Game-playing: DeepBlue and AlphaGo

Brief history of gameplaying frontiers

- 1990s: Othello world champions refuse to play computers
- 1994: Chinook defeats Checkers world champion
- 1997: DeepBlue defeats world champion Gary Kasparov
- 2016: AlphaGo defeats world champion Lee Sedol

Today, we're going to talk about **DeepBlue** and **AlphaGo**.

DeepBlue

- In 1997, DeepBlue beat world champion Gary Kasparov at chess.

DeepBlue

- In 1997, DeepBlue beat world champion Gary Kasparov at chess.

- How?
  - **Minimax**
  - **Alpha-beta pruning**
  - **Evaluation function**
  - Sound familiar?

First, some review

Let's play a two-player game.

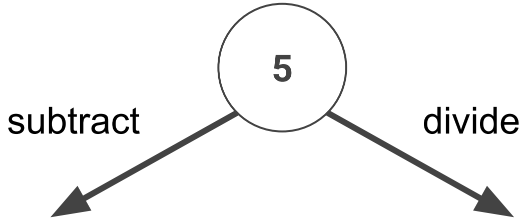
Start with **n=5**, and alternate turns.

- On every turn, player can either set **n = n - 1** or **n = floor(n/2)**
- The first player to set **n = 0** wins!

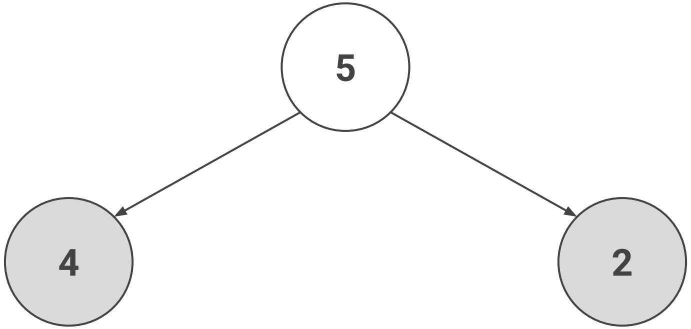
How can we model this?

Game trees

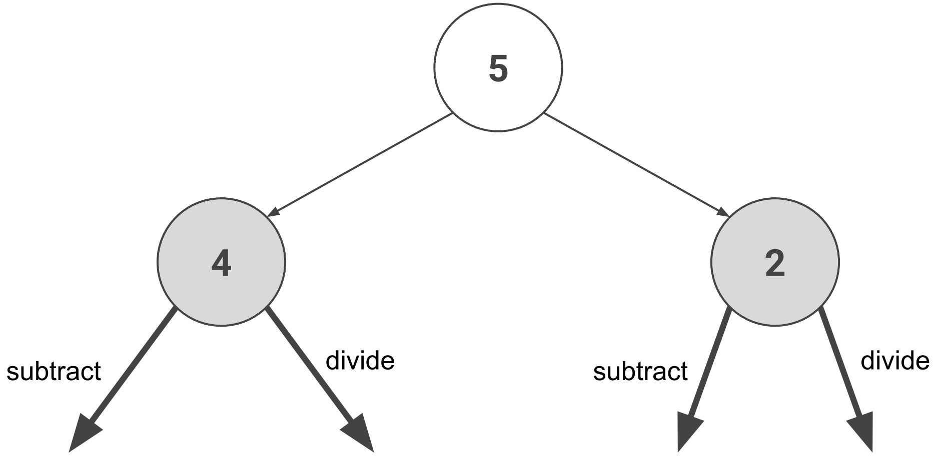
Game trees



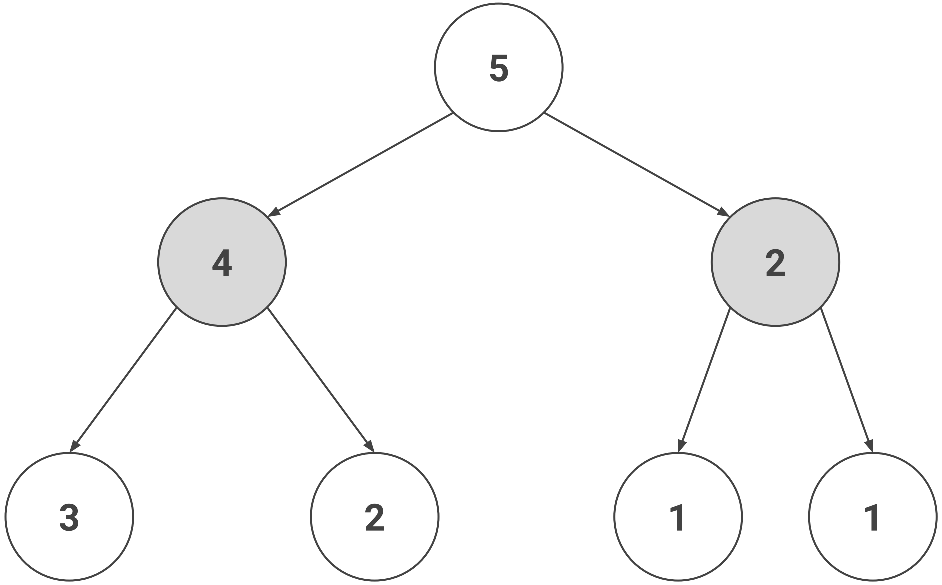
Game trees



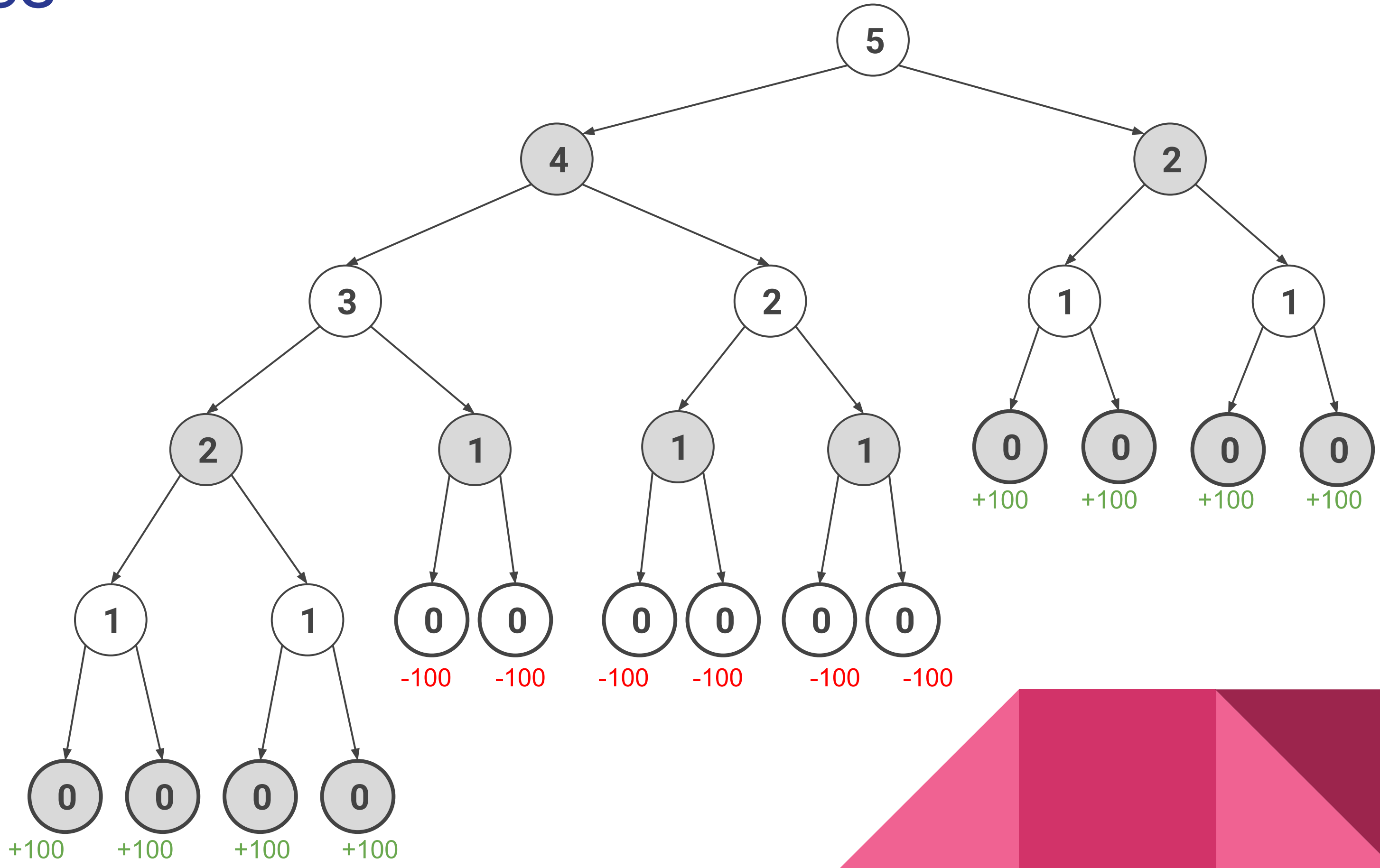
Game trees



Game trees



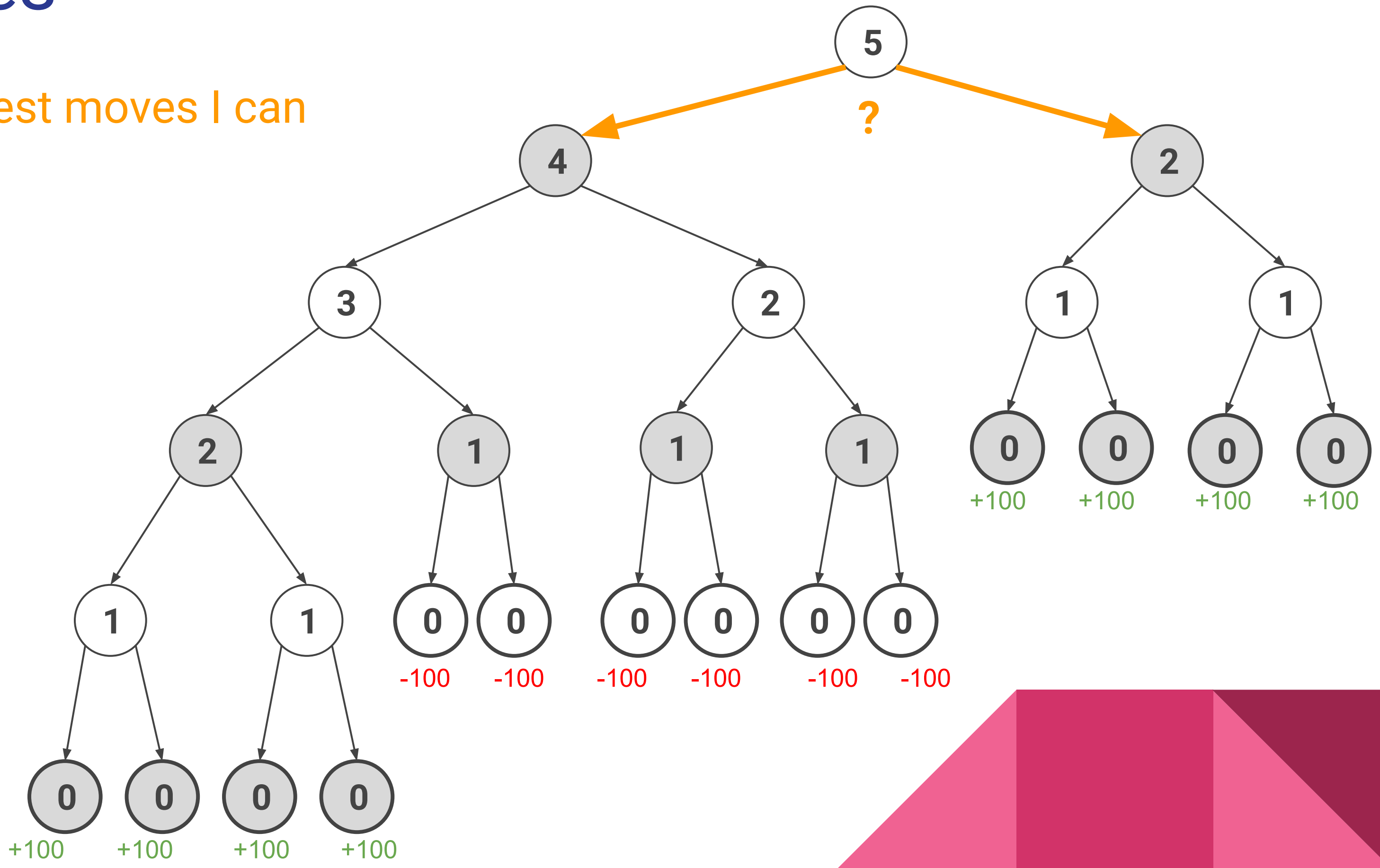
Game trees



5
4
2
3
2
1
1
2
1
1
1
0
0
0
0
+100
+100
+100
+100
1
1
0
0
0
0
0
0
-100
-100
-100
-100
-100
-100
0
0
0
0
+100
+100
+100
+100

Game trees

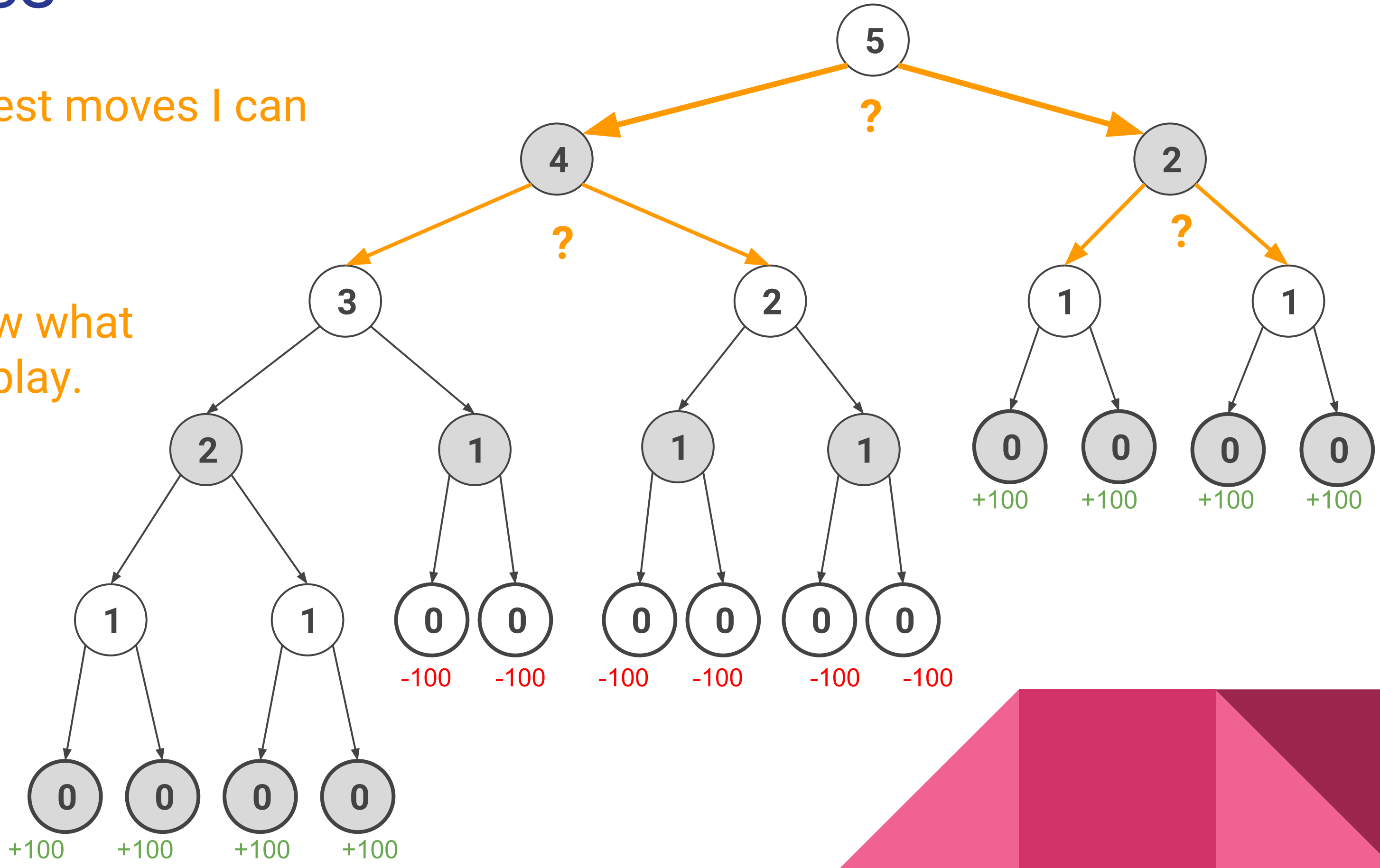
So what are the best moves I can play?



Game trees

So what are the best moves I can play?

Problem:
We also don't know what the opponent will play.



Expectimax

- We want to maximize our own utility.

Expectimax

- We want to maximize our own utility. If it's my turn, then:

$$V(s) = \max_{a \in Actions} V(Succ(s, a))$$

Expectimax

- We want to maximize our own utility. If it's my turn, then:

$$V(s) = \max_{a \in Actions} V(Succ(s, a))$$

= Take the action **a** that maximizes the utility of the resulting state.

Expectimax

- We want to maximize our own utility. If it's my turn, then:

$$V(s) = \max_{a \in Actions} V(Succ(s,a))$$

= Take the action **a** that maximizes the utility of the resulting state.

- We don't know what the enemy will do.

Expectimax

- We want to maximize our own utility. If it's my turn, then:

$$V(s) = \max_{a \in Actions} V(Succ(s,a))$$

= Take the action **a** that maximizes the utility of the resulting state.

- We don't know what the enemy will do. So let's guess!

$$V(s) = \sum_{a \in Actions} \pi_{opp}(s,a) V(Succ(s,a))$$

Expectimax

- We want to maximize our own utility. If it's my turn, then:

$$V(s) = \max_{a \in Actions} V(Succ(s,a))$$

= Take the action **a** that maximizes the utility of the resulting state.

- We don't know what the enemy will do. So let's guess!

$$V(s) = \sum_{a \in Actions} \underline{\pi_{opp}(s,a)} V(Succ(s,a))$$

Probability that our opponent will take action **a** from state **s**

Expectimax

- We want to maximize our own utility. If it's my turn, then:

$$V(s) = \max_{a \in Actions} V(Succ(s,a))$$

= Take the action **a** that maximizes the utility of the resulting state.

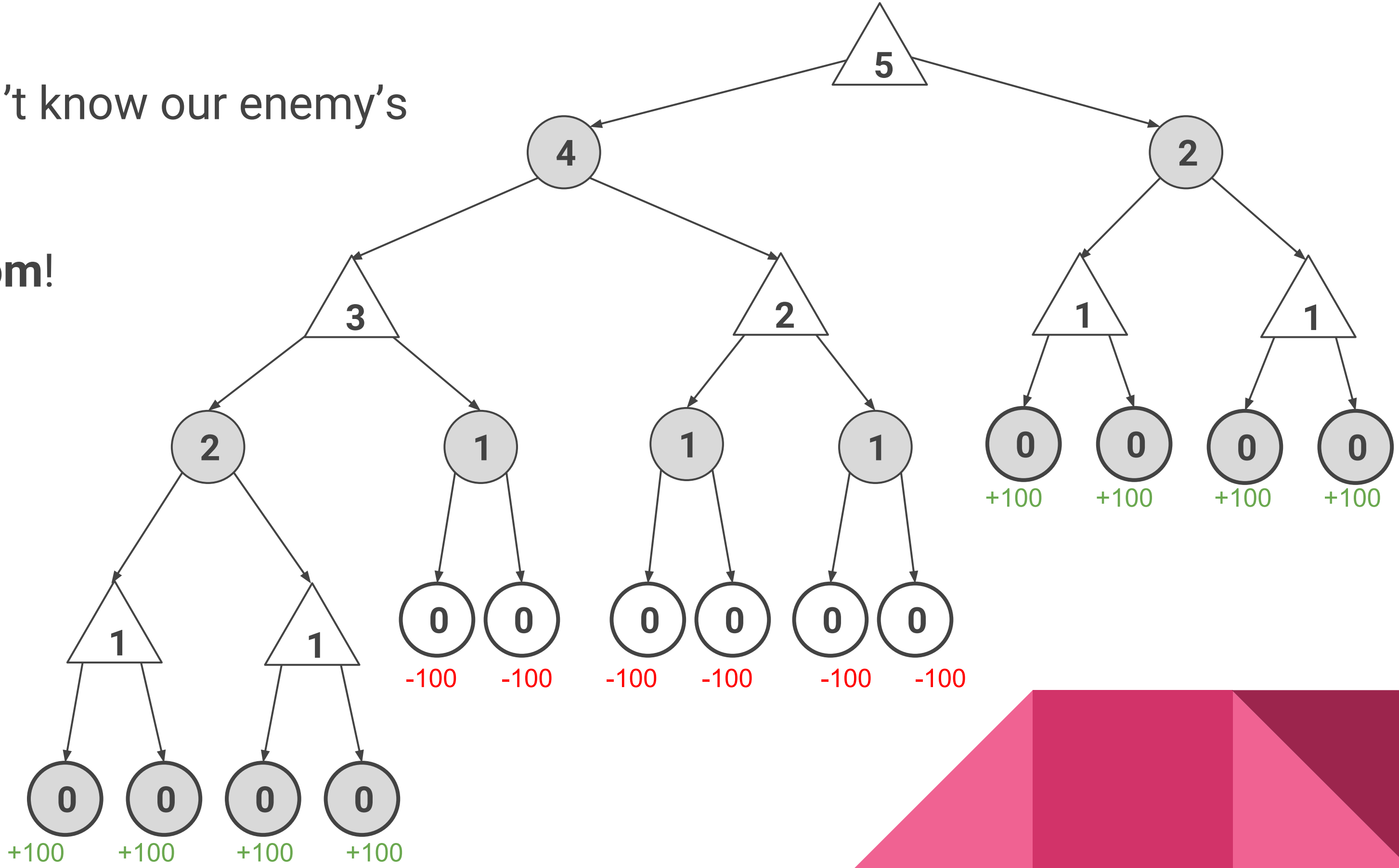
- We don't know what the enemy will do. So let's guess!

$$V(s) = \sum_{a \in Actions} \underbrace{\pi_{opp}(s,a)}_{\text{Probability that our opponent will take action } \mathbf{a} \text{ from state } \mathbf{s}} \underbrace{V(Succ(s,a))}_{\text{Utility of the next state.}}$$

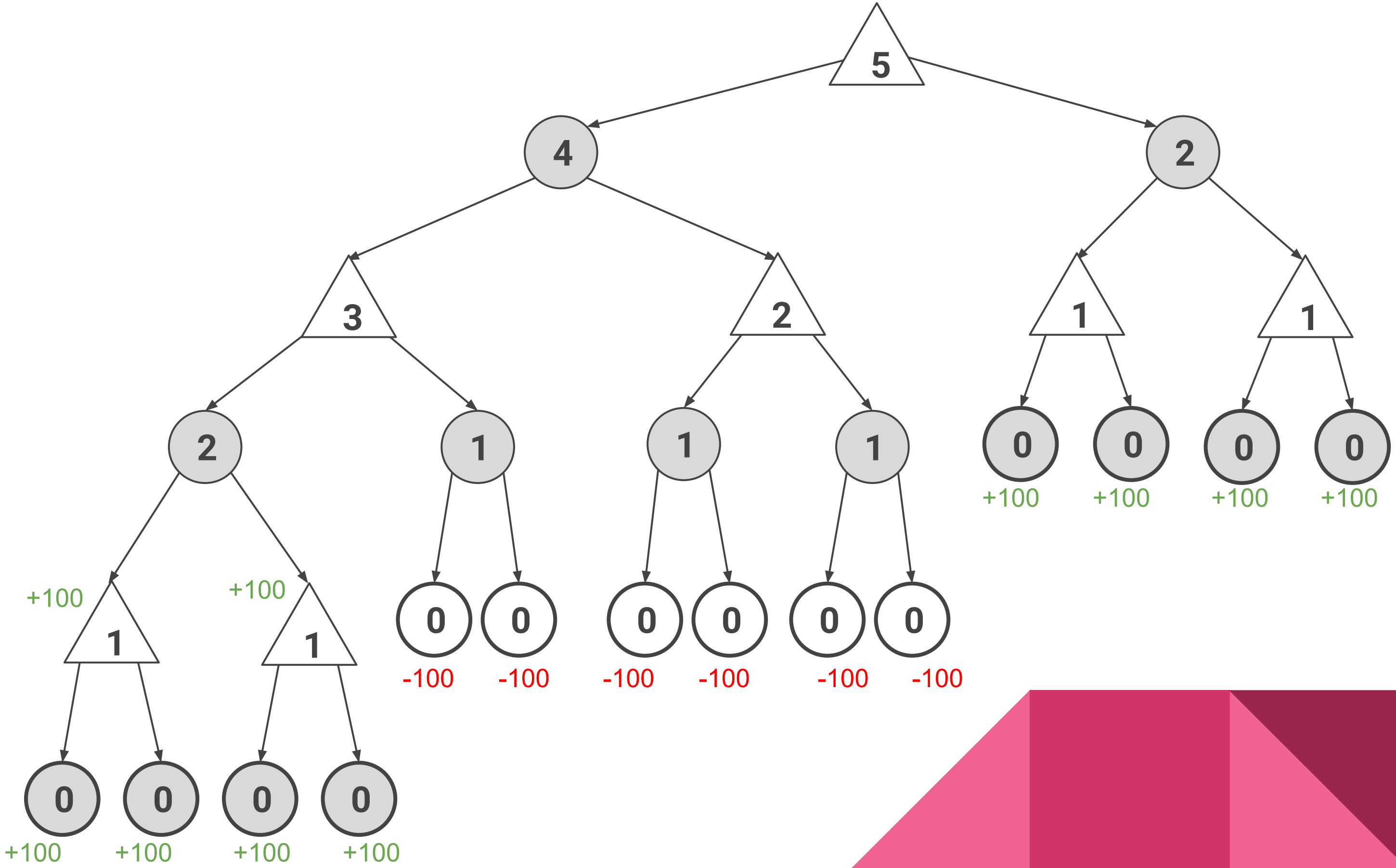
Expectimax

Let's say we don't know our enemy's policy at all.

Maybe it's **random**!

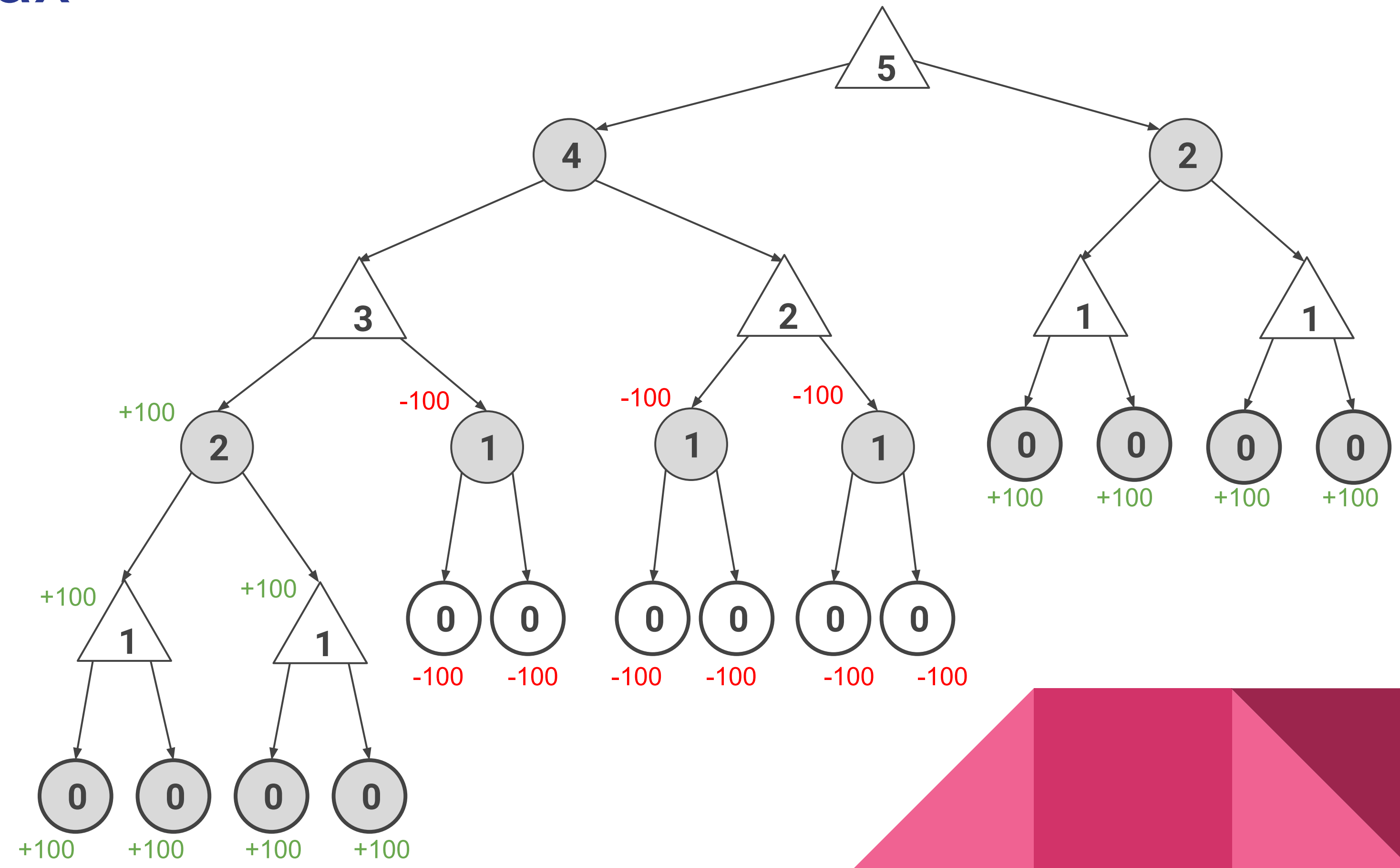


Expectimax

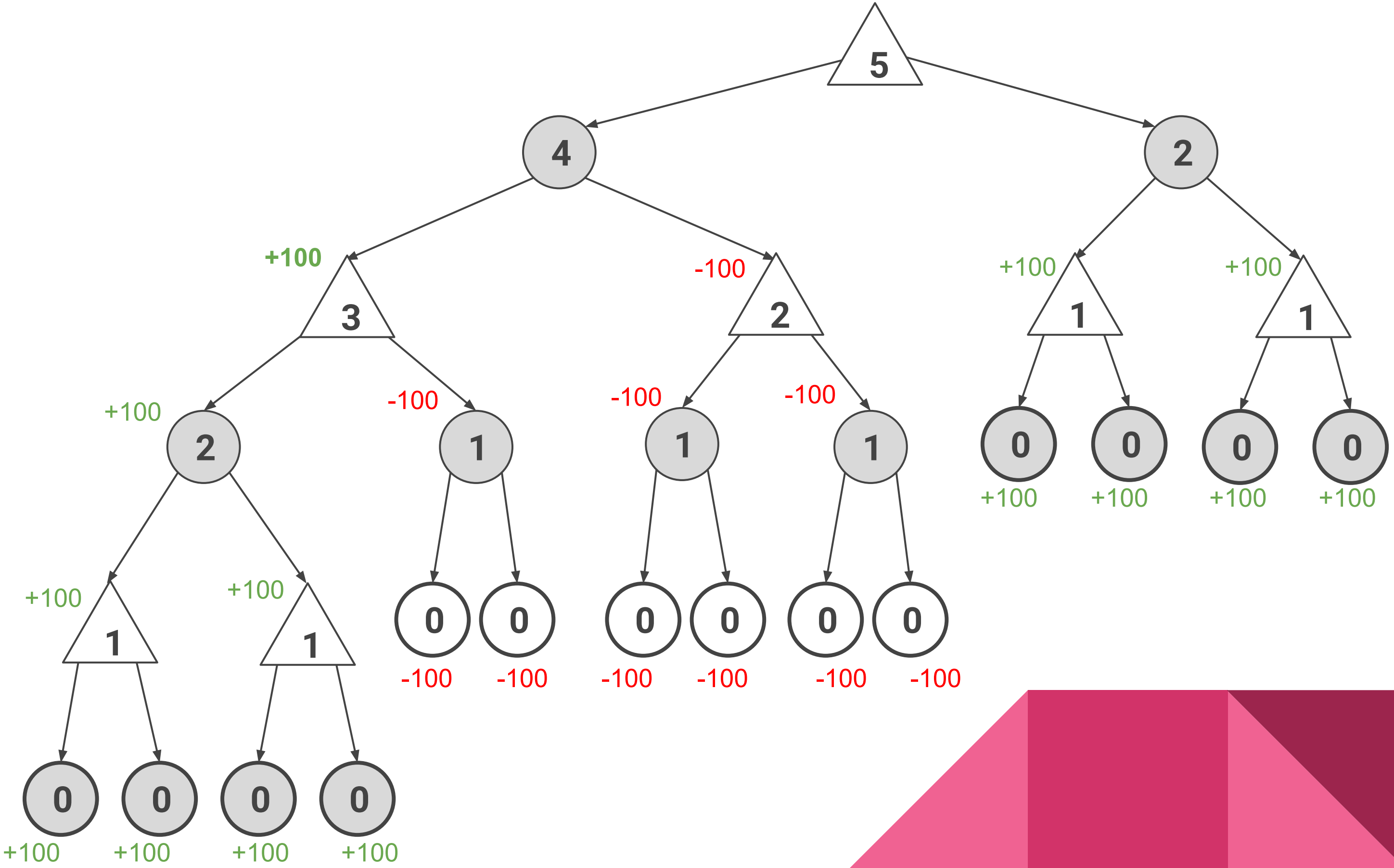


5
4
2
3
2
1
1
2
1
1
1
1
0
0
0
0
+100
+100
+100
+100
+100
+100
1
1
0
0
0
0
0
0
-100
-100
-100
-100
-100
-100
0
0
0
0
+100
+100
+100
+100

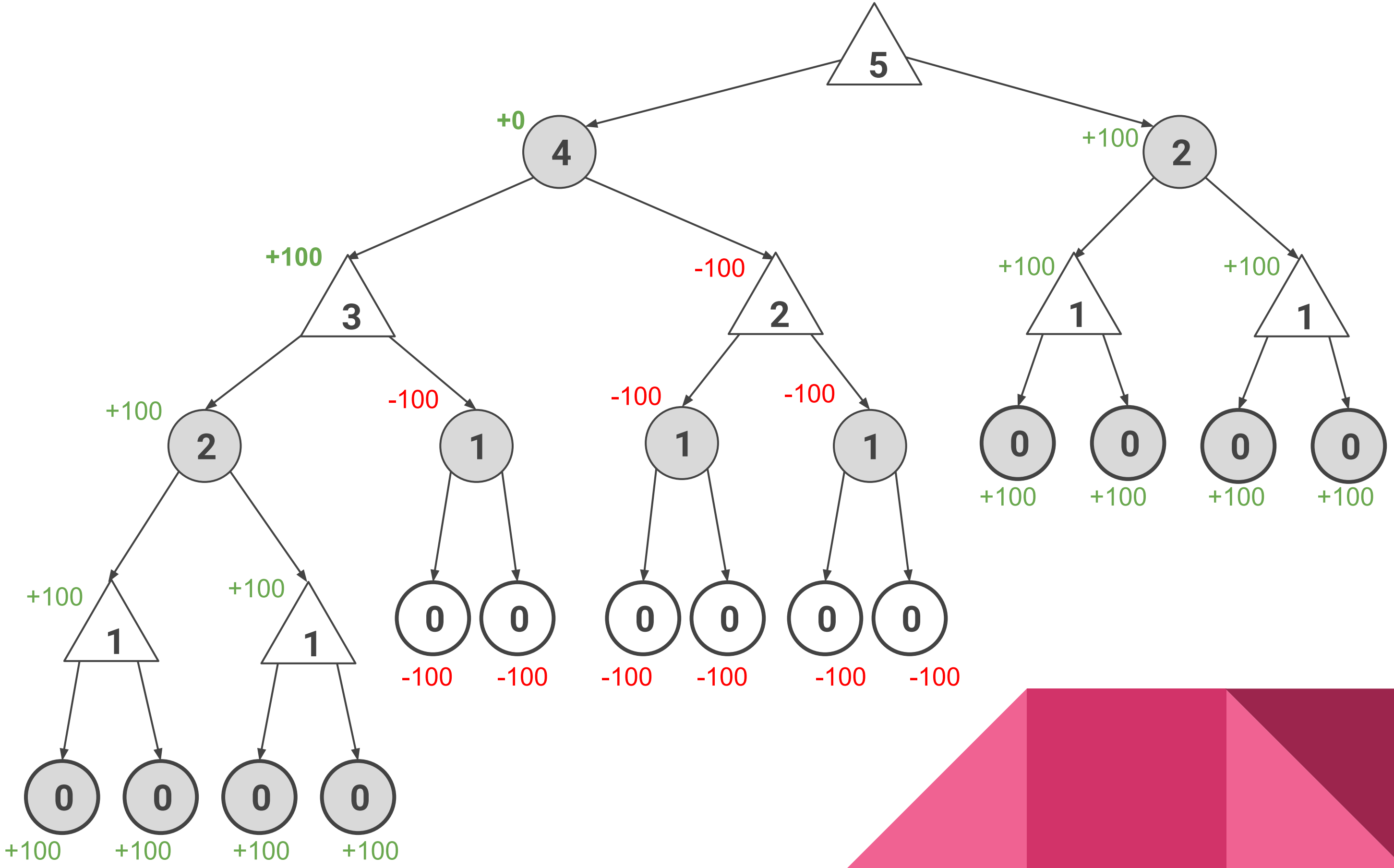
Expectimax


5
4
2
3
2
1
1
+100
2
-100
1
-100
1
-100
1
0
0
0
0
+100
+100
+100
+100
+100
1
+100
1
0
0
0
0
0
0
-100
-100
-100
-100
-100
-100
0
0
0
0
+100
+100
+100
+100

Expectimax

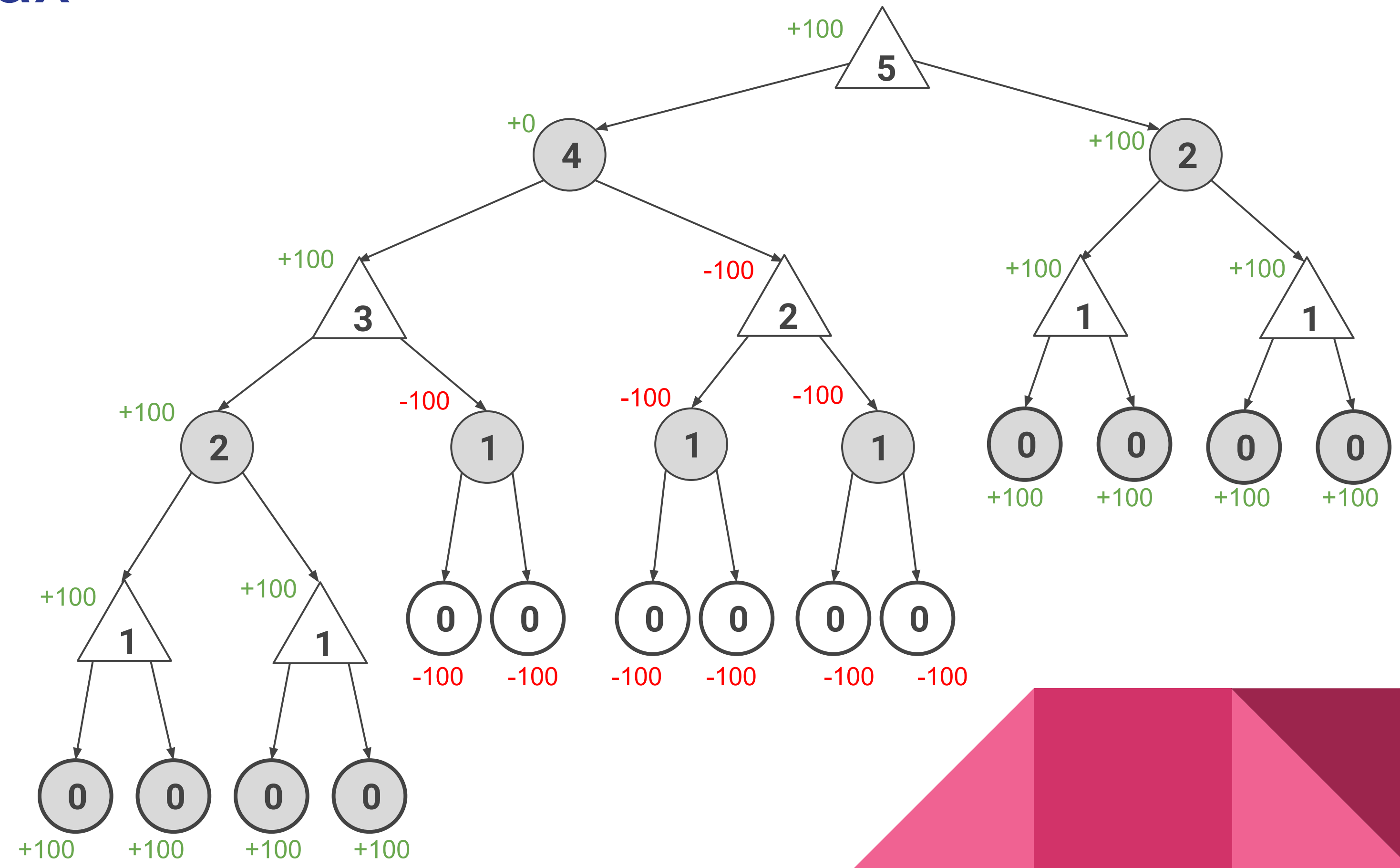

5
4
2
+100
3
-100
2
+100
1
+100
1
+100
2
-100
1
-100
1
-100
1
1
1
+100
1
0
0
0
0
+100
+100
+100
+100
0
0
0
0
0
0
-100
-100
-100
-100
-100
-100
0
0
0
0
+100
+100
+100
+100

Expectimax



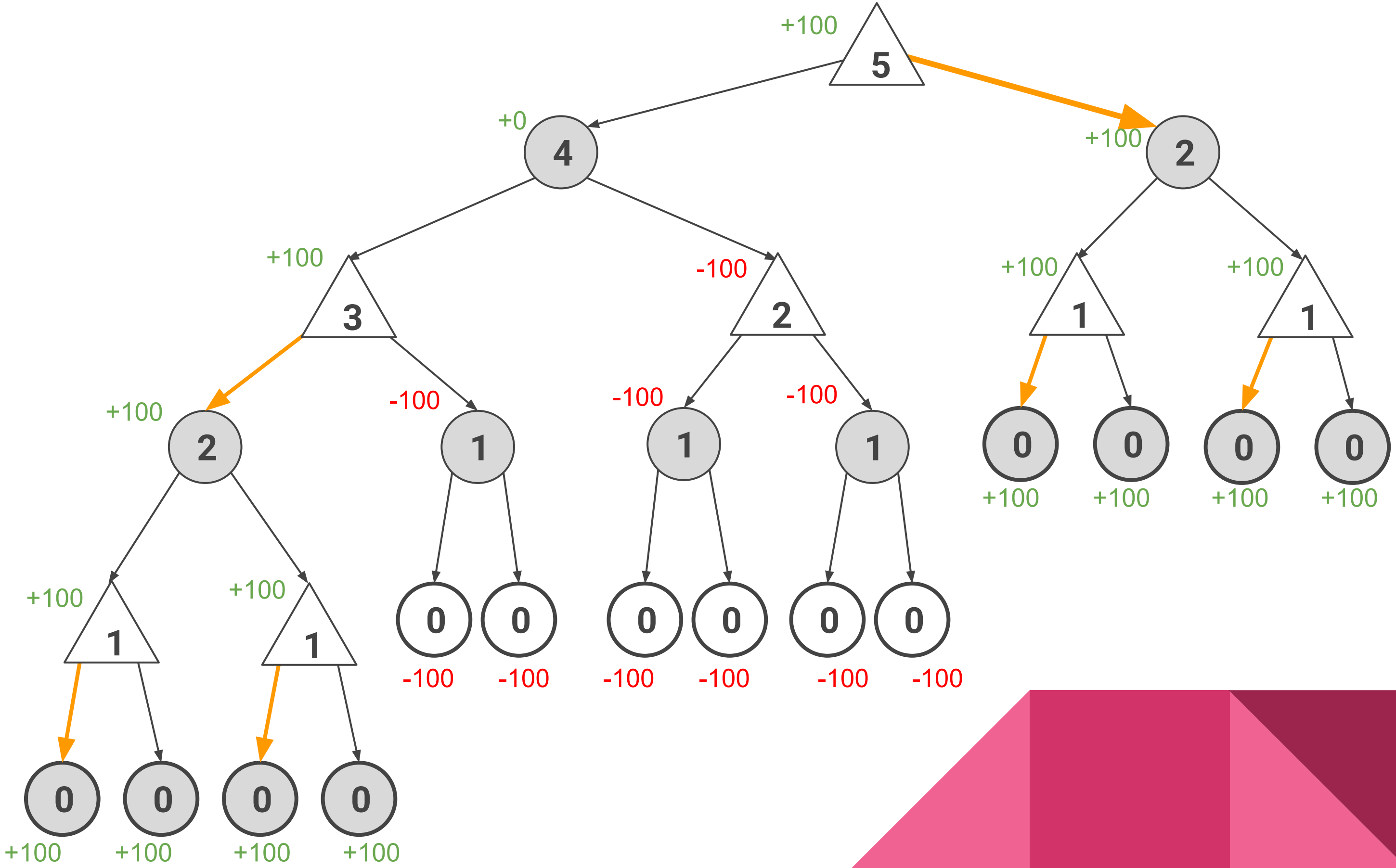
5
+0
4
+100
2
+100
3
-100
2
+100
1
+100
1
+100
2
-100
1
-100
1
-100
1
0
0
0
0
+100
+100
+100
+100
+100
1
+100
1
0
0
0
0
0
0
-100
-100
-100
-100
-100
-100
0
0
0
0
+100
+100
+100
+100

Expectimax



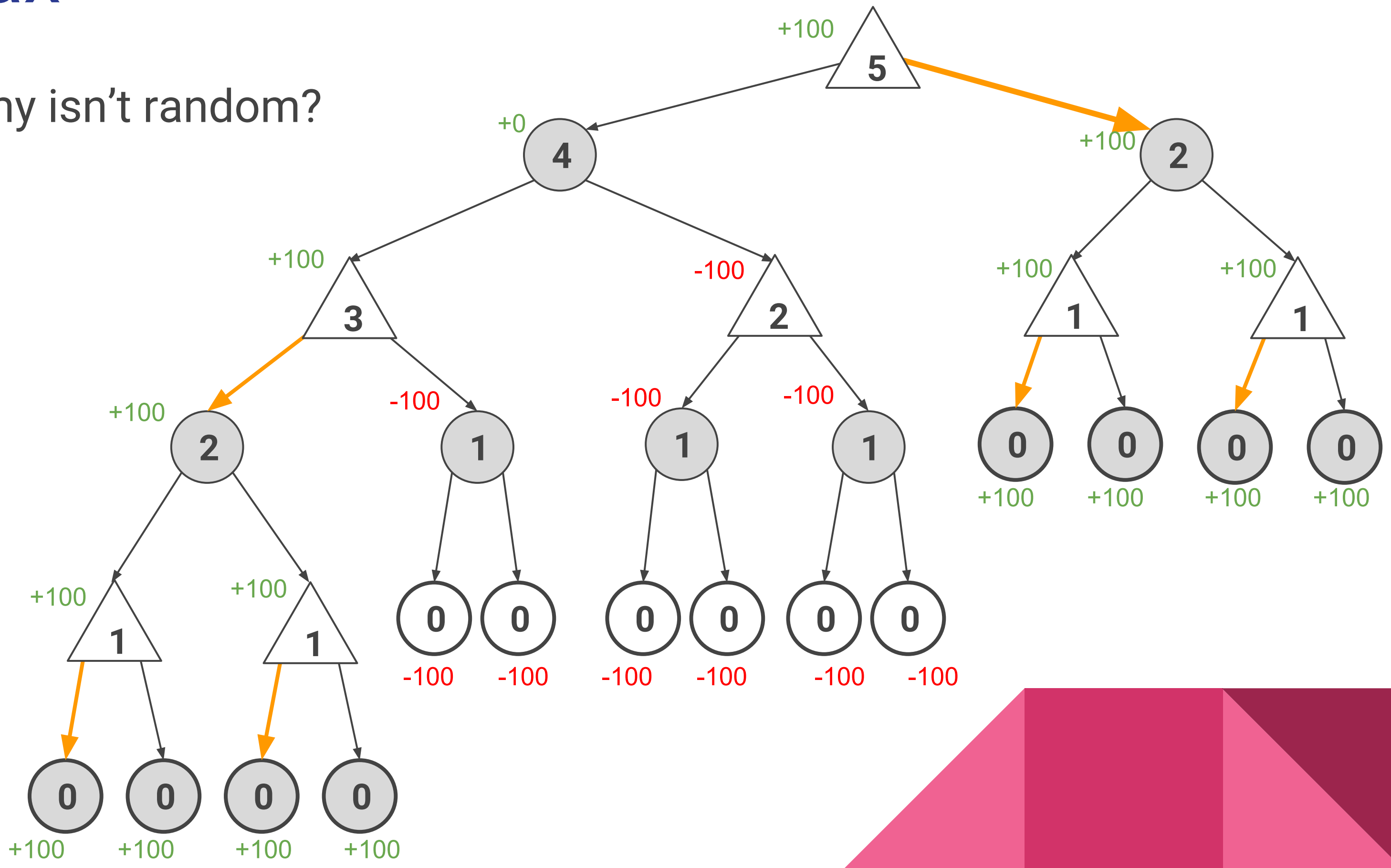
+100
5
+0
4
+100
2
+100
3
-100
2
+100
1
+100
1
+100
2
-100
1
-100
1
-100
1
0
0
0
0
+100
+100
+100
+100
+100
1
+100
1
0
0
0
0
0
0
-100
-100
-100
-100
-100
-100
0
0
0
0
+100
+100
+100
+100

Expectimax


+100
5
+0
4
+100
2
+100
3
-100
2
+100
1
+100
1
+100
2
-100
1
-100
1
-100
1
+100
1
+100
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
+100
+100
+100
+100
-100
-100
-100
-100
-100
-100
+100
+100
+100
+100

Expectimax

What if our enemy isn't random?



Minimax

- We know we want to **maximize our utility.**

$$V(s) = \max_{a \in Actions} V(Succ(s,a))$$

= Take the action **a** that maximizes the utility of the resulting state.

Minimax

- We know we want to **maximize our utility.**

$$V(s) = \max_{a \in Actions} V(Succ(s, a))$$

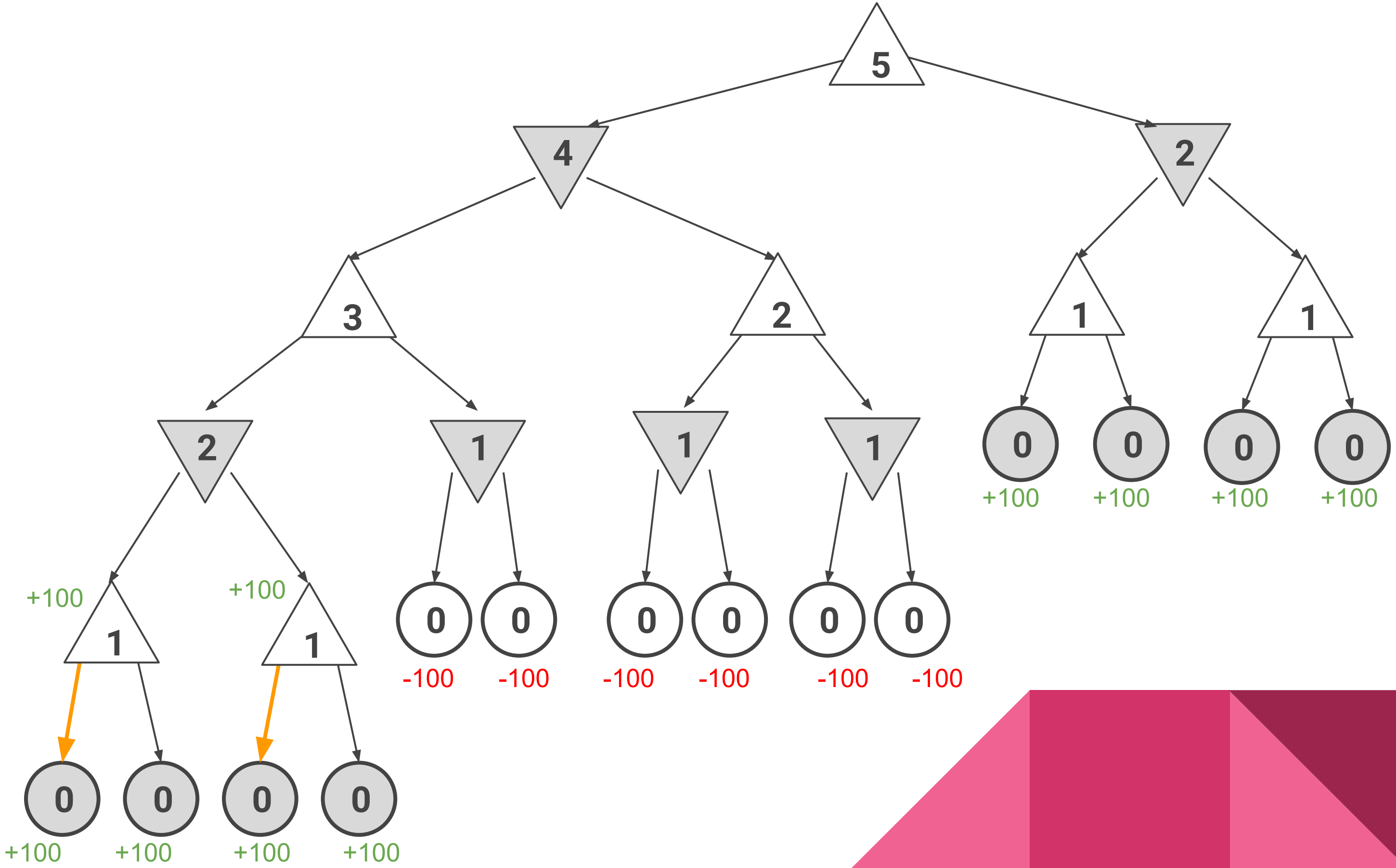
= Take the action **a** that maximizes the utility of the resulting state.

- Let's assume the enemy is adversarial, i.e. wants to **minimize our utility.**

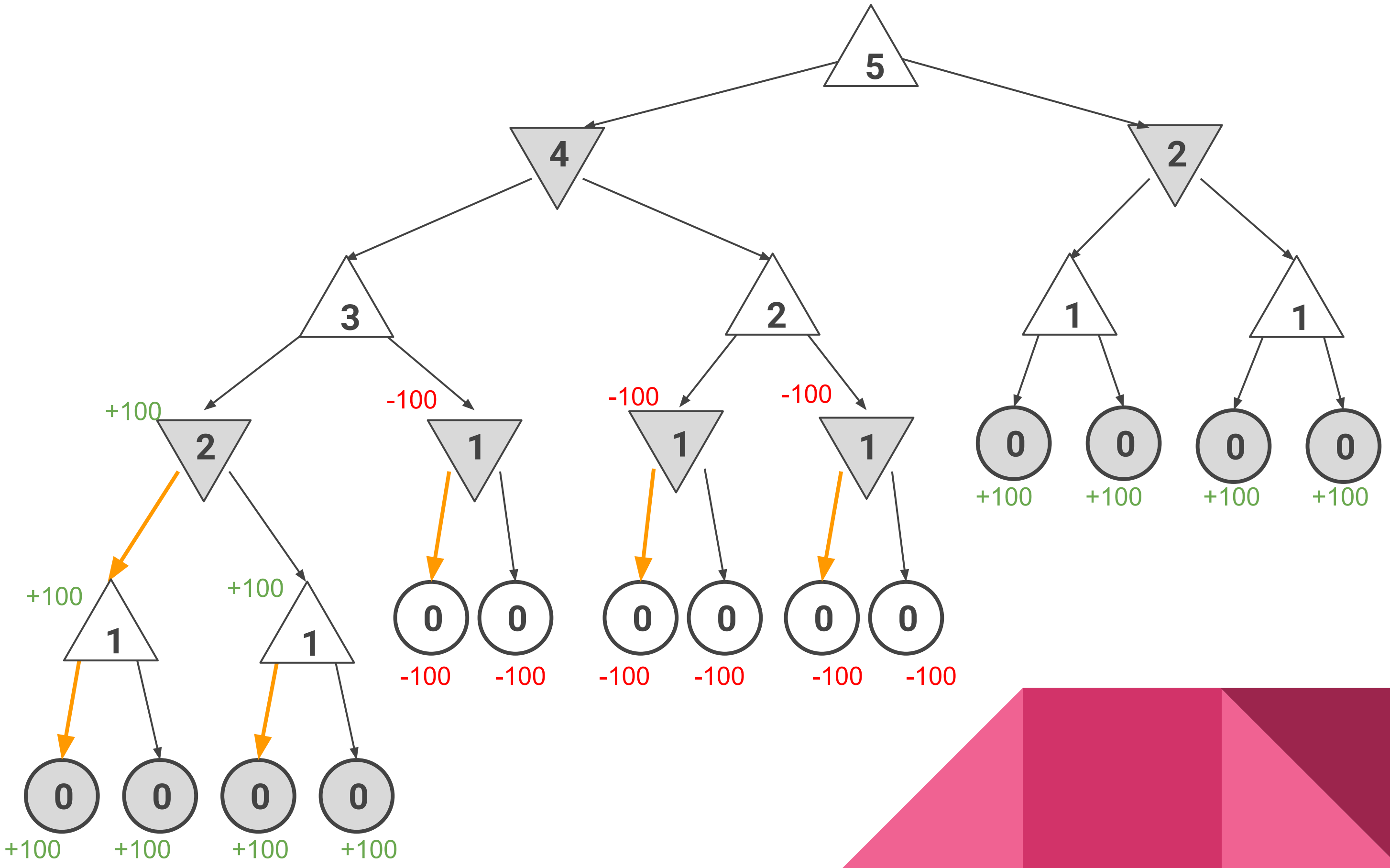
$$V(s) = \min_{a \in Actions} V(Succ(s, a))$$

= Take the action **a** that minimizes the utility of the resulting state.

Minimax

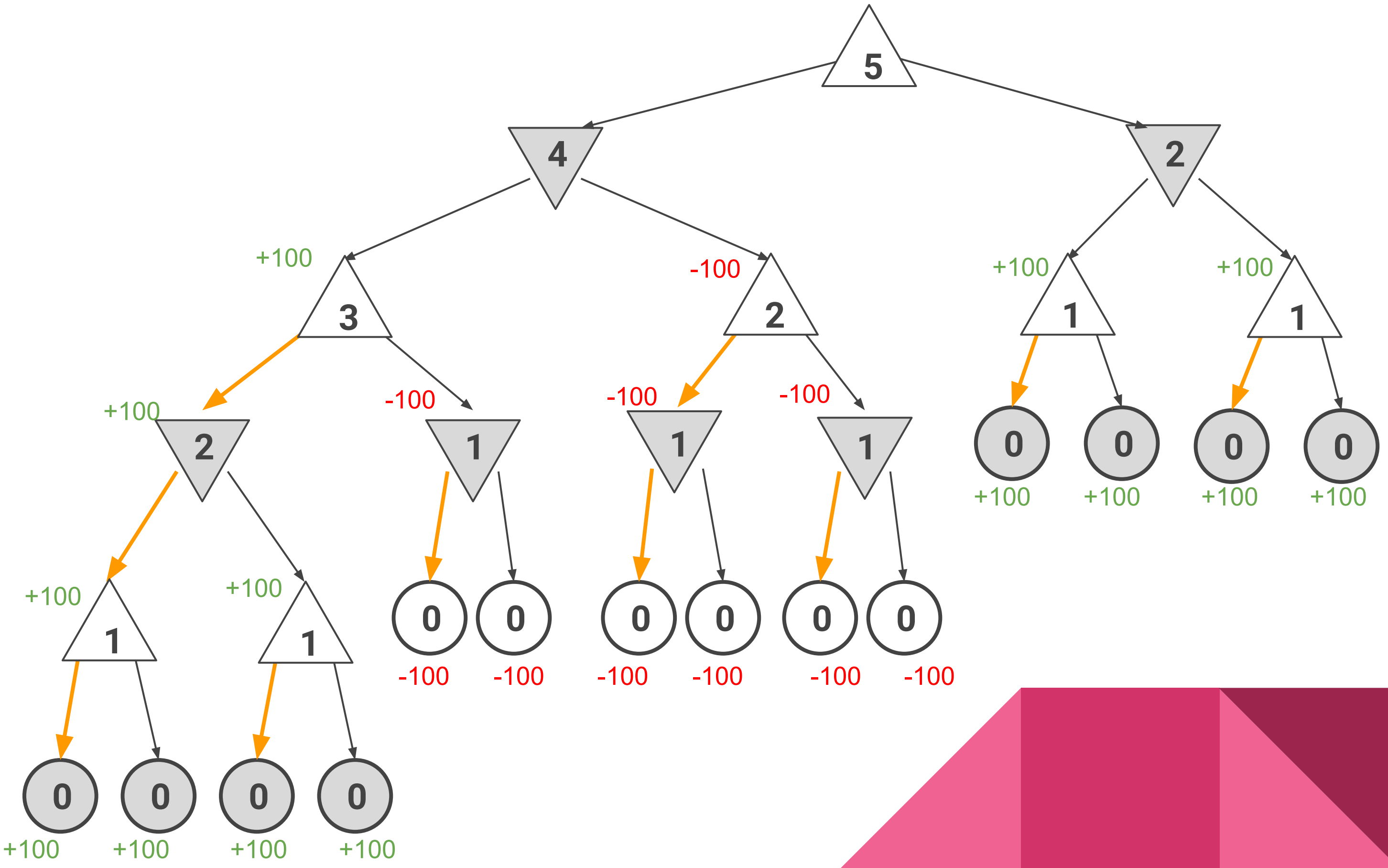

5
4
2
3
2
1
1
2
1
1
1
0
0
0
0
+100
+100
+100
+100
+100
+100
1
1
0
0
0
0
0
0
-100
-100
-100
-100
-100
-100
0
0
0
0
+100
+100
+100
+100

Minimax

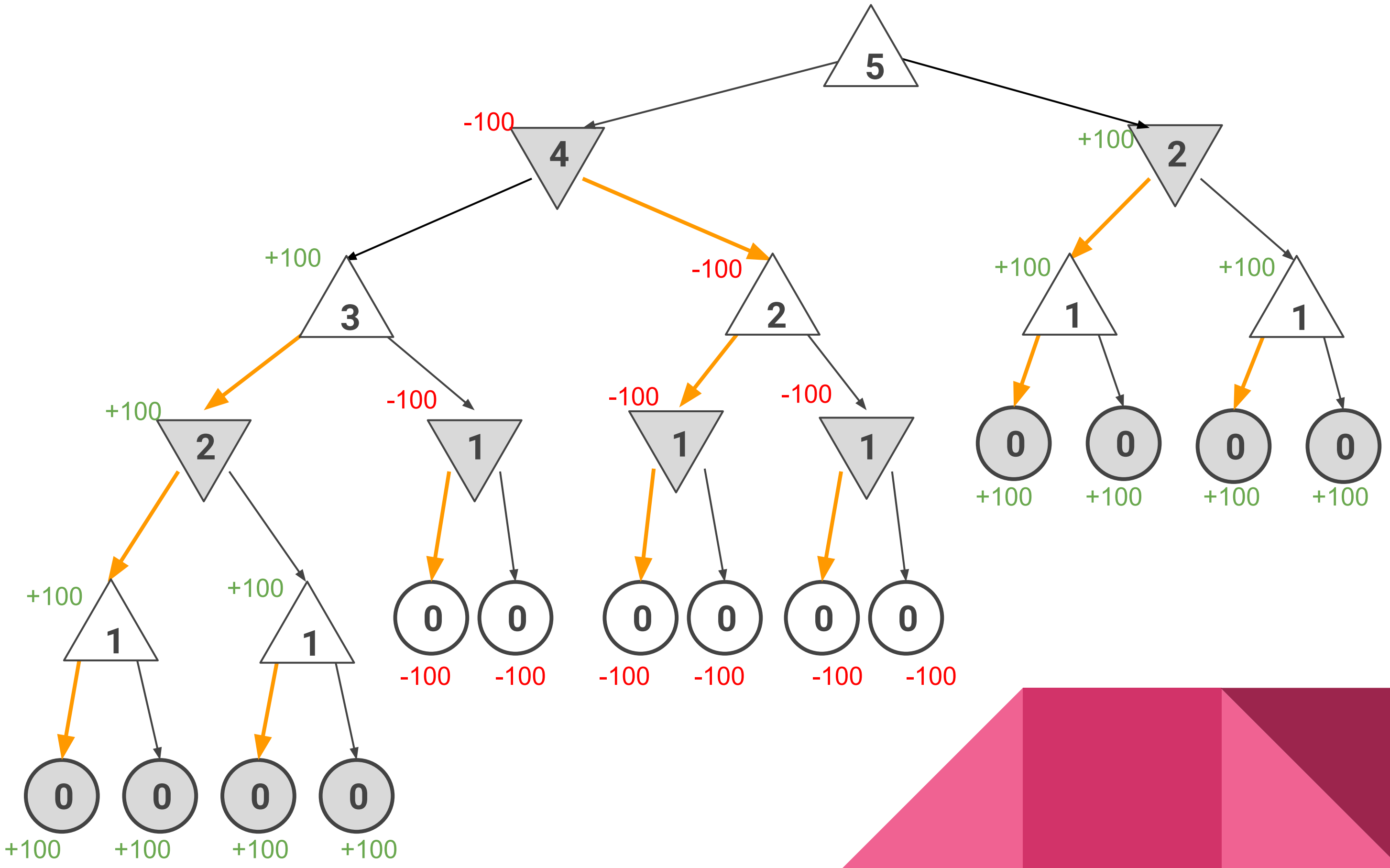


5
4
2
3
2
1
1
+100
2
-100
1
-100
1
-100
1
0
0
0
0
+100
+100
+100
+100
+100
1
+100
1
0
0
-100
-100
0
0
-100
-100
0
0
-100
-100
0
0
0
0
+100
+100
+100
+100

Minimax

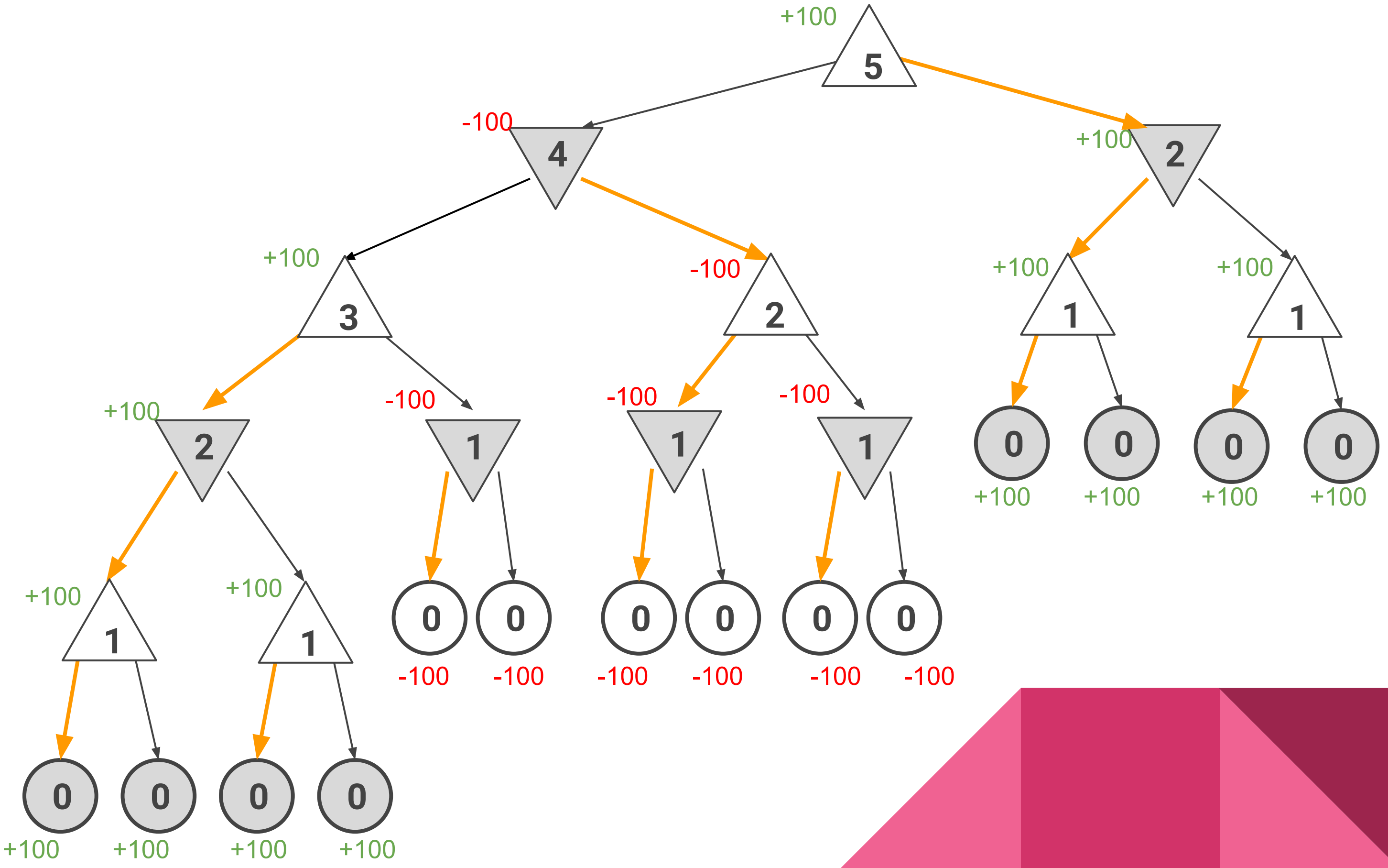

5
4
2
+100
3
-100
2
+100
1
+100
1
+100
2
-100
1
-100
1
-100
1
+100
1
+100
1
0
+100
+100
+100
+100
-100
-100
-100
-100
-100
-100
+100
+100
+100
+100

Minimax



5
-100
4
+100
2
+100
3
-100
2
+100
1
+100
1
+100
2
-100
1
-100
1
-100
1
+100
1
+100
1
0
0
0
0
+100
+100
+100
+100
0
0
0
0
0
0
-100
-100
-100
-100
-100
-100
0
0
0
0
+100
+100
+100
+100

Minimax


+100
5
-100
4
+100
2
+100
3
-100
2
+100
1
+100
1
+100
2
-100
1
-100
1
-100
1
0
0
0
0
+100
+100
+100
+100
+100
1
+100
1
0
0
0
0
0
0
-100
-100
-100
-100
-100
-100
0
0
0
0
+100
+100
+100
+100

Minimax

- DeepBlue did not use vanilla MiniMax.
    - What’s wrong?

Minimax

- DeepBlue did not use vanilla MiniMax.
  - What’s wrong?
- **Game trees are huge!!!**
- Can we do better?

Minimax

- DeepBlue did not use vanilla MiniMax.
  - What’s wrong?
- **Game trees are huge!!!**

- Can we do better?
  - Idea: Prune the search space!

Alpha-Beta pruning

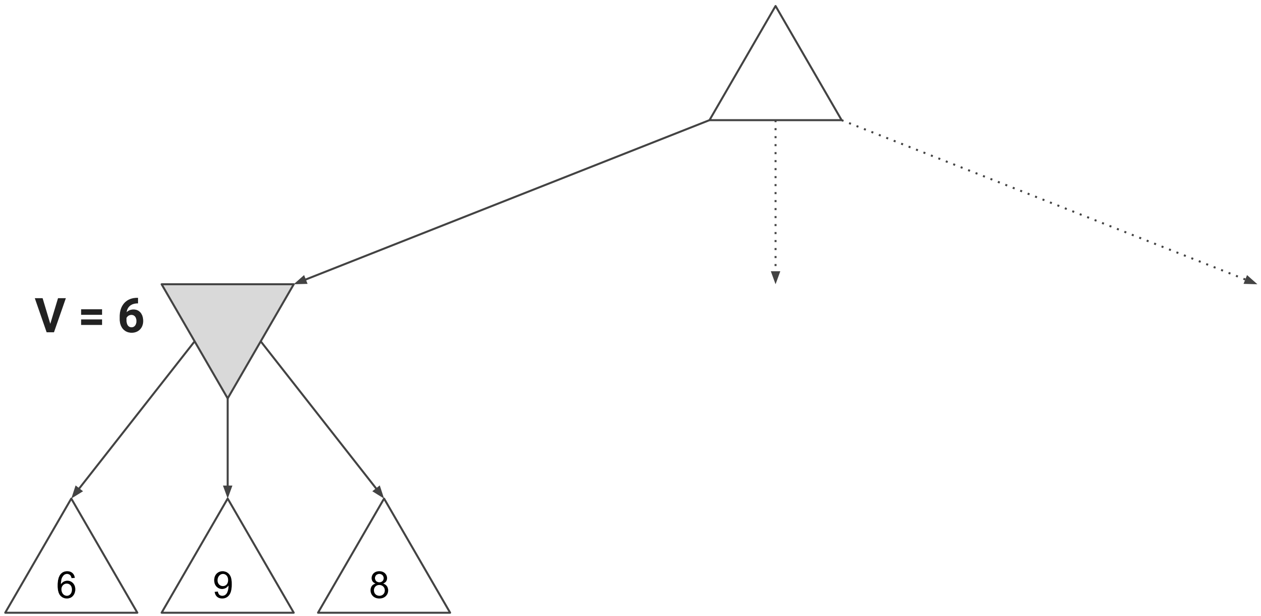
- From a max-node (our perspective):
  - If we know utility of action **a** is really high, we shouldn't have to evaluate other actions that we know will not be as good
- Inverse is true from a min-node (adversary's perspective)

Alpha-Beta pruning

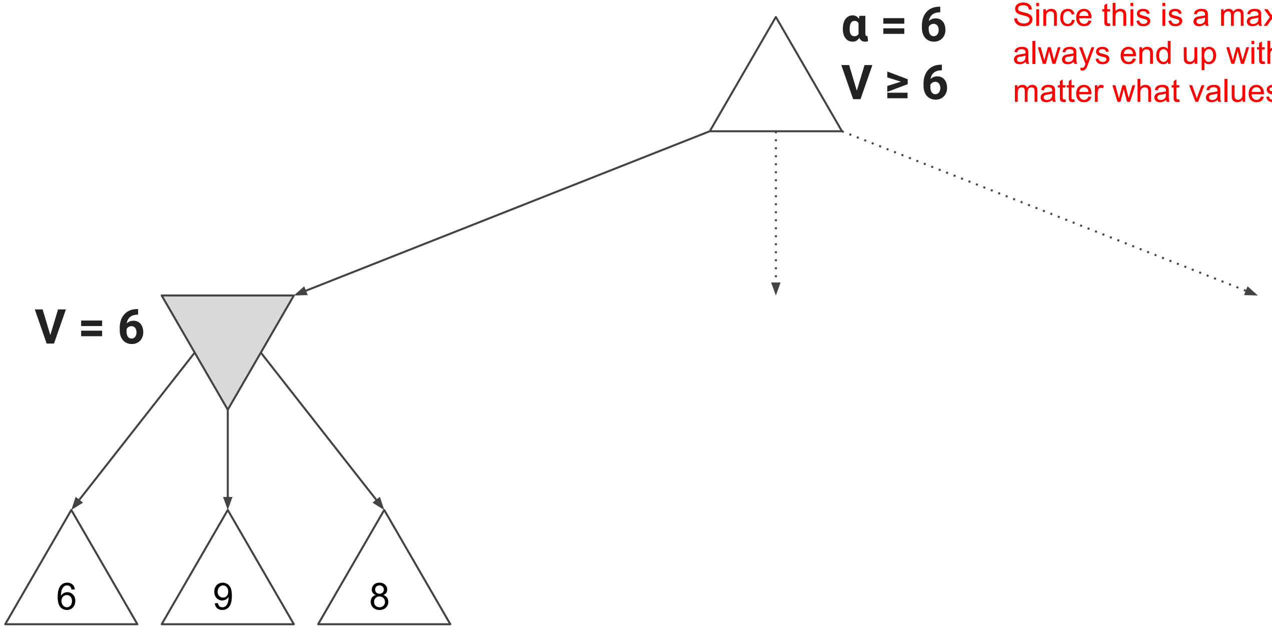
- From a max-node (our perspective):
  - If we know utility of action **a** is really high, we shouldn't have to evaluate other actions that we know will not be as good
- Inverse is true from a min-node (adversary's perspective)

- Alpha: lower bound on the value that a max-node may ultimately be assigned
  - $v >= \alpha$
- Beta: upper bound on the value that a min-node may ultimately be assigned
  - $v <= \beta$

Alpha-Beta pruning

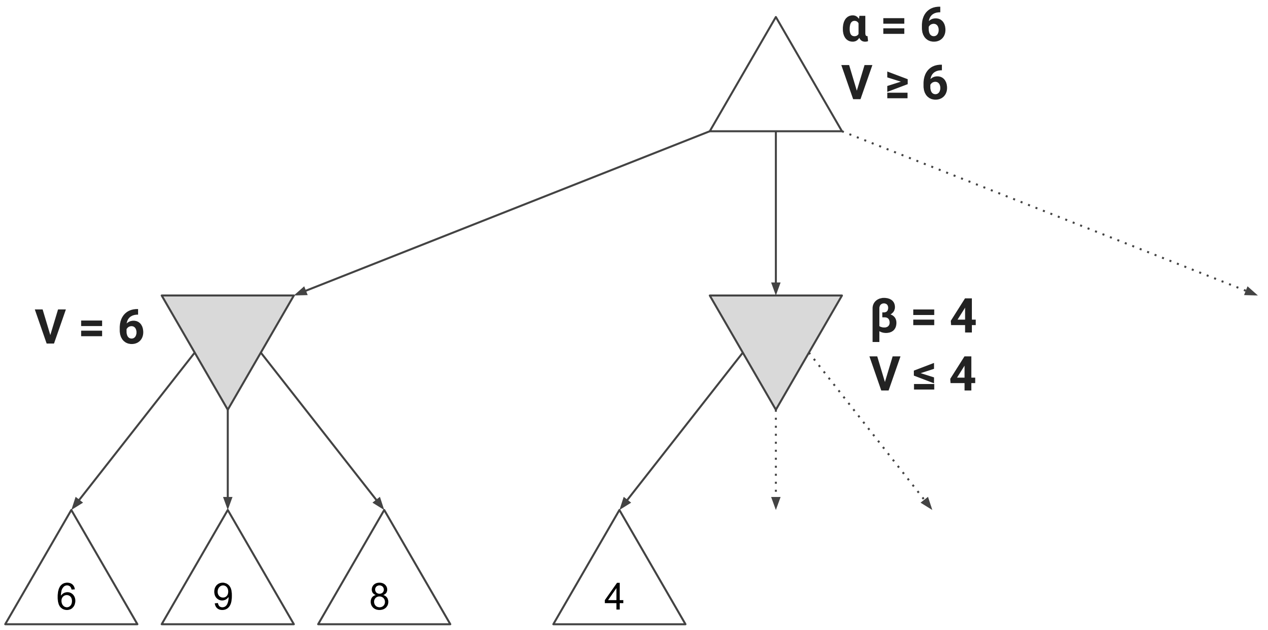


Alpha-Beta pruning

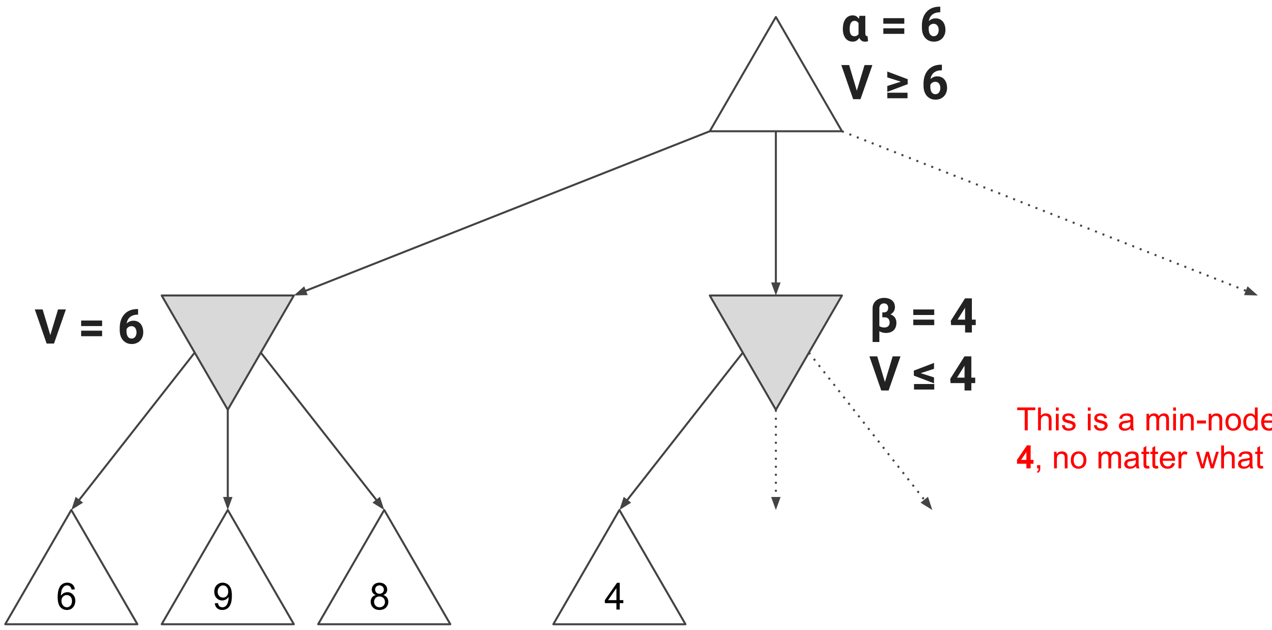


Since this is a max-node, the root node will always end up with a value of **at least 6**, no matter what values the children have

Alpha-Beta pruning

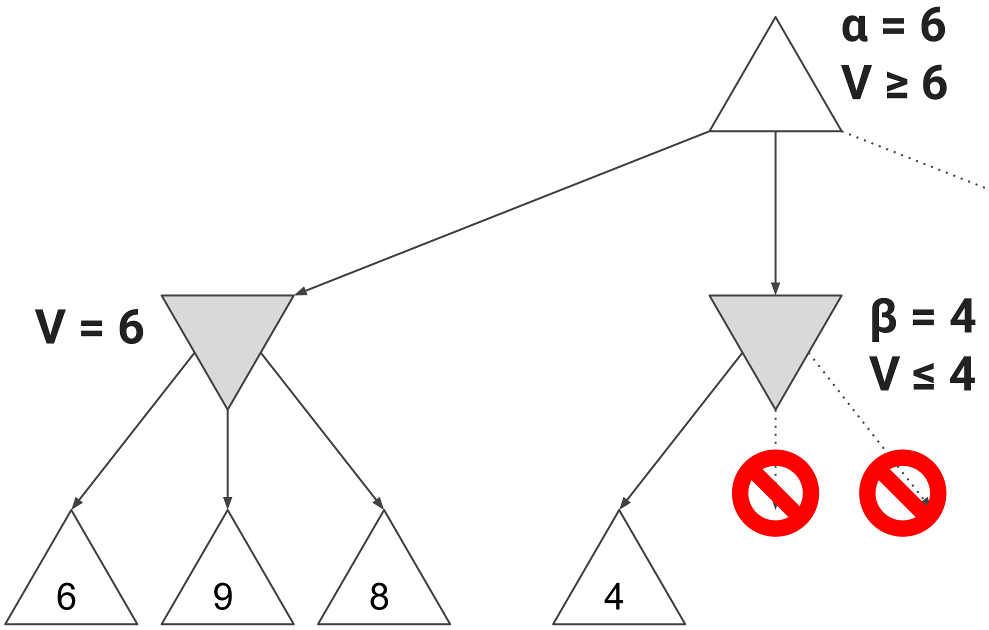


Alpha-Beta pruning



This is a min-node, so its value will be **at most 4**, no matter what values the children have.

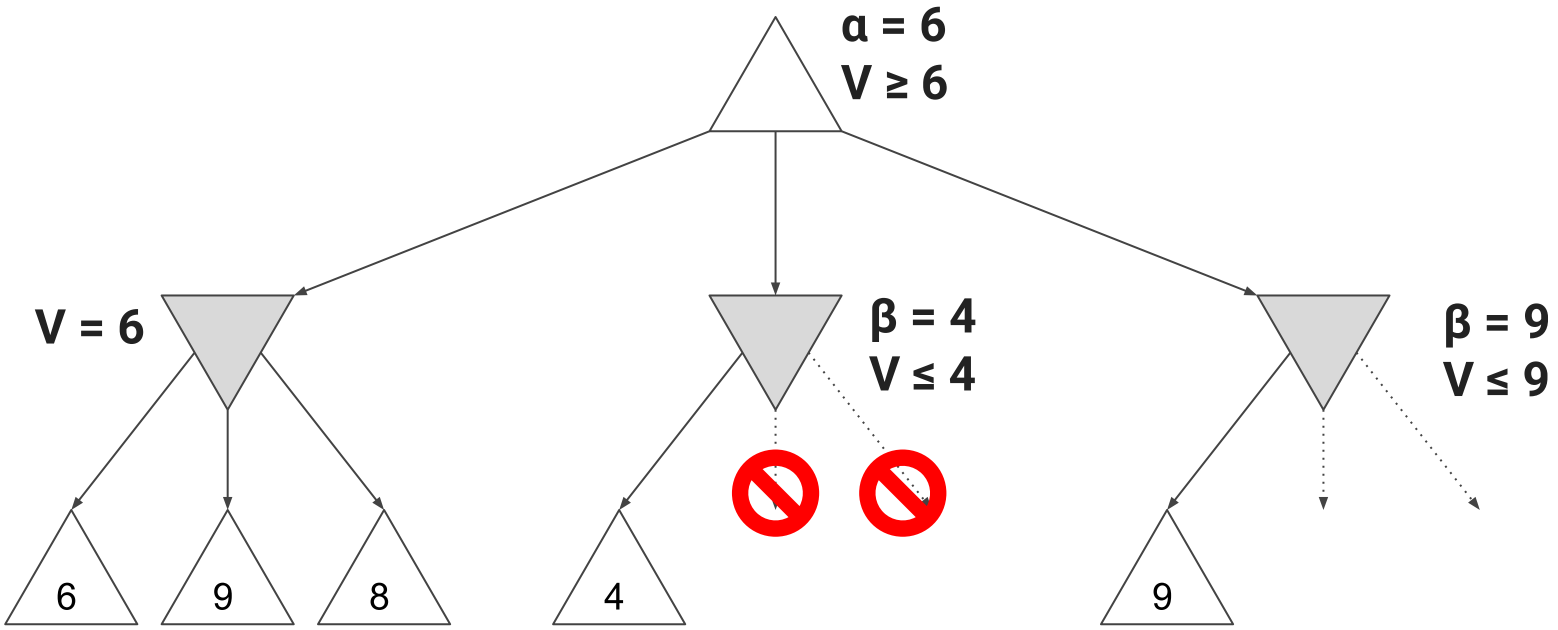
Alpha-Beta pruning



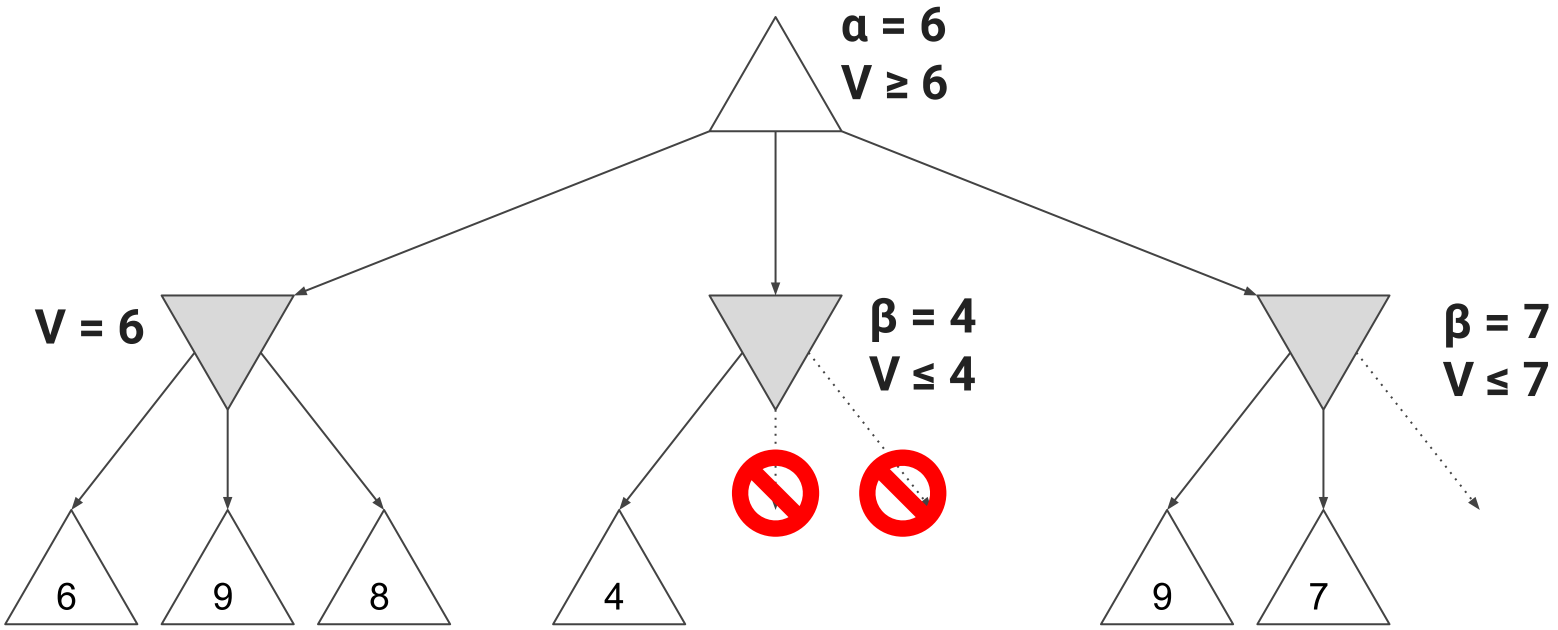
This is a min-node, so its value will be **at most 4**, no matter what values the children have.

4 < 6, so the value of this node is guaranteed to have no affect on the value of root node.

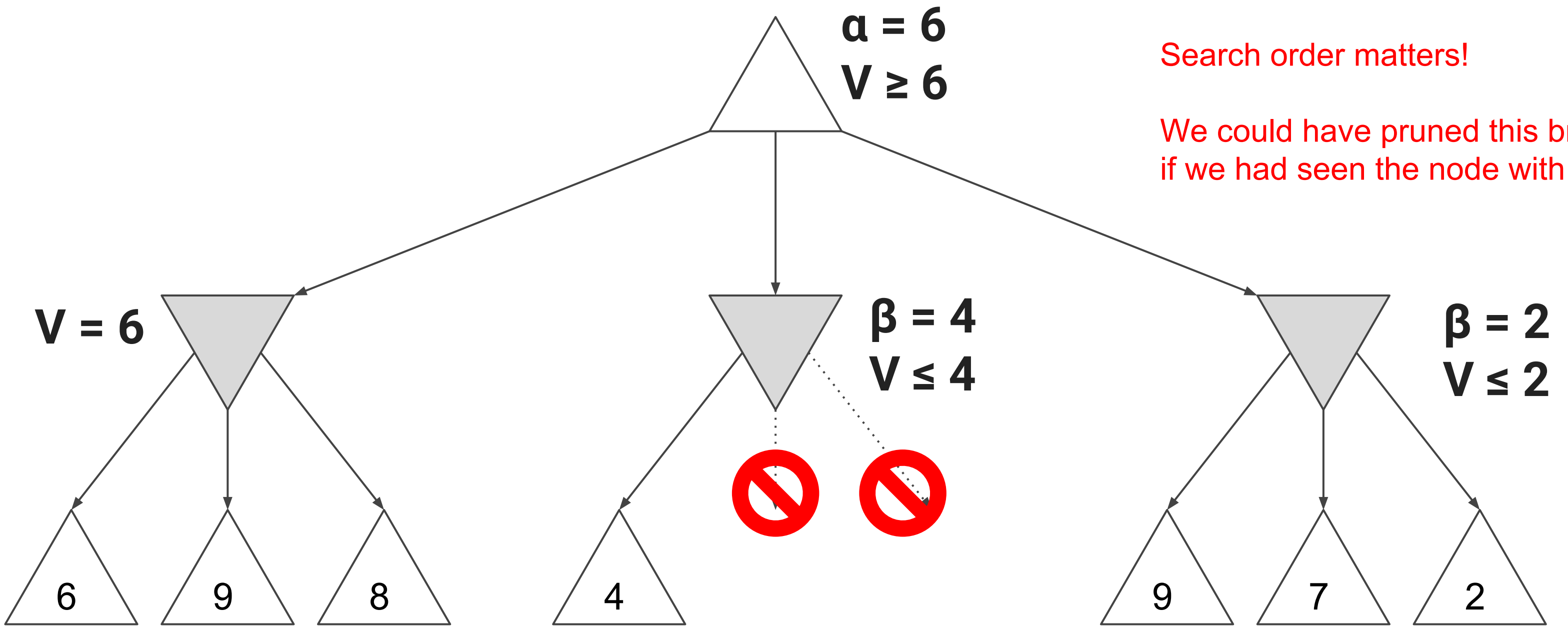
Alpha-Beta pruning



Alpha-Beta pruning



Alpha-Beta pruning



Search order matters!

We could have pruned this branch sooner if we had seen the node with value 2 first.

Alpha-Beta pruning

- DeepBlue did not just use MiniMax + Alpha-Beta pruning.
  - What’s wrong?

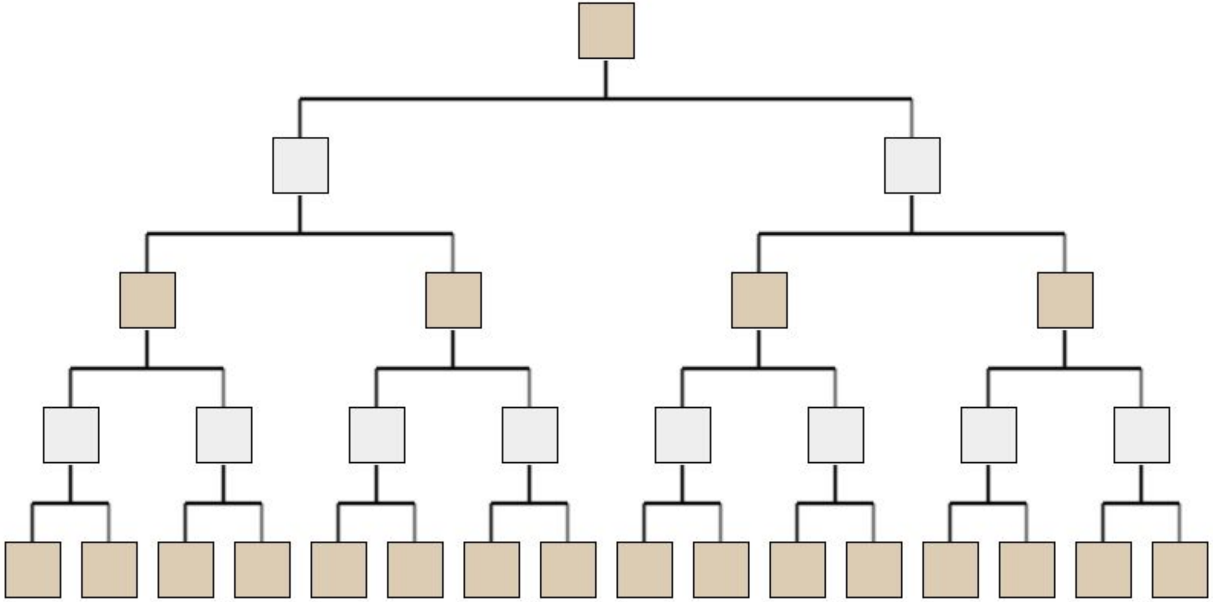
Alpha-Beta pruning

- Pretty cool, but DeepBlue did not just use MiniMax + Alpha-Beta pruning.
  - What’s wrong?
- **Game trees are too deep!!!**

- Can we do better?
  - Idea: Instead of playing the entire game, let’s guess how we’ll we’re doing after **d** moves.

Evaluation functions

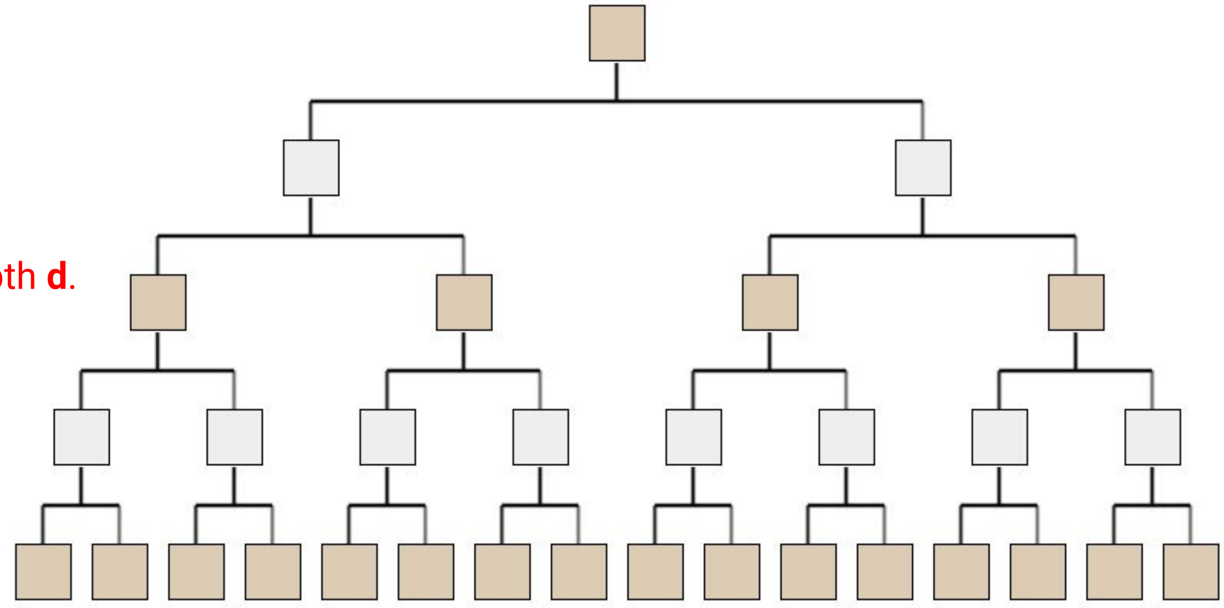
Suppose we have finite computing resources and can't afford to compute this entire tree.



Evaluation functions

Suppose we have finite computing resources and can't afford to compute this entire tree.

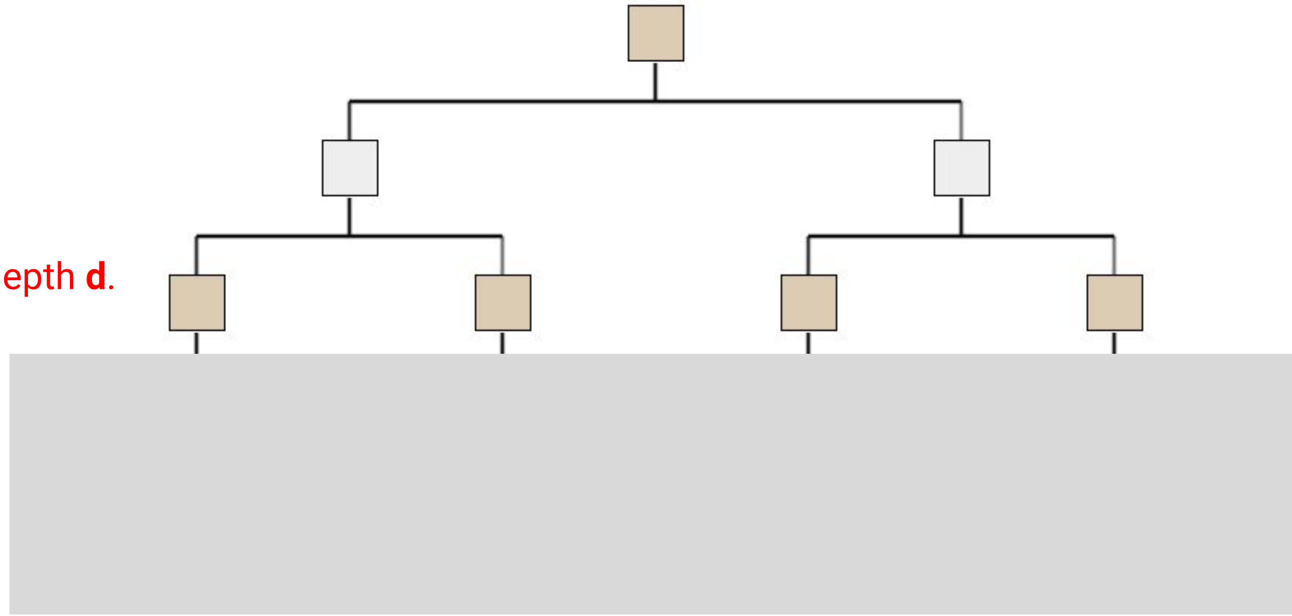
Let's stop our search at some fixed depth **d**.



Evaluation functions

Suppose we have finite computing resources and can't afford to compute this entire tree.

Let's stop our search at some fixed depth **d**.

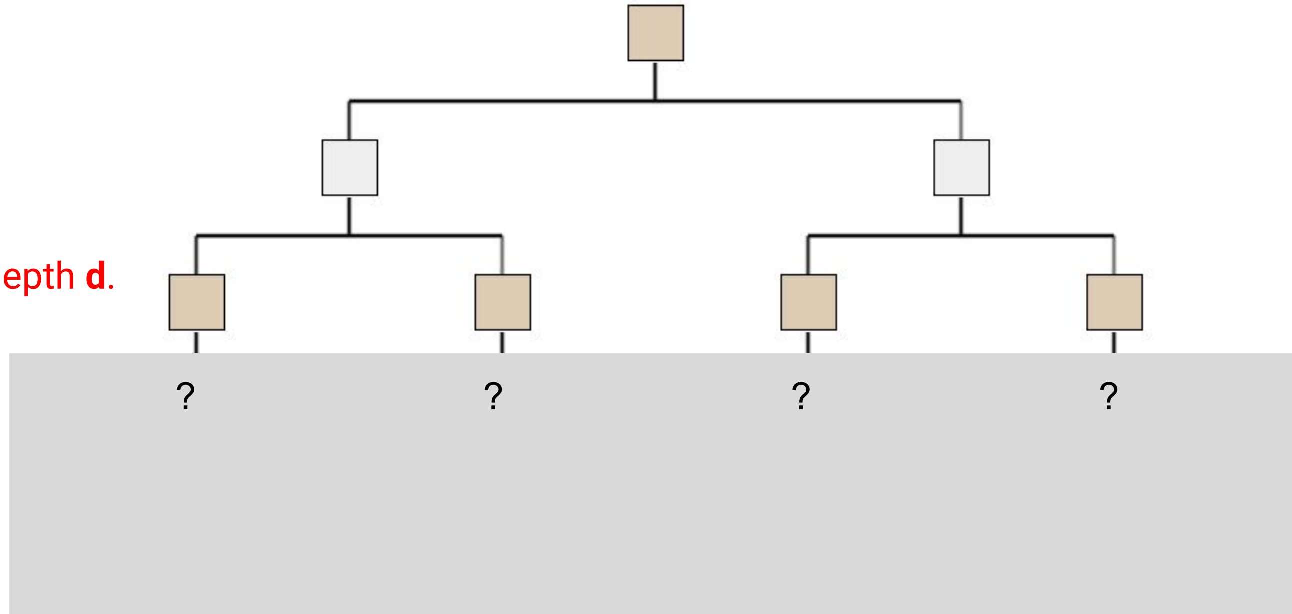


Evaluation functions

Suppose we have finite computing resources and can't afford to compute this entire tree.

Let's stop our search at some fixed depth **d**.

How do we know the utility of these new leaf nodes (to propagate up the game tree)?



Evaluation functions

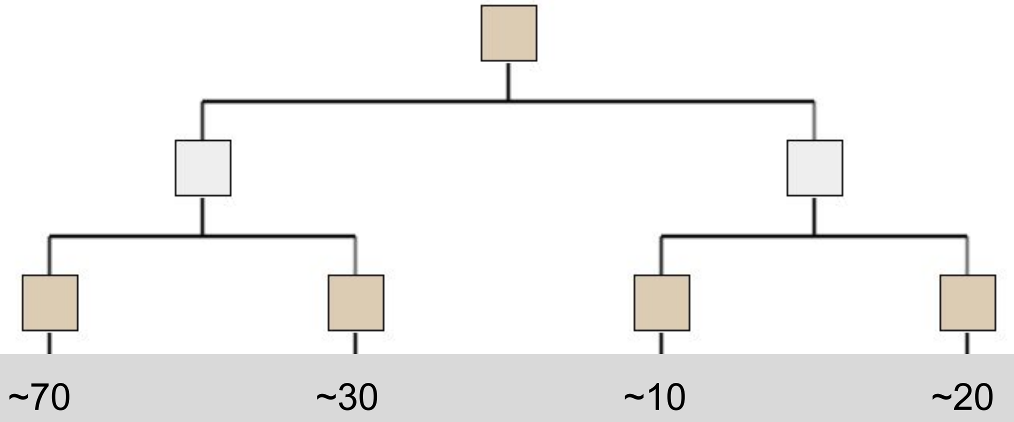
Suppose we have finite computing resources and can't afford to compute this entire tree.

Let's stop our search at some fixed depth **d**.

How do we know the utility of these new leaf nodes (to propagate up the game tree)?

Guess! (use an **heuristic**)

From current game state, how likely am I to win?



Evaluation functions

- Connect-4:
    - How many “open” connect-3’s do I have?
    - How many “open” connect-2’s do I have?
- Chess (DeepBlue): “material, position, King safety and tempo”
    - Material: How many pieces do I have left? And what are they worth?
    - Position: How many empty/safe squares can I attack?
    - King safety: How in-danger of attack is my King?
    - Tempo: Have I been making progress recently?

- DeepBlue: MiniMax tree + Alpha-Beta pruning to a depth of ~13.
    - After that depth, used evaluation function to estimate utility.

Go

- Why wasn't DeepBlue's algorithm good for Go?

Go

- Why wasn't DeepBlue's algorithm good for Go?

- Go is way harder than chess.
  - ~300 possible actions for every game board (vs ~30 in chess)
  - ~150 moves per game (vs ~70 in chess)
  - Total number of possible games
    - ~10^761 (vs ~10^120) for chess
    - There's only 10^80 atoms in the universe?

Alpha Go’s Approach

- Monte Carlo Tree Search
- “Value network” as evaluation function
  - What’s the expected utility of this board state?
- “Policy network” as selection function
  - What moves are more likely to happen from this state?
- Fed data from seeing many expert games

Monte Carlo Tree Search

- I have limited resources to find the optimal policy for every game state.

Monte Carlo Tree Search

- I have limited resources to ~~find~~ the optimal policy for every game state.

approximate

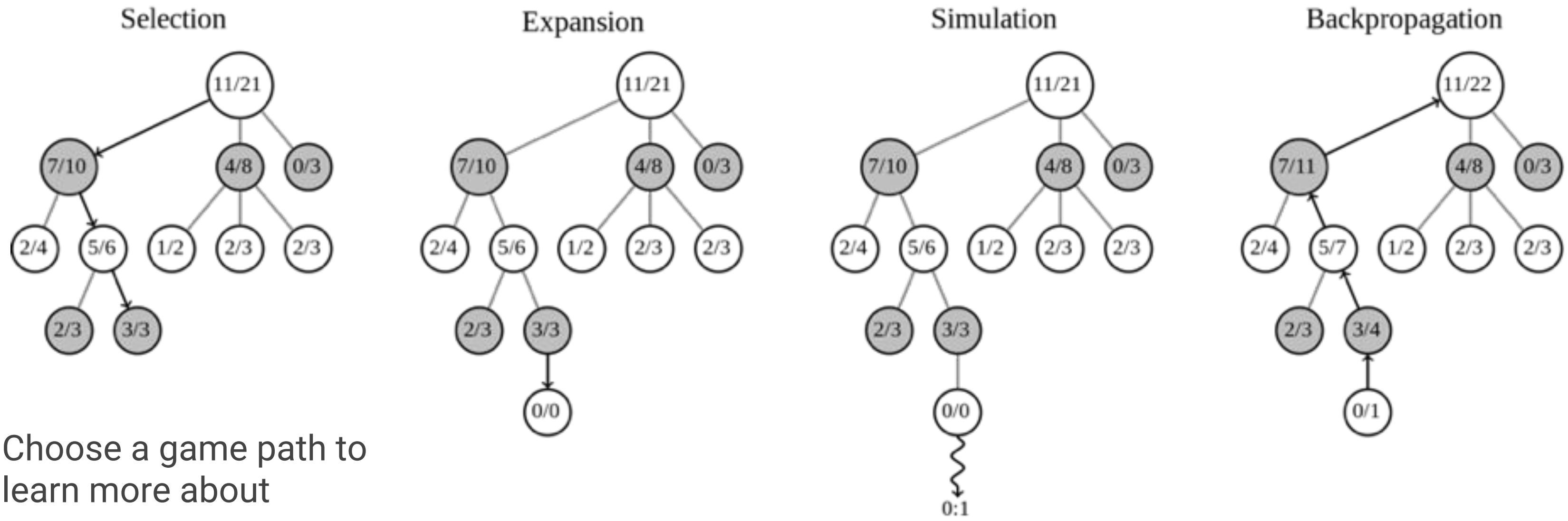
Monte Carlo Tree Search

- I have limited resources to ~~find~~ the optimal policy for ~~every game state~~.

approximate

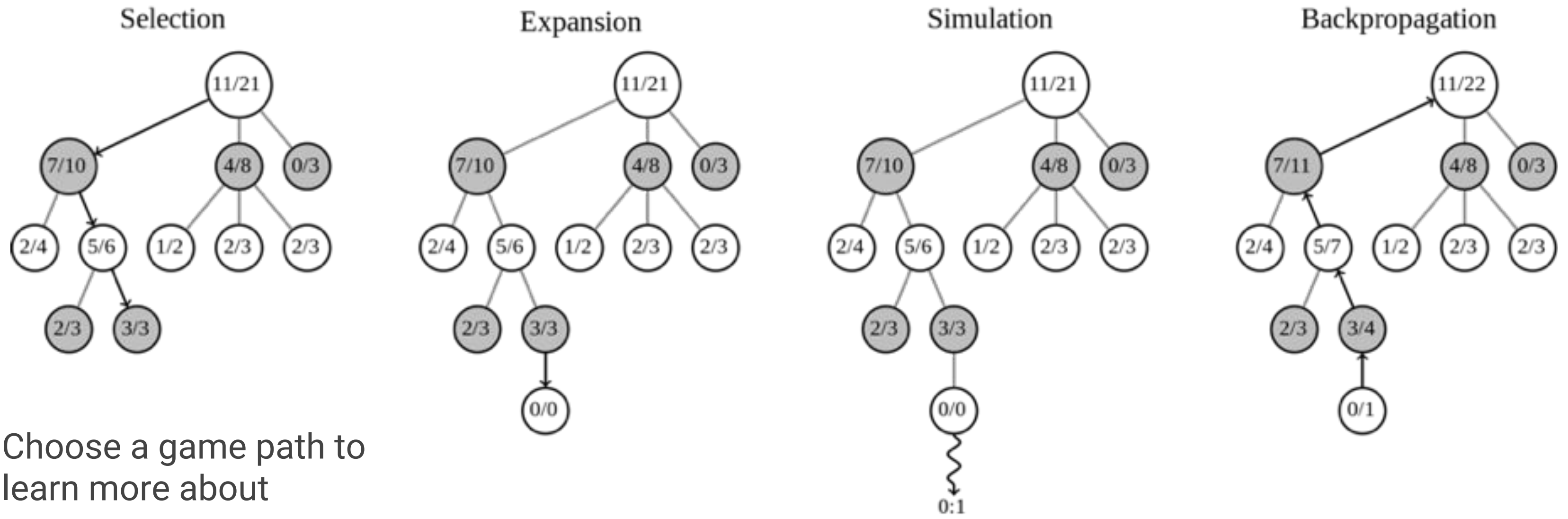
the most common
game states

Monte Carlo Tree Search: the core loop



Choose a game path to learn more about

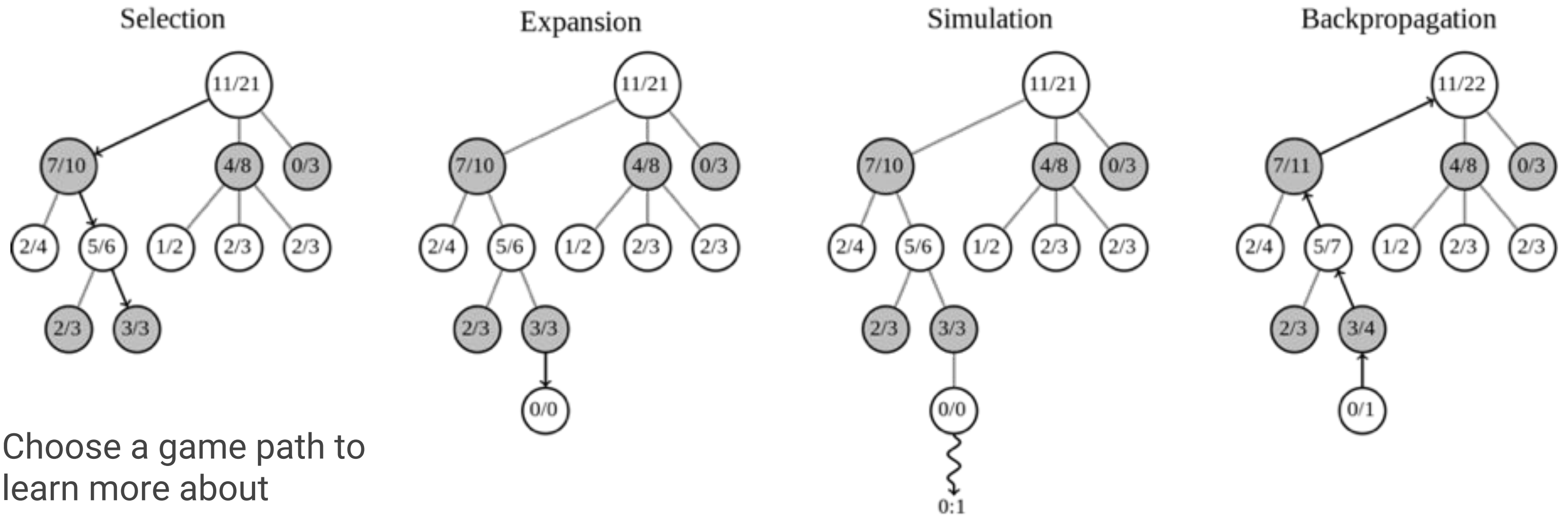
Monte Carlo Tree Search: the core loop



Choose a game path to learn more about

Add a MCTS node to our search tree

Monte Carlo Tree Search: the core loop

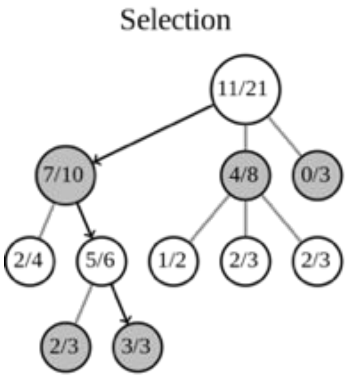


Choose a game path to learn more about

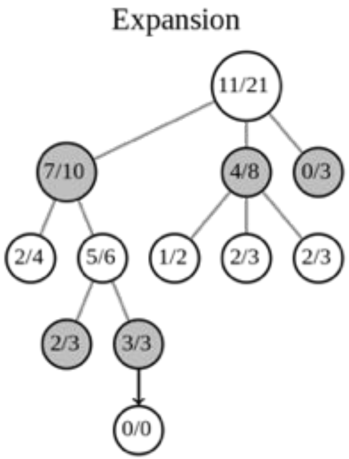
Add a MCTS node to our search tree

Play a game randomly: Did we win?

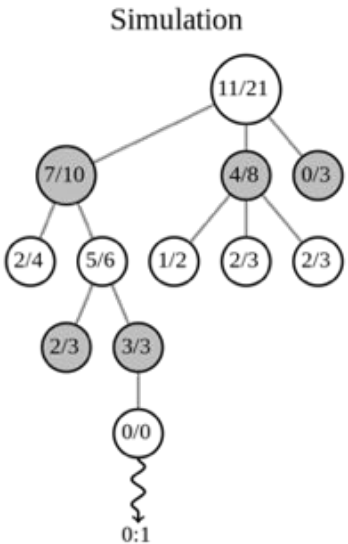
Monte Carlo Tree Search: the core loop



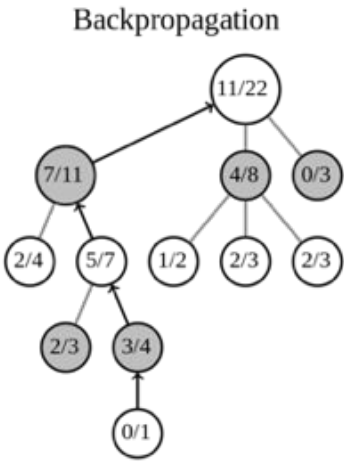
Choose a game path to learn more about



Add a MCTS node to our search tree

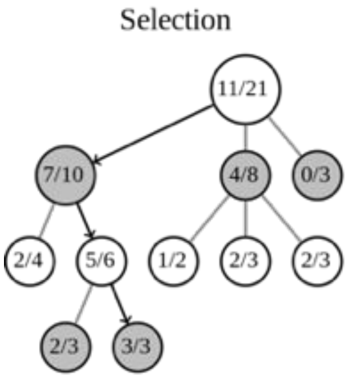


Play a game randomly: Did we win?



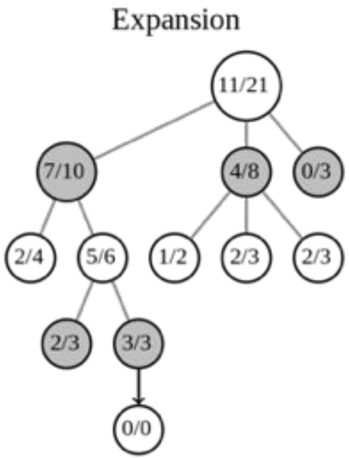
Propagate result up through path

Monte Carlo Tree Search: the core loop

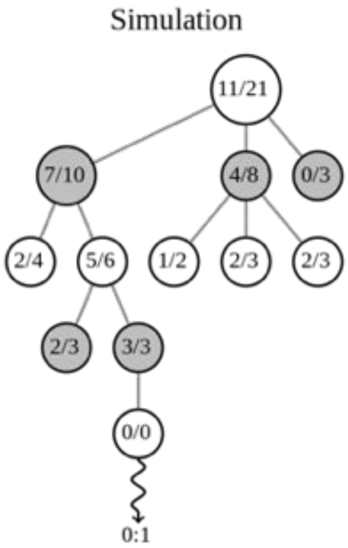


Choose a game path to learn more about

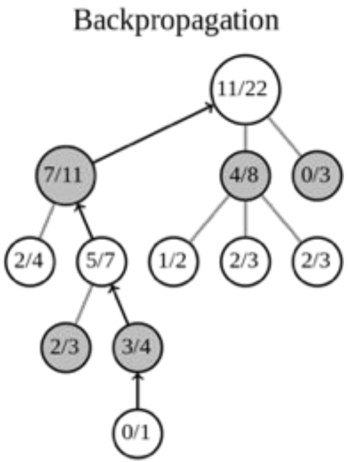
a good selection policy explores “common” game paths more often, while also exploring unknown states



Add a MCTS node to our search tree

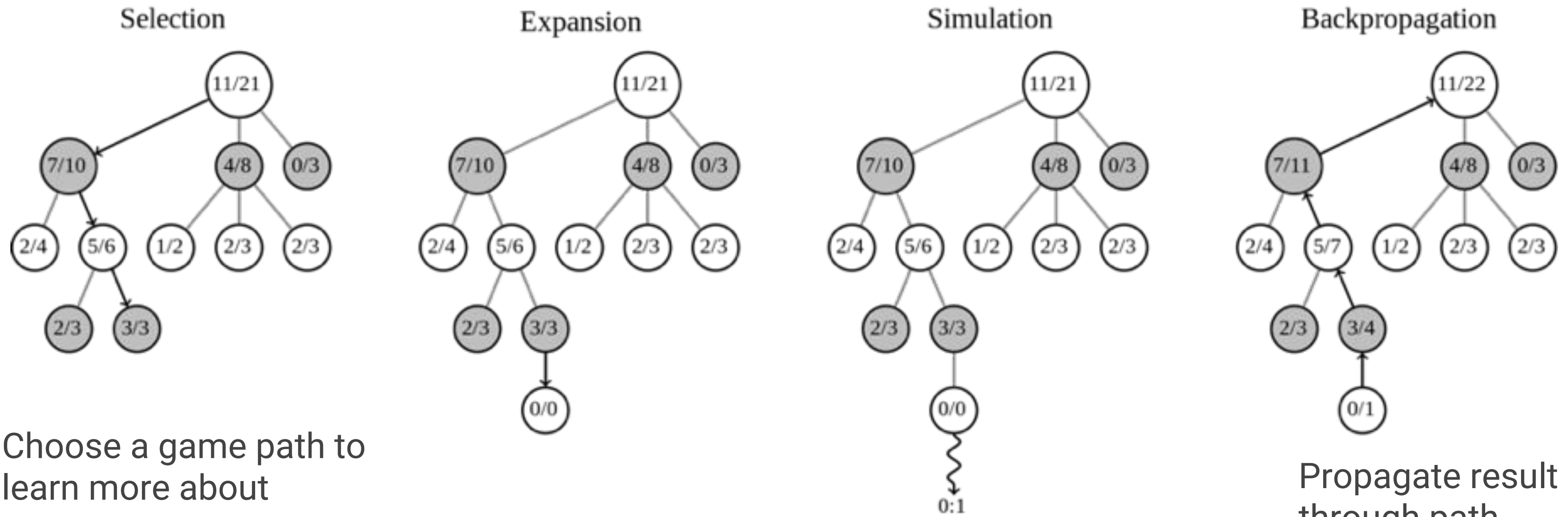


Play a game randomly: Did we win?



Propagate result up through path

Monte Carlo Tree Search: the core loop



Choose a game path to learn more about

Add a MCTS node to our search tree

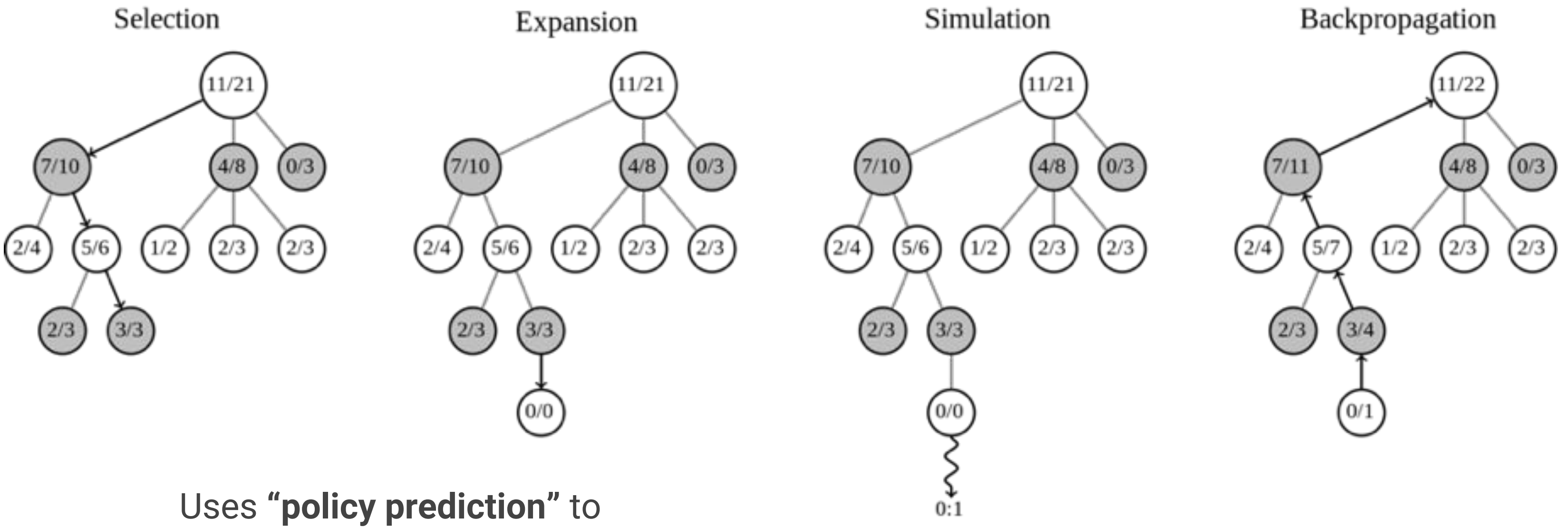
Play a game randomly: Did we win?

Propagate result up through path

a good selection policy explores "common" game paths more often, while also exploring unknown states

Instead of doing a full playout, some MCTS use an evaluation function.

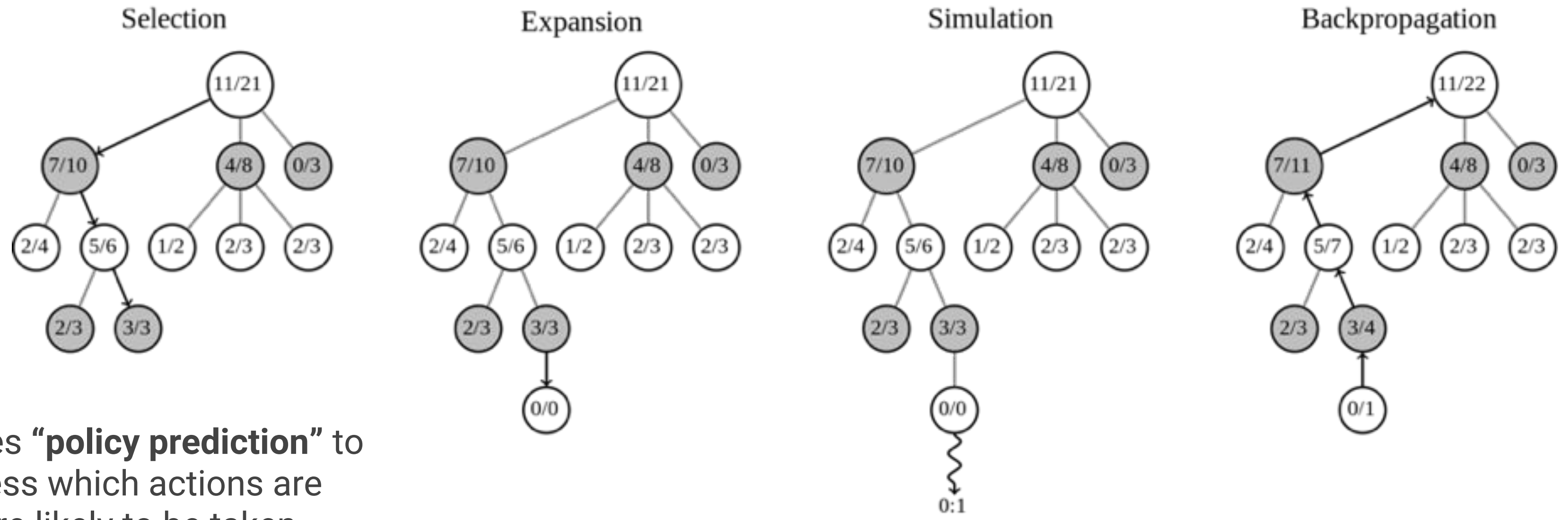
AlphaGo’s Monte Carlo Tree Search



Uses “**policy prediction**” to guess which actions are more likely to be taken.

Uses “**value prediction**” as an evaluation function instead of performing full playout.

AlphaGo’s Monte Carlo Tree Search

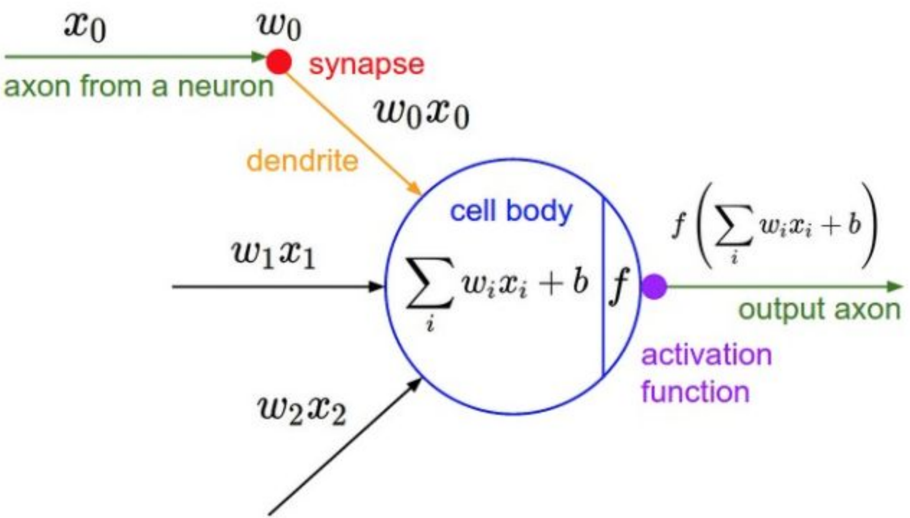


Uses “**policy prediction**” to guess which actions are more likely to be taken.

Uses “**value prediction**” as an evaluation function instead of performing full playout.

These predictions are trained using a **convolutional neural network.**

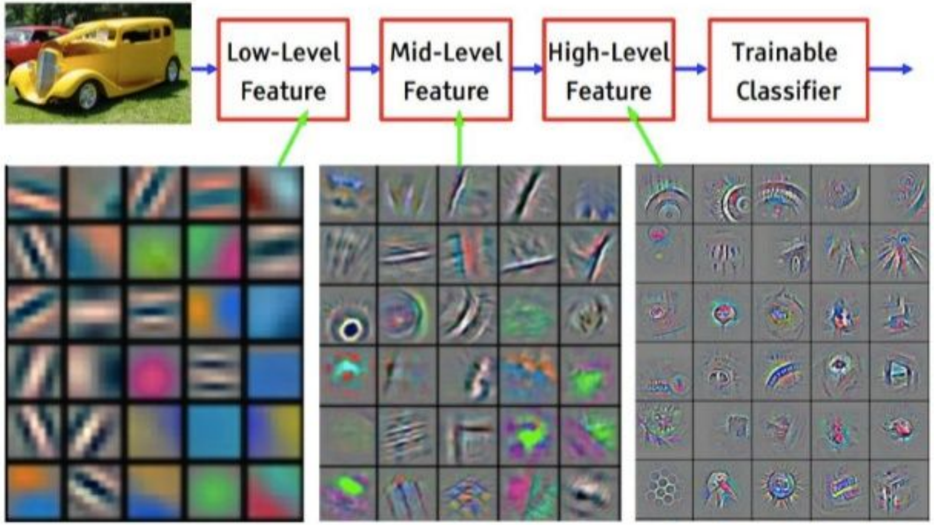
Convolutional Neural Networks



How does training work?

- Take an affine function of input (with weights)
- Pass this output through a nonlinear function -- activation function.

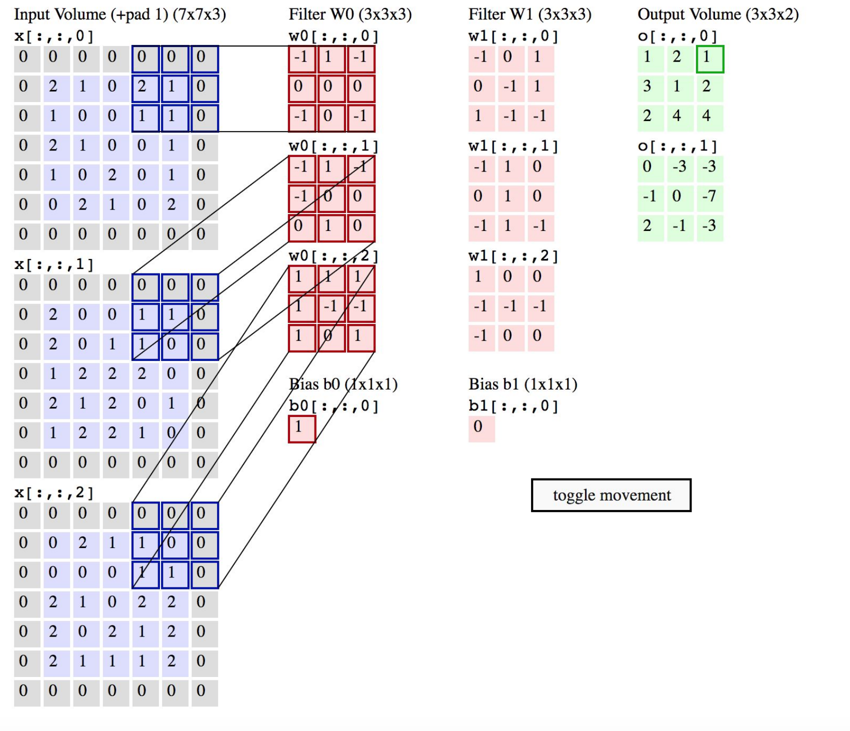
Convolutional Neural Networks



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

How do you train a classifier from these features

Convolutional Neural Networks

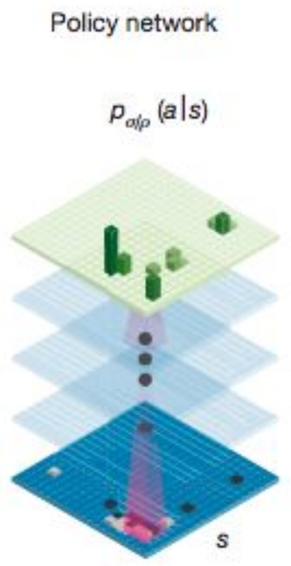


What are they doing mechanically?

- Finding local features in a picture
- Prioritizing features that help predict outcome of interest
- Value Network -> Predict Rewards
- Policy Network -> Predict Next Moves

Policy Network

- Given a 19x19 Go board, output **probability distribution over all legal moves**
- Data from 30 million positions, and data from "self-plays"
- 13 layers!



Value Network

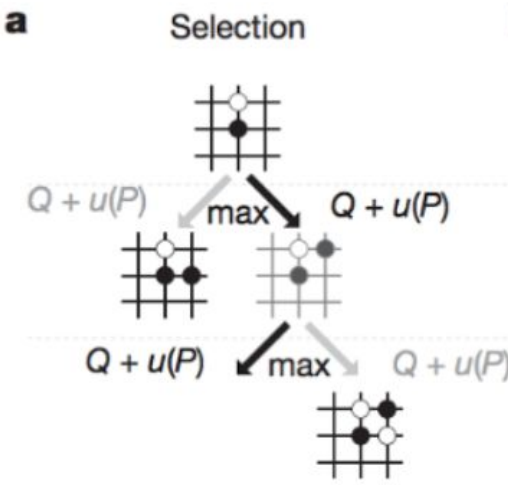
- Given a 19x19 Go board, output a **value.**
  - How likely am I to win?
- Learned on same games as policy network

MCTS in Alpha Go

Selection

We choose which path to “learn more” about by selecting paths with max “**Q + u(P)**”

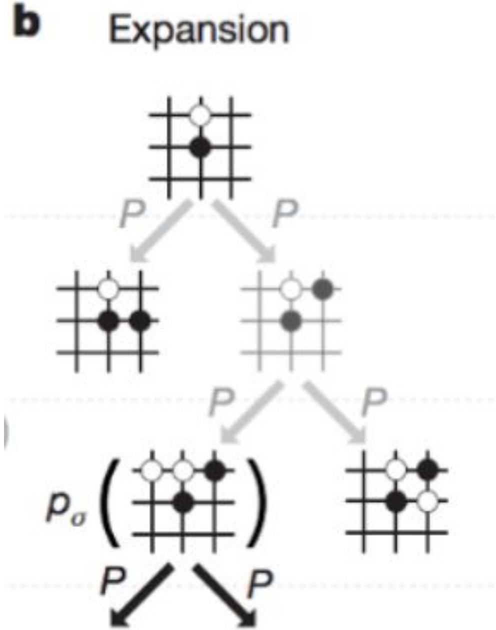
- **Q** trained by value network, **u(P)** samples probability of this action from policy network



MCTS in Alpha Go

Expansion

To choose a node to expand, randomly sample probability distribution from **policy network**.

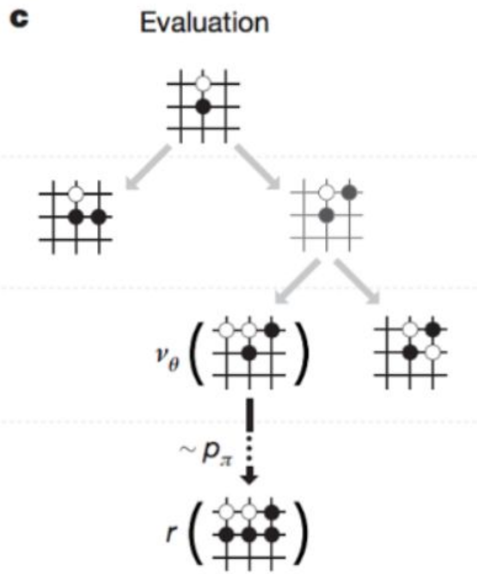


MCTS in Alpha Go

Evaluation

Heuristic is either:

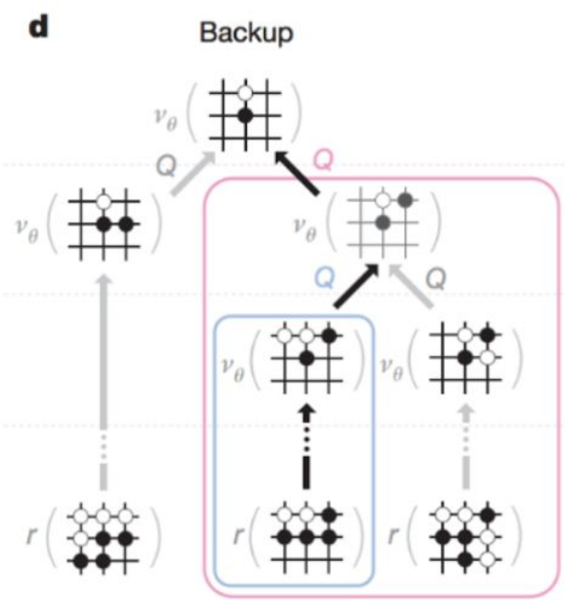
- **Q** from value network
- **r** from "fast rollout"
  - i.e. simulated game



MCTS in Alpha Go

Backpropagation

Q values in the entire path are backpropagated based on the evaluation result.



It’s not perfect

- Alpha Go’s only loss against Lee:
- White 78, Lee played an unexpected move
- AlphaGo failed to explore this in MCTS

Two possible reasons:

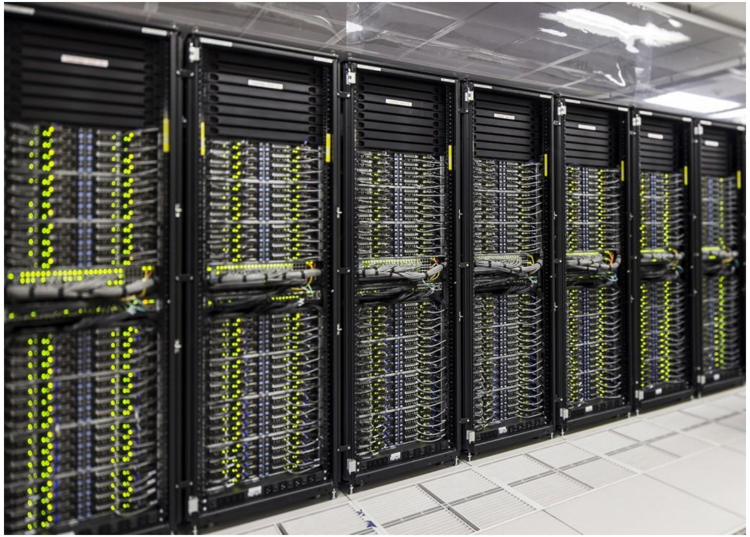
- Policy network hadn’t been trained for long enough
- Selection too aggressively chooses “common” game paths, not enough exploration

AlphaGo

- We just designed AlphaGo!
- … Almost

Computational Power

- 1202 CPUs!
- 176 GPUs!
- Specialized hardware against Lee Sedol



Summary

AlphaGo applied advanced versions of techniques in this class!

Name	ELO
Lee Sedol	3517
AlphaGo (2016)	~3594
Ke Jie (world champion)	3616