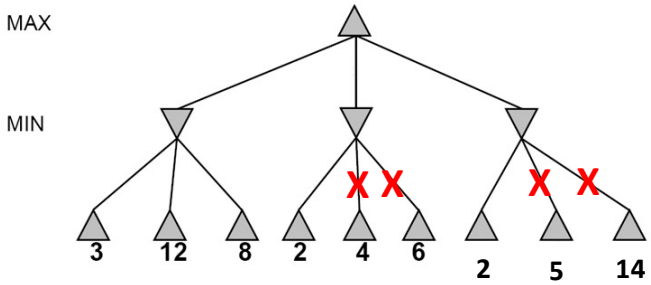

- **Minimax**(*node*) =
  - Utility(*node*) if *node* is terminal
  - $\max_{action}$ **Minimax**(Succ(*node, action*)) if *player* = MAX
  - $\min_{action}$ **Minimax**(Succ(*node, action*)) if *player* = MIN

# The pruning thresholds, alpha and beta

Alpha-beta pruning requires us to keep track of two pruning thresholds, alpha and beta.

- alpha ($\alpha$) is the highest score that MAX knows how to force MIN to accept.
- beta ($\beta$) is the lowest score that MIN knows how to force MAX to accept.
- $\alpha < \beta$

# Computational Complexity




Consider a sequence of two levels, with n moves per level, and with optimal ordering.

- There are $n^2$ terminal nodes.
- Alpha-beta will evaluate all the daughters of the first daughter: $n$ nodes.
- Alpha-beta will also evaluate the first daughter of each non-first daughter: $n-1$ nodes.
- In total, alpha-beta will evaluate $2n-1$ out of every $n^2$ nodes.
- For a tree of depth d, the number of nodes evaluated by alpha-beta is

$$(2n-1)^{d/2} = O\{n^{d/2}\}$$

# Expectiminimax: notation

= MAX node. $U(s) = \max_{a \in A(s)} Q(s,a)$

= MIN node. $U(s) = \min_{a \in A(s)} Q(s,a)$

= Chance node. $Q(s,a) = \sum_{s'} P(s'|s,a)U(s')$

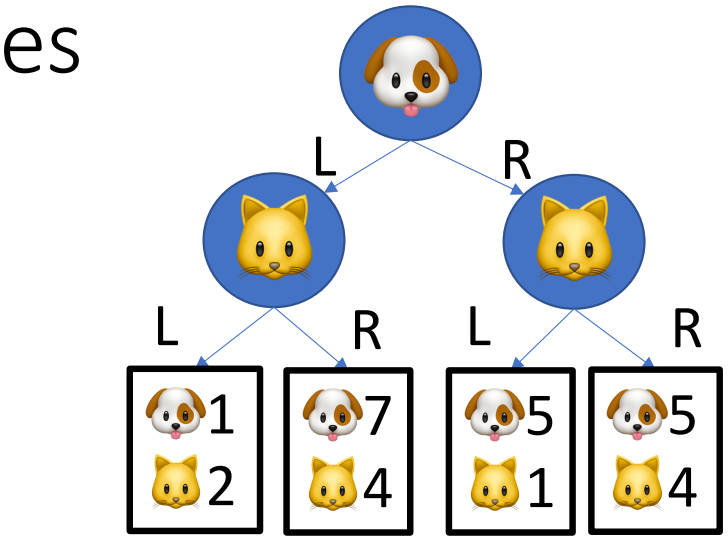
# Outline: Exam 3 will be multiple-choice, on prairielearn, including 24 problems

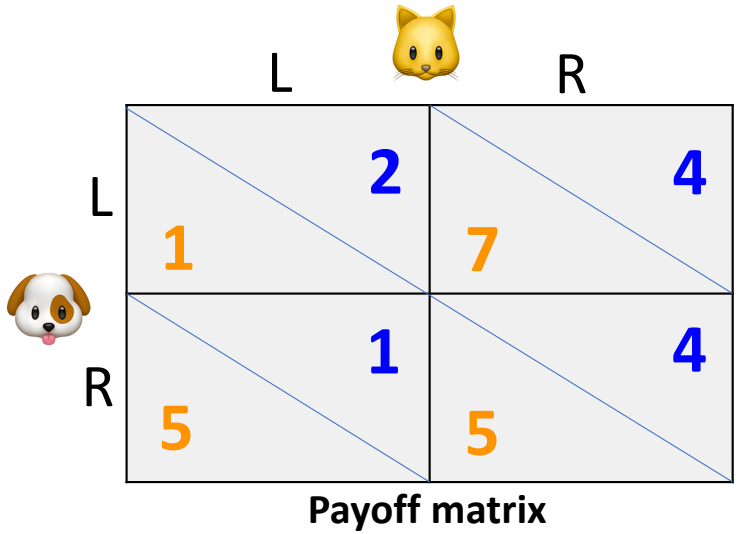
- 4 problems: Midterm 1 material
- 4 problems: Midterm 2 material
- 4-5 problems: minimax, alpha-beta, expectiminimax
- 3-4 problems: game theory
- 5-7 problems: MDP, reinforcement learning
- 1 problem: AI fairness (definitions of fairness; causal graphs)
- 1 problem: computer vision (similar triangles; convolution)

# Notation: simultaneous games

The payoff matrix shows:

- Each column is a different move for player 1.
- Each row is a different move for player 2.
- Each square is labeled with the rewards earned by each player in that square.




**Payoff matrix**

# Nash Equilibrium

Photo by Scott Bauer, Public Domain, https://commons.wikimedia.org/w/index.php?curid=245466

| Bob \ Alice | Defect | Cooperate |
|---|---|---|
| Defect | 10 / 10 | 0 / 10 |
| Cooperate | 10 / 0 | 100 / 100 |

By Ancheta Wis, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=68432449

A Nash Equilibrium is a game outcome such that each player, knowing the other player's move in advance, responds rationally.

# Pareto optimal outcome

Photo by Scott Bauer, Public Domain, https://commons.wikimedia.org/w/index.php?curid=245466

| Bob \ Alice | Stag | Alligator |
|---|---|---|
| Stag | 20, 10 | 0, 0 |
| Alligator | 0, 0 | 10, 20 |

By Ancheta Wis, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=68432449

An outcome is Pareto-optimal if the only way to increase value for one player is by decreasing value for the other.

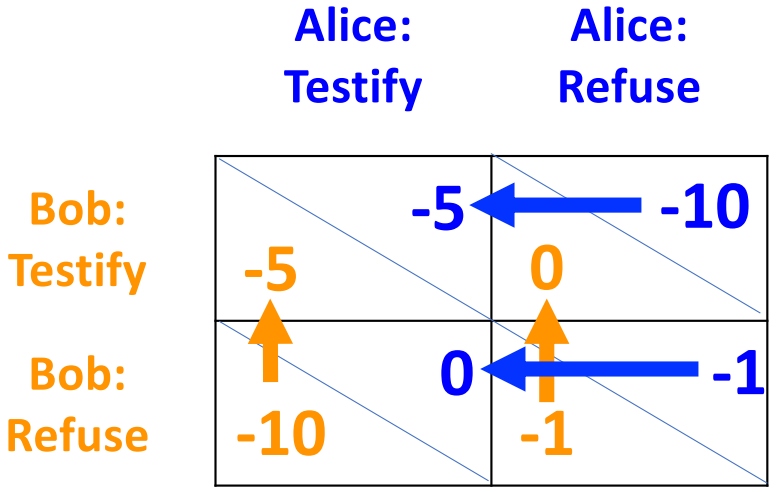
- (Stag,Stag) is Pareto-optimal: one could increase Alice's value, but only by decreasing Bob's value.
- (Alligator,Alligator) is Pareto-optimal: one could increase Bob's value, but only by decreasing Alice's value.

# Dominant strategy

If you didn't know in advance what your opponent was going to do, what would you do?

- If Bob knew that Alice was going to refuse, then it be rational for Bob to testify (he'd get 0 years, instead of 1).
- If Bob knew that Alice was going to testify, then it would still be rational for him to testify (he'd get 5 years, instead of 10).

A ***dominant strategy*** is an action that minimizes cost, for one player, regardless of what the other player does.





By Monogram Pictures, Public Domain, https://commons.wikimedia.org/w/index.php?curid=50338507

# Mixed-strategy Nash Equilibrium

| Bob \ Alice | Defect w/ Prob. $1-p$ | Coop. w/ Prob. $p$ |
|---|---|---|
| Defect w/ Prob. $1-q$ | $a$ / $w$ | $b$ / $x$ |
| Coop. w/ Prob. $q$ | $c$ / $y$ | $d$ / $z$ |

For Bob, random selection is rational only if he can't improve his winnings by definitively choosing one action or the other. If Alice's strategy is to cooperate w/probability p,

- If Bob defects, he expects to win $(1-p)w + px$.
- If Bob cooperates, he expects to win $(1-p)y + pz$.

So

- it's only logical for Bob to use a mixed strategy if $(1-p)w + px = (1-p)y + pz$.

# Outline: Exam 3 will be multiple-choice, on prairielearn, including 24 problems

- 4 problems: Midterm 1 material
- 4 problems: Midterm 2 material
- 4-5 problems: minimax, alpha-beta, expectiminimax
- 3-4 problems: game theory
- 5-7 problems: MDP, reinforcement learning
- 1 problem: AI fairness (definitions of fairness; causal graphs)
- 1 problem: computer vision (similar triangles; convolution)

# The Bellman equation

- Recursive relationship between the utilities of successive states:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')$$

- For *N* states, we get *N* **nonlinear** equations in *N* unknowns
  - Known quantities: $P(s'|s,a)$, $R(s)$, and $\gamma$. Unknowns: $U(s)$.
  - Solving these N equations solves the MDP.
  - Nonlinear -> no closed-form solution.
    - If it weren't for the "max," this would be N linear equations in N unknowns. We could solve it by just inverting an NxN matrix.
    - The "max" means that there is no closed-form solution. Need to use an iterative solution method, which might not converge to the globally optimum soluton.
  - Two solution methods: **value iteration** and **policy iteration**

# Method 1: Value iteration

- Start out with iteration $i = 0$, every $U_i\ (s)\ =\ 0$
- Iterate until convergence
  - During the $i^{th}$ iteration, update the utility of each state according to this rule:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')$$

- In the limit of infinitely many iterations, guaranteed to find the correct utility values.
  - Error decreases exponentially, so in practice, don't need an infinite number of iterations…

# Method 2: Policy Iteration

Start with some initial policy $\pi_0$ and alternate between the following steps:

- **Policy Evaluation:** calculate the utility of every state under the assumption that the given policy is fixed and unchanging.

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

- **Policy Improvement:** calculate a new policy $\pi_{i+1}$ based on the updated utilities.

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s, a) U^{\pi}(s')$$

# Mode-Based Reinforcement Learning

**The observation-model-policy loop:**

1. Observation: Follow some initial policy, to guide your actions. Create a **replay buffer**, i.e., a list of observed (s,a,R,s') tuples.
2. Model: Try to learn $P(s'|s,a)$ and $R(s)$.
3. Policy: Use your estimated $P(s'|s,a)$ and $R(s)$ to decide on a new policy, and repeat.

# How to trade off exploration vs. exploitation

**Epsilon-first strategy**: when you reach state $s$, check how many times you've tested each of its available actions.

- **Explore for the first $\epsilon$ trials**: If the least-explored action has been tested fewer than $\epsilon$ times, then perform that action ($\epsilon$ is an integer).
- **Exploit thereafter:** Once you've finished exploring, start exploiting (work to maximize your personal utility).

**Epsilon-greedy strategy**: in every state, every time, forever,

- **Explore with probability $0 < \epsilon < 1$**: choose any action, uniformly at random.
- **Exploit with probability $(1 - \epsilon)$**: choose the action with the highest expected utility, according to your current estimates.
- Guarantee: $P(s'|s, a)$ converges to its true value as #trials $\rightarrow \infty$.

# Q-learning

Q-learning is a model-free reinforcement learning strategy. It learns a function Q(s,a) such that $U(s) = \max_{a \in A(s)} Q(s,a)$. Bellman's equation therefore becomes:

$$Q(s,a) = R(s) + \gamma \sum_{s\prime} P(s'|s,a) \max_{a\prime \in A(s\prime)} Q(s',a')$$

Remember, it has these steps::

- Collect our current reward, R(s)
- Discount all future rewards by $\gamma$
- Make a transition to a future state, s', according to P(s'|s,a)
- Choose the optimum action, a', from state s', and collect all future rewards.

# TD learning

1. When you reach state s, use your current exploration versus exploitation policy, $\pi_t(s)$, to choose some action $a = \pi_t(s)$.
2. Observe the tuple (s,a,R,s'), and calculate:

$$Q_{local}(s,a) = R_t(s) + \gamma \max_{a' \in A(s')} Q_t(s',a')$$

3. Calculate the time difference, and update:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\big(Q_{local}(s,a) - Q_t(s,a)\big)$$

Repeat.

# On-policy learning: SARSA

We can create an "on-policy learning" algorithm by deciding in advance which action ($a$') we'll perform in state $s$', and then using that action in the update equation:

1. Use your current exploration versus exploitation policy, $\pi_t(s)$, to choose some action $a = \pi_t(s)$.
2. Observe the state $s$' that you end up in, and then use your current policy to choose $a' = \pi_t(s')$.
3. Instead of the observation (s,a,R,s'), you now have the observation (s,a,R,s',a'). Calculate Qlocal and the update equation as:

$$Q_{local}(s,a) = R_t(s) + \gamma Q_t(s',a')$$

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\big(Q_{local}(s,a) - Q_t(s,a)\big)$$

4. Go to step 2.

# Deep Q-learning

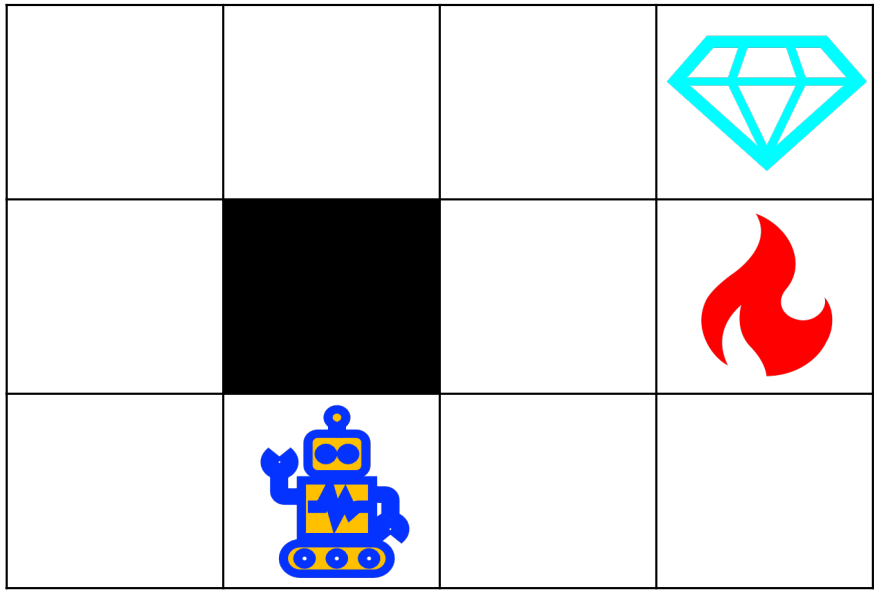
Let's define deep Q learning using the same $Q_{local}$:

$$\mathcal{L} = \frac{1}{2} E[(Q(s,a) - Q_{local}(s,a))^2]$$

where $Q_{local}(s,a)$ is:

$$Q_{local}(s,a) = R_t(s) + \gamma \max_{a'} Q(s',a')$$

Now we have an L that depends only on things we know ($Q(s,a)$, $R_t(s)$, and $Q(s',a')$), so it can be calculated, differentiated, and used to update the neural network.



# Policy learning methods

- Suppose that $s$ is continuous, but $a$ is discrete (e.g., a one-hot vector).
- Then learning the policy directly can be much faster than learning Q values.
- We can train a neural network for a stochastic policy---a policy that chooses an action at random, using the probability distribution:

$$\pi(s,a) = \frac{e^{f(s,a)}}{\sum_{a'} e^{f(s,a')}}$$

Actors from the Comédie Française, by Antoine Watteau, 1720. Public Domain, https://commons.wikimedia.org/w/index.php?curid=15418670

# Actor-critic algorithm

So let's train two neural nets!

- $Q(s, a)$ is the critic, and is trained according to the deep Q-learning algorithm (MMSE).
- $\pi(s, a)$ is the actor, and is trained to satisfy the critic:

The Critic, by Lajos Tihanyi. Oil on canvas, 1916. Public Domain, https://commons.wikimedia.org/w/index.php?curid=17837438

$$\pi(s, a) = \text{argmax} \sum_{a} \pi(s, a) Q(s, a)$$

# Outline: Exam 3 will be multiple-choice, on prairielearn, including 24 problems

- 4 problems: Midterm 1 material
- 4 problems: Midterm 2 material
- 4-5 problems: minimax, alpha-beta, expectiminimax
- 3-4 problems: game theory
- 5-7 problems: MDP, reinforcement learning
- 1 problem: AI fairness (definitions of fairness; causal graphs)
- 1 problem: computer vision (similar triangles; convolution)

# Standard Definitions of Fairness in AI

**Demographic Parity:**

The probability of a positive outcome is the same, regardless of protected attribute.

$$P\left(\hat{Y} = 1 | A = a\right) = P\left(\hat{Y} = 1 | A = a'\right) \quad \forall a, a'$$

**Predictive Parity:**

Precision is the same, regardless of protected attribute.

$$P\left(Y = 1 | \hat{Y} = 1, A = a\right) = P\left(Y = 1 | \hat{Y} = 1, A = a'\right) \quad \forall a, a'$$

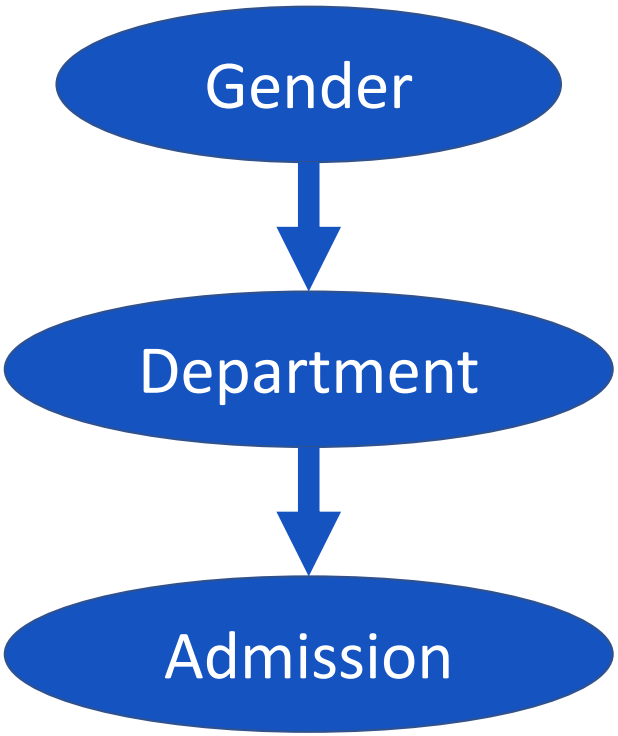
**Error Rate Balance:**

Recall is the same, regardless of protected attribute.

$$P\left(\hat{Y} = 1 | Y = 1, A = a\right) = P\left(\hat{Y} = 1 | Y = 1, A = a'\right) \quad \forall \hat{y}, a, a'$$

# State-of-the-art Solution: Explain Your Assumptions to Your Users

- The Transparency Dogma says that an algorithm's assumptions should be published in a way that users can understand, so that users can participate in a public debate about the fairness of the assumptions.
- A common way to do this is by drawing a Bayesian network.
- Example shown: the Bickel-Hammel-O'Connell data. Admission was conditionally independent of Gender, given Department.



# Outline: Exam 3 will be multiple-choice, on prairielearn, including 24 problems

- 4 problems: Midterm 1 material
- 4 problems: Midterm 2 material
- 4-5 problems: minimax, alpha-beta, expectiminimax
- 3-4 problems: game theory
- 5-7 problems: MDP, reinforcement learning
- 1 problem: AI fairness (definitions of fairness; causal graphs)
- 1 problem: computer vision (similar triangles; convolution)

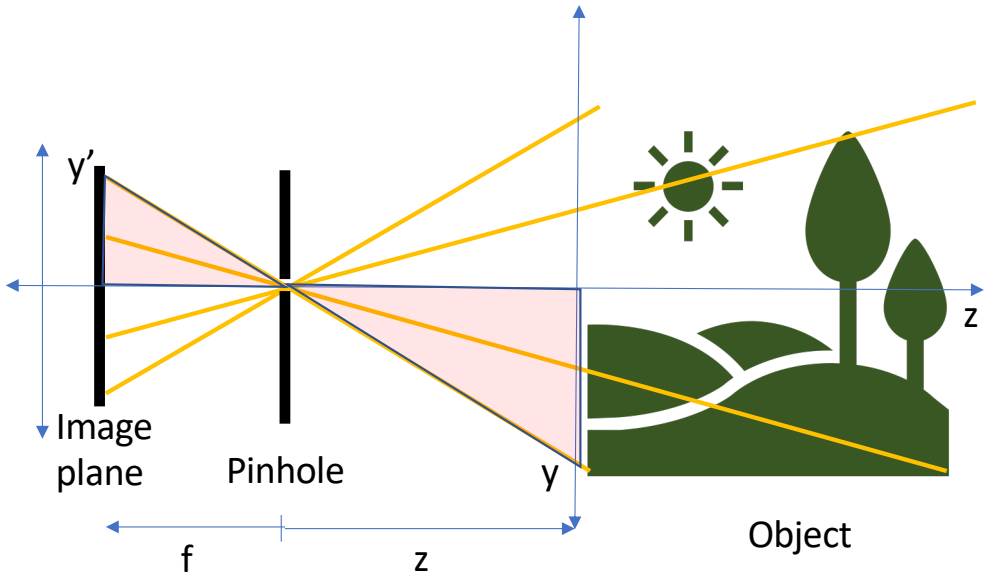
# The pinhole camera equations

- These are similar triangles! So

$$\frac{y'}{f} = \frac{-y}{z}, \qquad \frac{x'}{f} = \frac{-x}{z}$$

- Solving for (x',y'), we get the pinhole camera equations:

$$x' = \frac{-fx}{z}, \qquad y' = \frac{-fy}{z}$$



# Vanishing point

- Suppose we are imaging a couple of parallel lines, for which x and y depend on z:

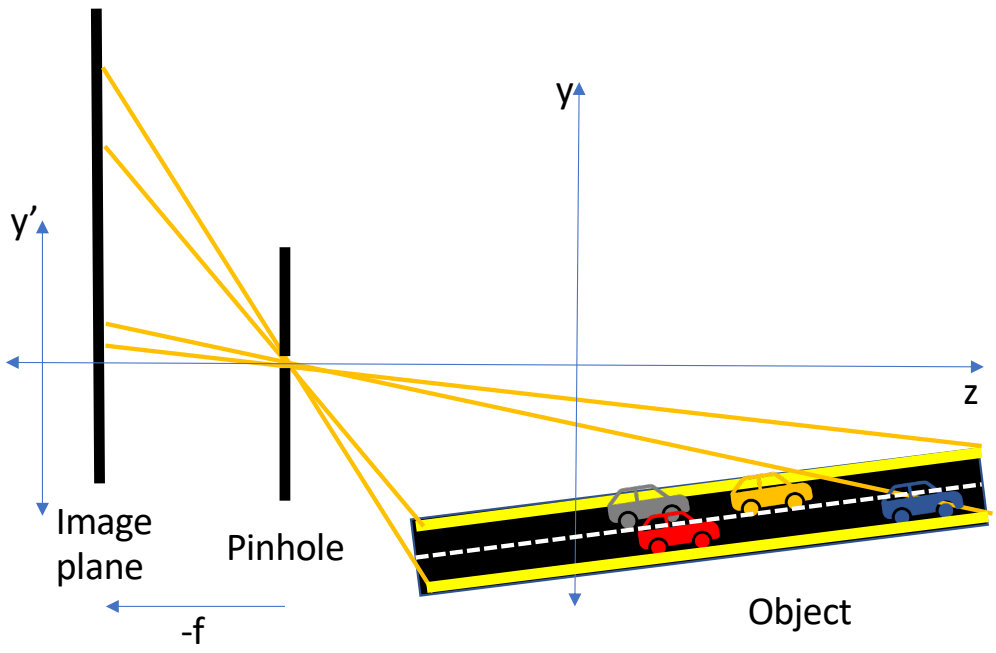
Line 1: $ax_1 + z = b_1, cy_1 + z = d_1$

Line 2: $ax_2 + z = b_2, cy_2 + z = d_2$

So

Line 1: $\lim_{z\to\infty} {x_1}' = \frac{f}{a}, \lim_{z\to\infty} y_1' = \frac{f}{c}$

Line 2: $\lim_{z\to\infty} {x_2}' = \frac{f}{a}, \lim_{z\to\infty} y_2' = \frac{f}{c}$



- A **convolution** is exactly the same thing as a **weighted local average**. We give it a special name, because we will use it very often. It's defined as:

$$y[n] = \sum_m g[m]f[n-m] = \sum_m g[n-m]f[m]$$

- We use the symbol $*$ to mean "convolution:"

$$y[n] = g[n] * f[n] = \sum_m g[m]f[n-m] = \sum_m g[n-m]f[m]$$

# Difference of Gaussians

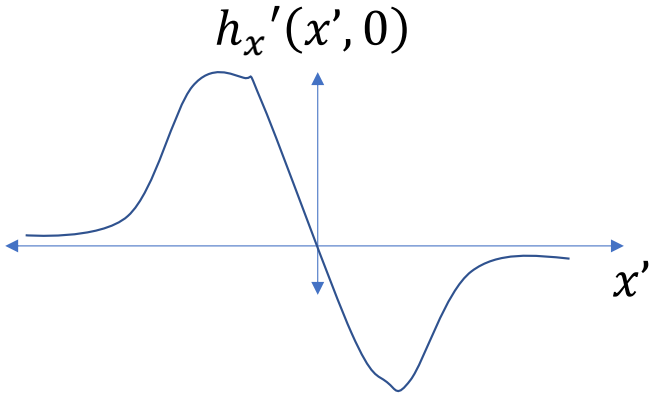
A "difference-of-Gaussians" filter is created by subtracting two Gaussian-blur filters, like this:

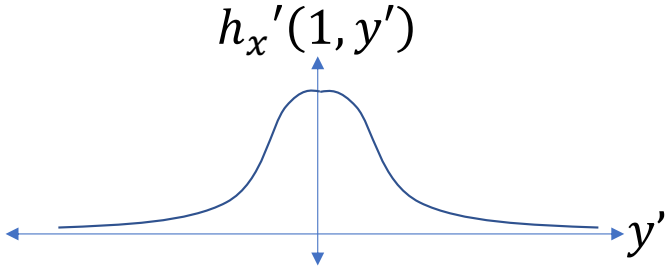
$$h[m,n] = \frac{1}{2\pi\sigma^2} e^{-\left(\left(\frac{m}{\sigma}\right)^2 + \left(\frac{n}{\sigma}\right)^2\right)}$$

$$h_x'[m,n] = \frac{h[m,n+1] - h[m,n-1]}{2}$$

$$h_y'[m,n] = \frac{h[m+1,n] - h[m-1,n]}{2}$$

A "difference-of-Gaussians" filter looks kind of like this:



# Some Sample Problems

- Review exam Question 3: Alpha-beta pruning
- Review exam Question 10: Game theory
- Review exam Question 14: MDP

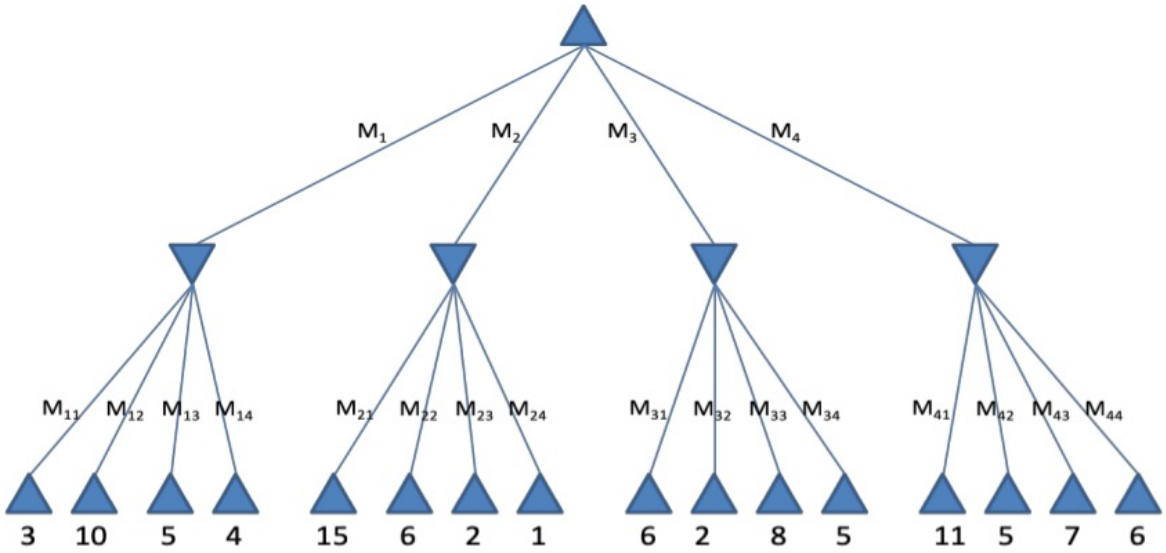
# Question 3: alpha-beta pruning

a) Write down minimax value of each node.

b) How will the game proceed?

c) Cross out branches that would be pruned by alpha-beta.

d) Suppose you could re-order the branches using some heuristic. What re-ordering would maximize the number pruned?



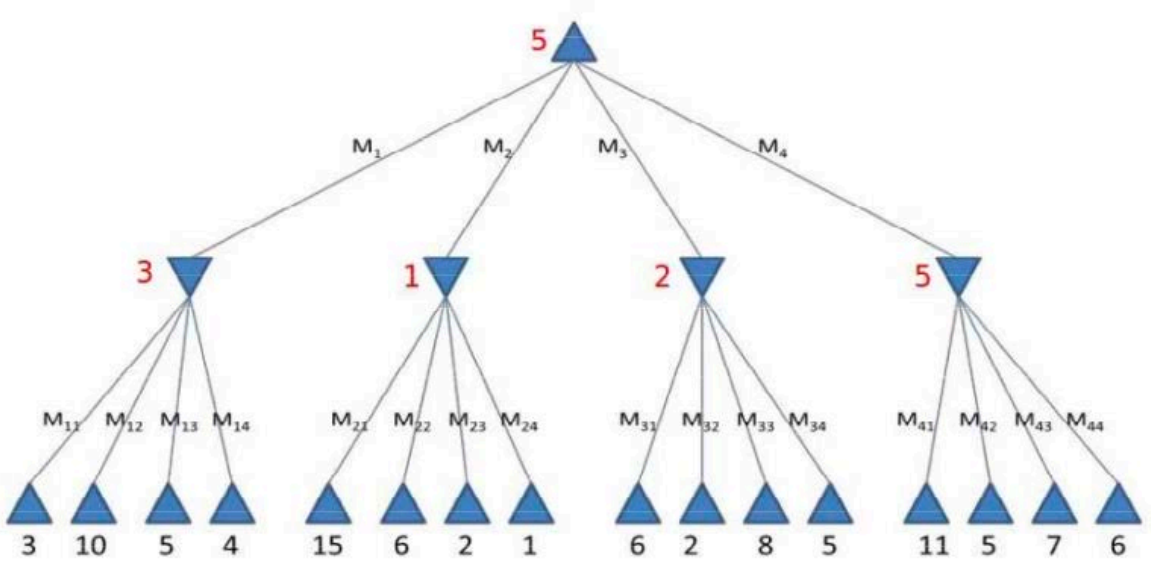
# Question 3: alpha-beta pruning

a) Write down minimax value of each node.

b) How will the game proceed?

c) Cross out branches that would be pruned by alpha-beta.

d) Suppose you could re-order the branches using some heuristic. What re-ordering would maximize the number pruned?



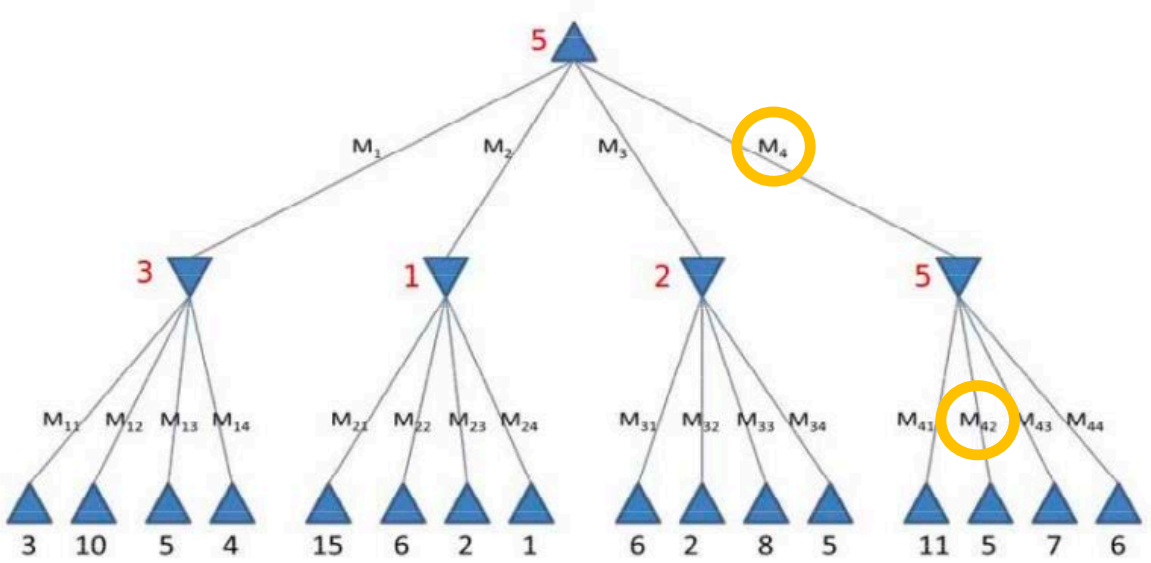
# Question 3: alpha-beta pruning

a) Write down minimax value of each node.

b) How will the game proceed?

c) Cross out branches that would be pruned by alpha-beta.

d) Suppose you could re-order the branches using some heuristic. What re-ordering would maximize the number pruned?



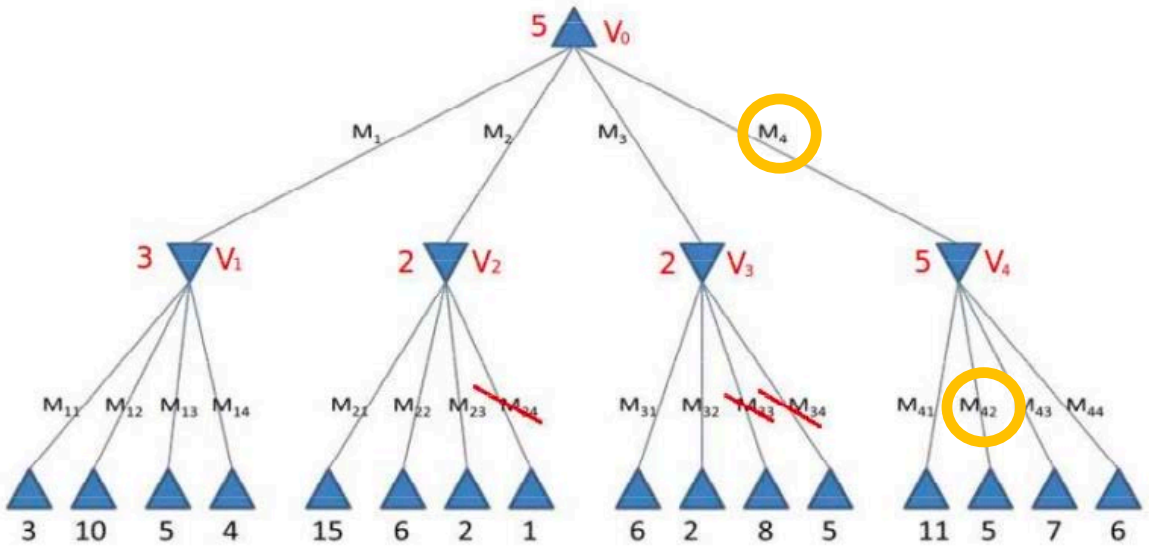
# Question 3: alpha-beta pruning

a) Write down minimax value of each node.

b) How will the game proceed?

c) Cross out branches that would be pruned by alpha-beta.

d) Suppose you could re-order the branches using some heuristic. What re-ordering would maximize the number pruned?



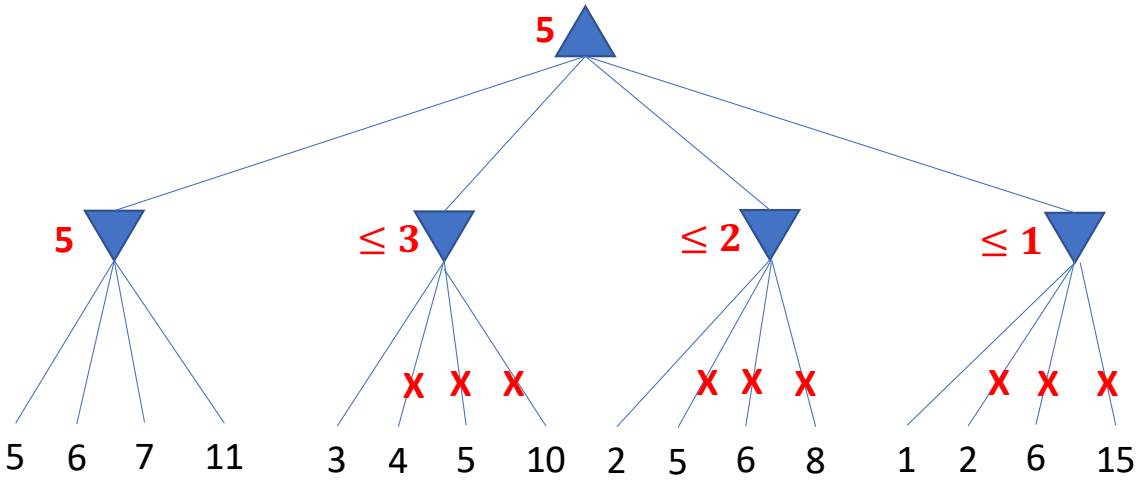
# Question 3: alpha-beta pruning

a) Write down minimax value of each node.

b) How will the game proceed?

c) Cross out branches that would be pruned by alpha-beta.

d) Suppose you could re-order the branches using some heuristic. What re-ordering would maximize the number pruned?



# Question 10: Game theory

Suppose that both Alice and Bob want to go from one place to another. There are two routes, R1 and R2. The utility of a route is inversely proportional to the number of cars on the road. For instance, if both Alice and Bob choose route R1, the utility of R1 for each of them is 1/2.

- Write out the payoff matrix.
- Is this a zero-sum game?
- Find dominant strategies, if any. If there are no dominant strategies, explain why not.
- Find pure strategy equilibria, if any. If there is no pure strategy equilibrium, explain why not.
- Find the mixed-strategy equilibrium.

# Question 10: Game theory

Suppose that both Alice and Bob want to go from one place to another. There are two routes, R1 and R2. The utility of a route is inversely proportional to the number of cars on the road. For instance, if both Alice and Bob choose route R1, the utility of R1 for each of them is 1/2.

- Write out the payoff matrix.

| | B:R1 | B:R2 |
|---|---|---|
| A:R1 | 0.5,0.5 | 1,1 |
| A:R2 | 1,1 | 0.5,0.5 |

# Question 10: Game theory

Suppose that both Alice and Bob want to go from one place to another. There are two routes, R1 and R2. The utility of a route is inversely proportional to the number of cars on the road. For instance, if both Alice and Bob choose route R1, the utility of R1 for each of them is 1/2.

- Write out the payoff matrix.

| | B:R1 | B:R2 |
|---|---|---|
| A:R1 | 0.5,0.5 | 1,1 |
| A:R2 | 1,1 | 0.5,0.5 |

- Is this a zero-sum game?

No. The rewards for Alice and Bob don't sum to zero within each square.

# Question 10: Game theory

| | B:R1 | B:R2 |
|---|---|---|
| A:R1 | 0.5,0.5 | 1,1 |
| A:R2 | 1,1 | 0.5,0.5 |

- Find dominant strategy, if any. If there is none, explain why not.

There is no dominant strategy. The best action for Alice depends on what Bob does. The best action for Bob depends on what Alice does.

- Find pure-strategy equilibria, if any. If there is none, explain why not.

There are two. If Alice takes R1 and Bob takes R2, then Alice has no reason to want to change. Similarly, if Alice takes R2 and Bob takes R1, then Alice has no reason to want to change.

# Question 10: Game theory

| | B:R1 | B:R2 |
|---|---|---|
| A:R1 | 0.5,0.5 | 1,1 |
| A:R2 | 1,1 | 0.5,0.5 |

- Find the mixed-strategy equilibrium.

Suppose Alice chooses R2 with probability p. Then it is rational for Bob to choose randomly only if

$$0.5(1-p)+p=(1-p)+0.5p$$

So p=0.5. Since it is now rational for Bob to choose randomly, suppose he chooses R2 with probability q. Then it is rational for Alice to choose randomly only if

$$0.5(1-q)+q=(1-q)+0.5q$$

So random choice is a Nash equilibrium only if p=0.5, q=0.5.

# Question 14: MDP

Consider a 3x2 gridworld, with rewards for each state as shown at right. Robot starts in the square with R(s)=0; if it reaches the square with R(s)=-1 or R(s)=1, the game ends. Transition probability is

$$P(s'|s,a) = \begin{cases} 0.8 & s' = a \\ 0.2 & s' = s \end{cases}$$

| | |
|---|---|
| -0.04 | -0.04 |
| -1 | 1 |
| 0 | -0.04 |

- What is U(s) after second round of value iteration, with $\gamma = 1$?
- After how many rounds of value iteration is U(s)>0, for the first time, in the starting state?

# Question 14: MDP

- What is U(s) after second round of value iteration, with $\gamma = 1$?

$$U_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) U_t(s')$$

So if we start with $U_0(s) = 0$, then we have $U_1(s) = R(s)$, and $U_2(s)$ as shown at right.

| | |
|---|---|
| $R(s) = -0.04$ | $R(s) = -0.04$ |
| $R(s) = -1$ | $R(s) = 1$ |
| $R(s) = 0$ | $R(s) = -0.04$ |

| | |
|---|---|
| $U_2(s) = -0.08$ | $U_2(s) = 0.752$ |
| $U_2(s) = -1$ | $U_2(s) = 1$ |
| $U_2(s) = -0.032$ | $U_2(s) = 0.752$ |

# Question 14: MDP

- After how many rounds of value iteration is U(s)>0, for the first time, in the starting state?

$$U_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) U_t(s')$$

$U_1(s) = R(s)$. $U_2(s)$ takes into account rewards that can be earned after taking up to one step. In general, $U_t(s)$ takes into account rewards that can be earned after taking up to t-1 steps. The R(s)=1 square is t-1=2 steps away from the start state, so $U_t(s)$ will be able to count that reward on iteration number t=3.

| | |
|---|---|
| $R(s) = -0.04$ | $R(s) = -0.04$ |
| $R(s) = -1$ | $R(s) = 1$ |
| $R(s) = 0$ | $R(s) = -0.04$ |

| | |
|---|---|
| $U_2(s) = -0.08$ | $U_2(s) = 0.752$ |
| $U_2(s) = -1$ | $U_2(s) = 1$ |
| $U_2(s) = -0.032$ | $U_2(s) = 0.752$ |

# Outline: Exam 3 will be multiple-choice, on prairielearn, including 24 problems

- 4 problems: Midterm 1 material
- 4 problems: Midterm 2 material
- 4-5 problems: minimax, alpha-beta, expectiminimax
- 3-4 problems: game theory
- 5-7 problems: MDP, reinforcement learning
- 1 problem: AI fairness (definitions of fairness; causal graphs)
- 1 problem: computer vision (similar triangles; convolution)