CSE 473: Artificial Intelligence
Spring 2013

Adversarial Search

Luke Zettlemoyer

Based on slides from Dan Klein

Many slides over the course adapted from either Stuart Russell or Andrew Moore

Today

- Adversarial Search
  - Minimax search
  - α-β search
  - Evaluation functions
  - Expectimax

- Reminder:
  - Programming 1 due one week from Friday!
  - Programming 2 will be on adversarial search

Game Playing State-of-the-Art

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions. Checkers is now solved!
- **Chess:** Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue examined 200 million positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Othello:** Human champions refuse to compete against computers, which are too good.
- **Go:** Human champions are beginning to be challenged by machines, though the best humans still beat the best machines. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves, along with aggressive pruning.
- **Pacman:** unknown

General Game Playing

General Intelligence in Game-Playing Agents (GIGA'13)

(http://giga13.ru.is)

General Information

Artificial Intelligence (AI) researchers have for decades worked on building game-playing agents capable of matching wits with the strongest humans in the world, resulting in several success stories for games like chess and checkers. The success of such systems has been partly due to years of relentless knowledge-engineering effort on behalf of the program developers, manually adding application-dependent knowledge to their game-playing agents. The various algorithmic enhancements used are often highly tailored towards the game at hand.

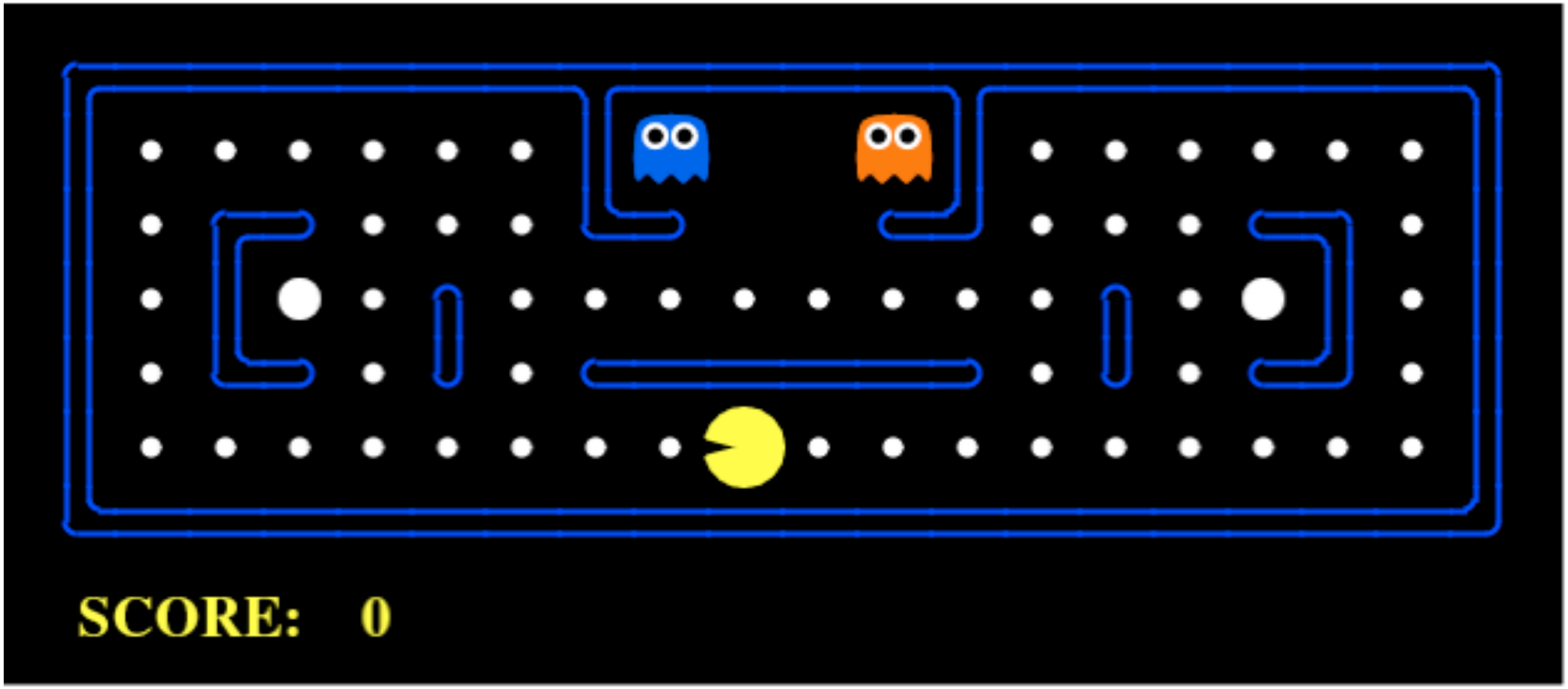
Research into general game playing (GGP) aims at taking this approach to the next level: to build intelligent software agents that can, given the rules of any game, automatically learn a strategy for playing that game at an expert level without any human intervention. In contrast to software systems designed to play one specific game, systems capable of playing arbitrary unseen games cannot be provided with game-specific domain knowledge a priori. Instead, they must be endowed with high-level abilities to learn strategies and perform abstract reasoning. Successful realization of such programs poses many interesting research challenges for a wide variety of artificial-intelligence sub-areas including (but not limited to):

- knowledge representation and reasoning
- heuristic search and automated planning
- computational game theory
- multi-agent systems
- machine learning

The aim of this workshop is to bring together researchers from the above sub-fields of AI to discuss how best to address the challenges of and further advance the state-of-the-art of general game-playing systems and generic artificial intelligence.

The workshop is one-day long and will be held onsite at IJCAI during the scheduled workshop period August 3rd-5th (exact day is to be announced later).

Adversarial Search



Game Playing

- Many different kinds of games!

- Choices:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Perfect information (can you see the state)?

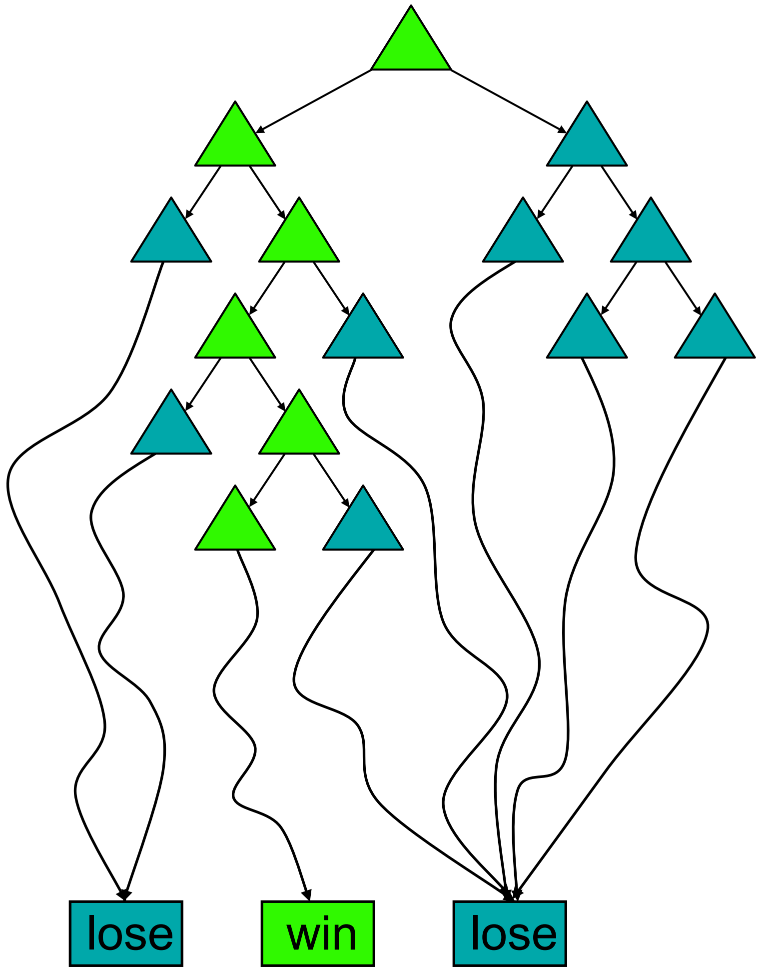
- Want algorithms for calculating a strategy (policy) which recommends a move in each state

Deterministic Games

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

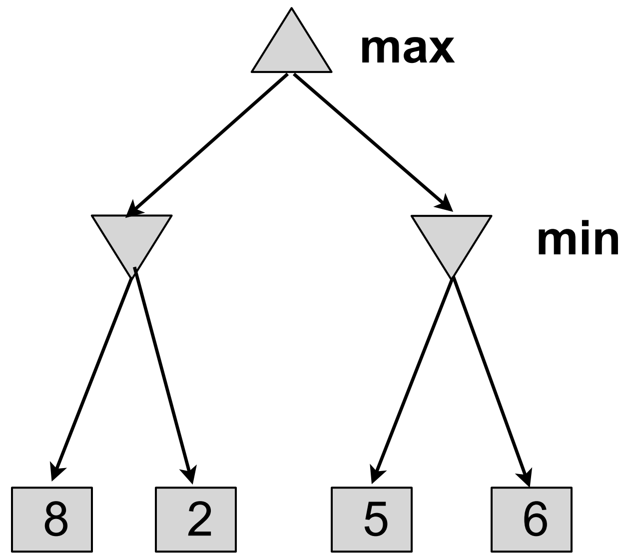
Deterministic Single-Player

- Deterministic, single player, perfect information:
  - Know the rules, action effects, winning states
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- … it's just search!
- Slight reinterpretation:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal outcome of its children (the max value)
  - Note that we don't have path sums as before (utilities at end)
- After search, can pick move that leads to best node

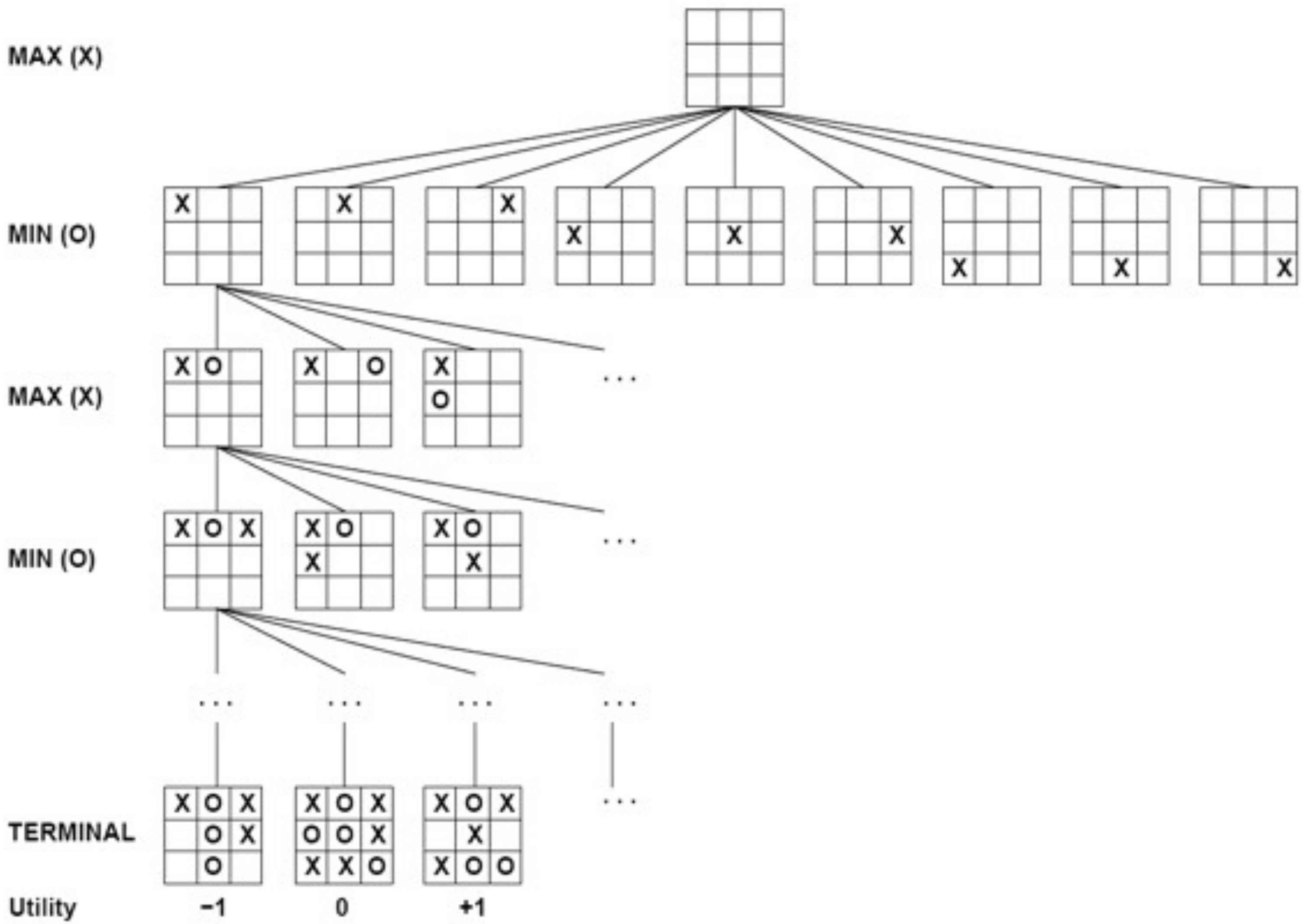


Deterministic Two-Player

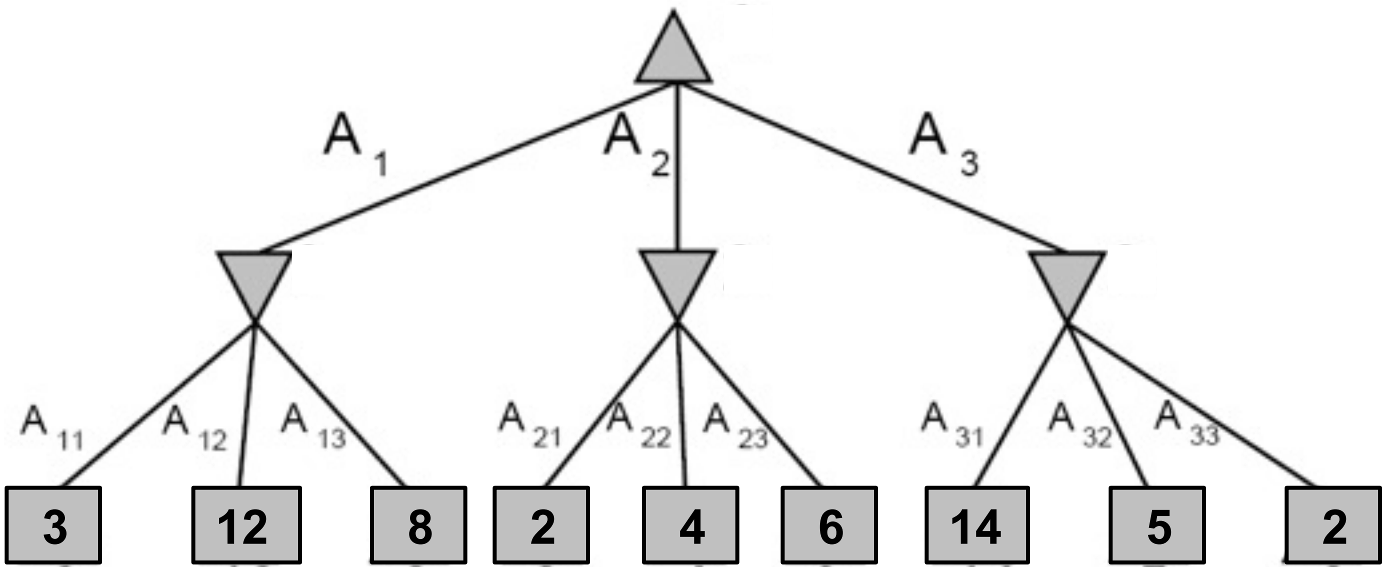
- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result
- **Minimax search**
  - A state-space search tree
  - Players alternate
  - Choose move to position with highest minimax value = best achievable utility against best play



Tic-tac-toe Game Tree



Minimax Example



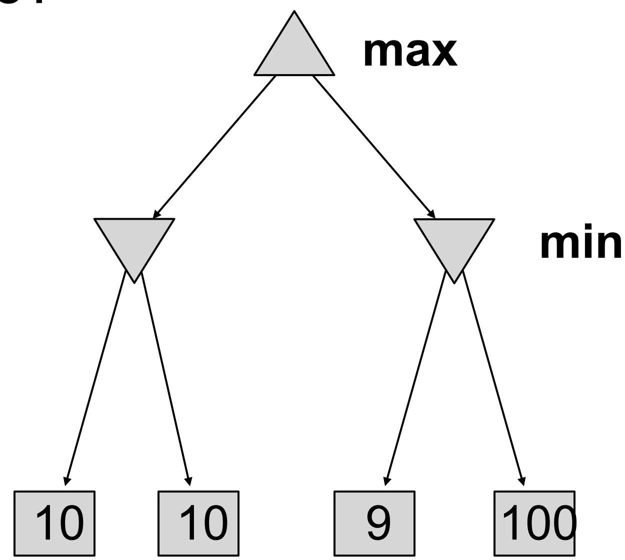
Minimax Search

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

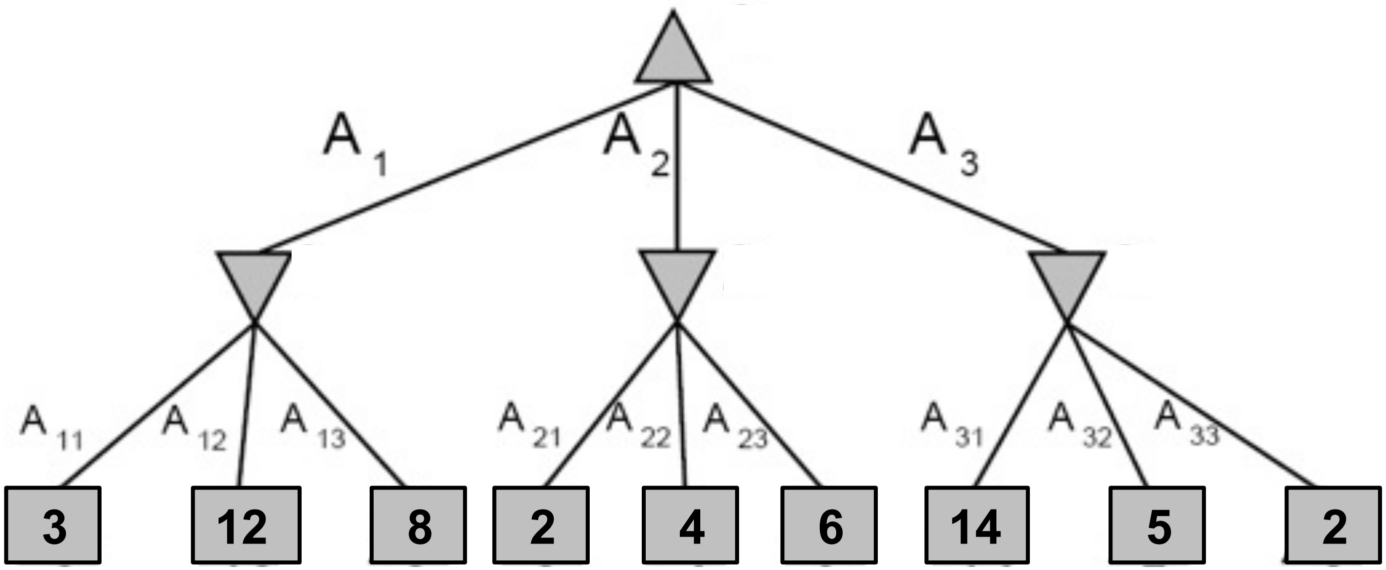
```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

Minimax Properties

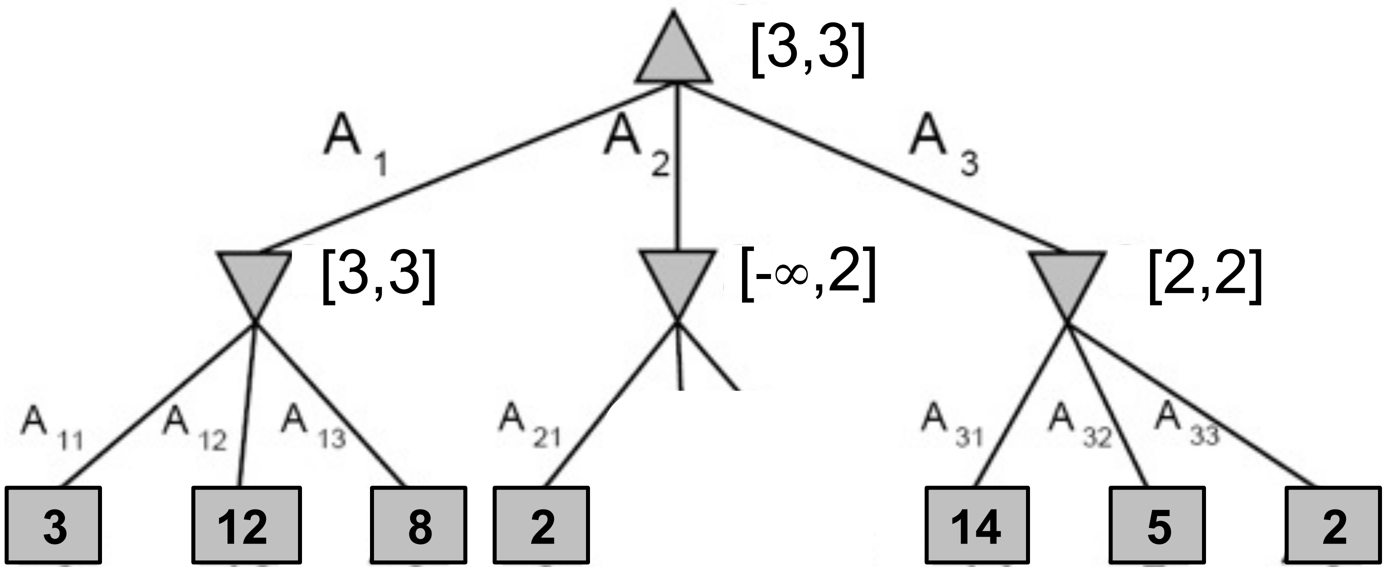
- Optimal?
  - Yes, against perfect player. Otherwise?
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - $O(bm)$
- For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Can we do better?

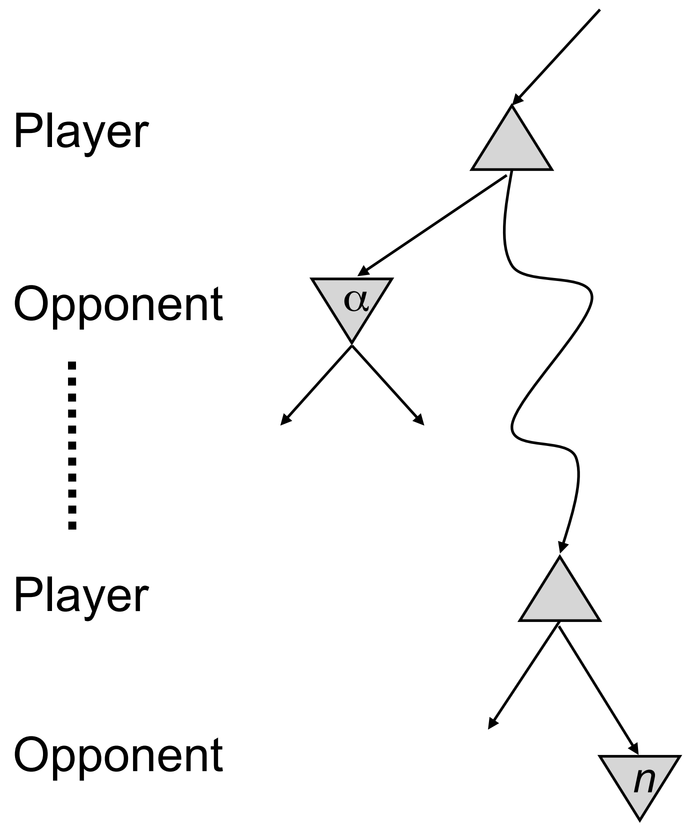


α-β Pruning Example



α-β Pruning

- General configuration
  - α is the best value that MAX can get at any choice point along the current path
  - If n becomes worse than α, MAX will avoid it, so can stop considering n's other children
  - Define β similarly for MIN



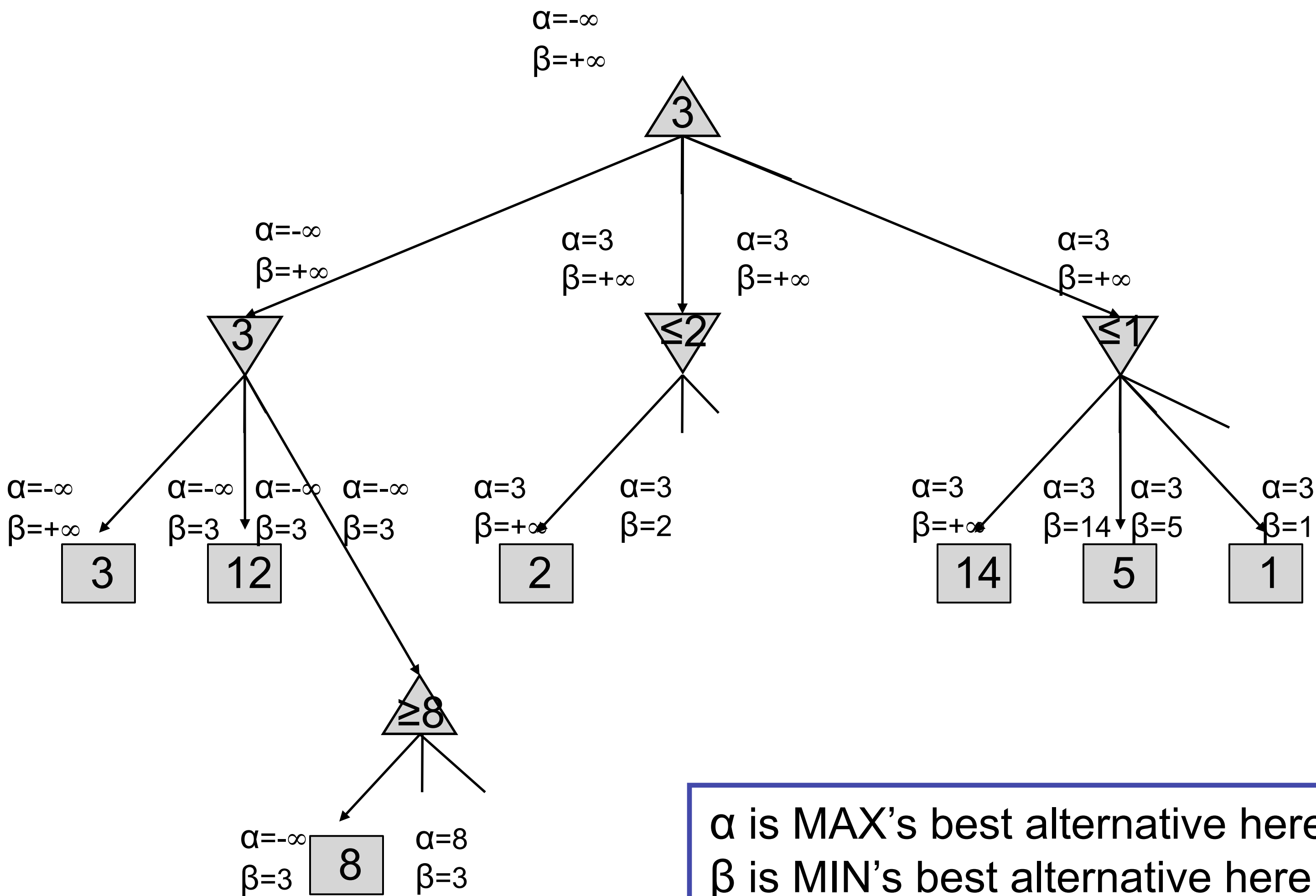
Alpha-Beta Pseudocode

inputs: *state*, current game state
α, value of best alternative for MAX on path to *state*
β, value of best alternative for MIN on path to *state*
returns: *a utility value*

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

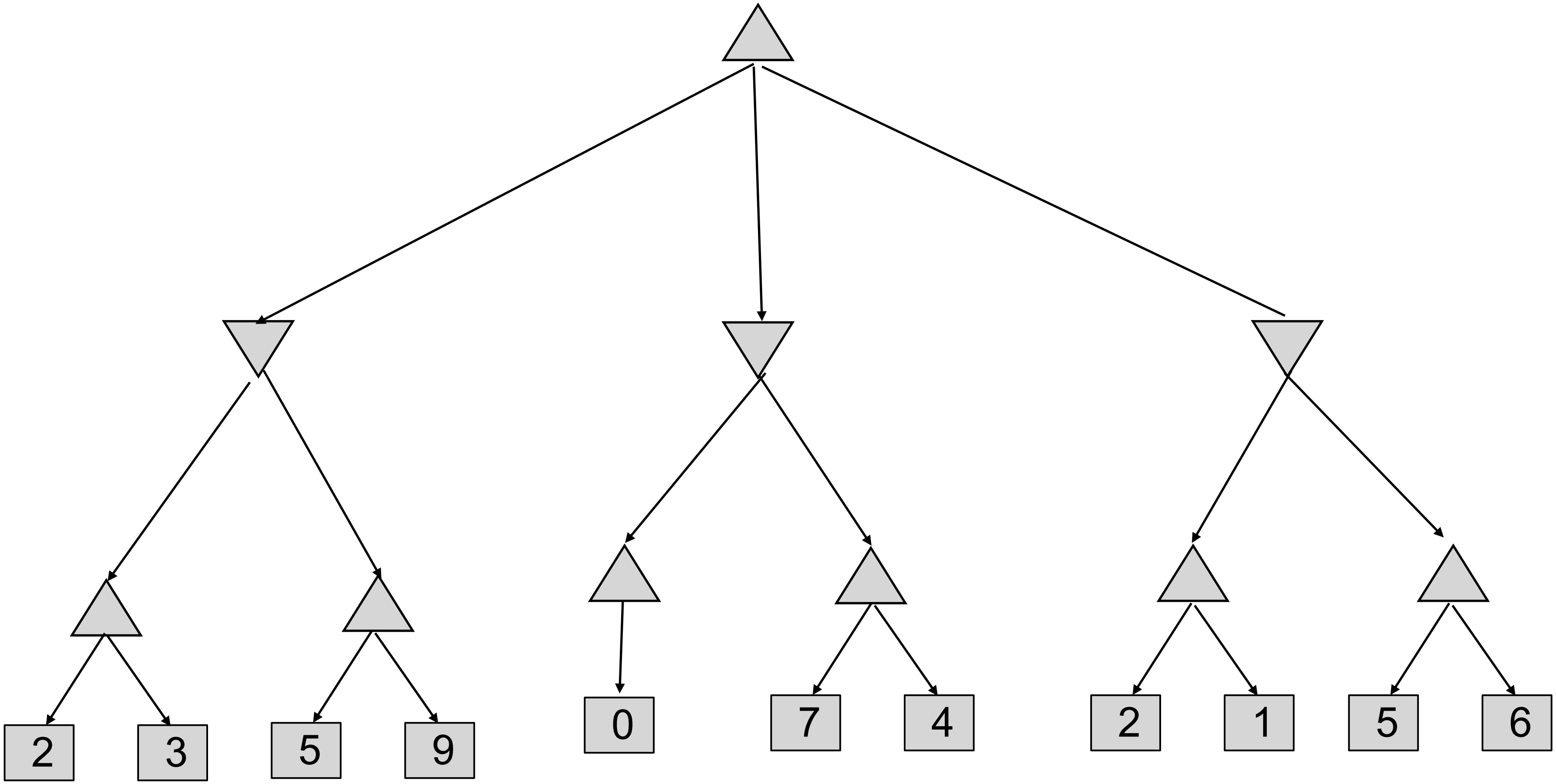
```
function MIN-VALUE(state,α,β)
    if TERMINAL-TEST(state) then
        return UTILITY(state)
    v ← +∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s,α,β))
        if v ≤ α then return v
        β ← MIN(β,v)
    return v
```

Alpha-Beta Pruning Example



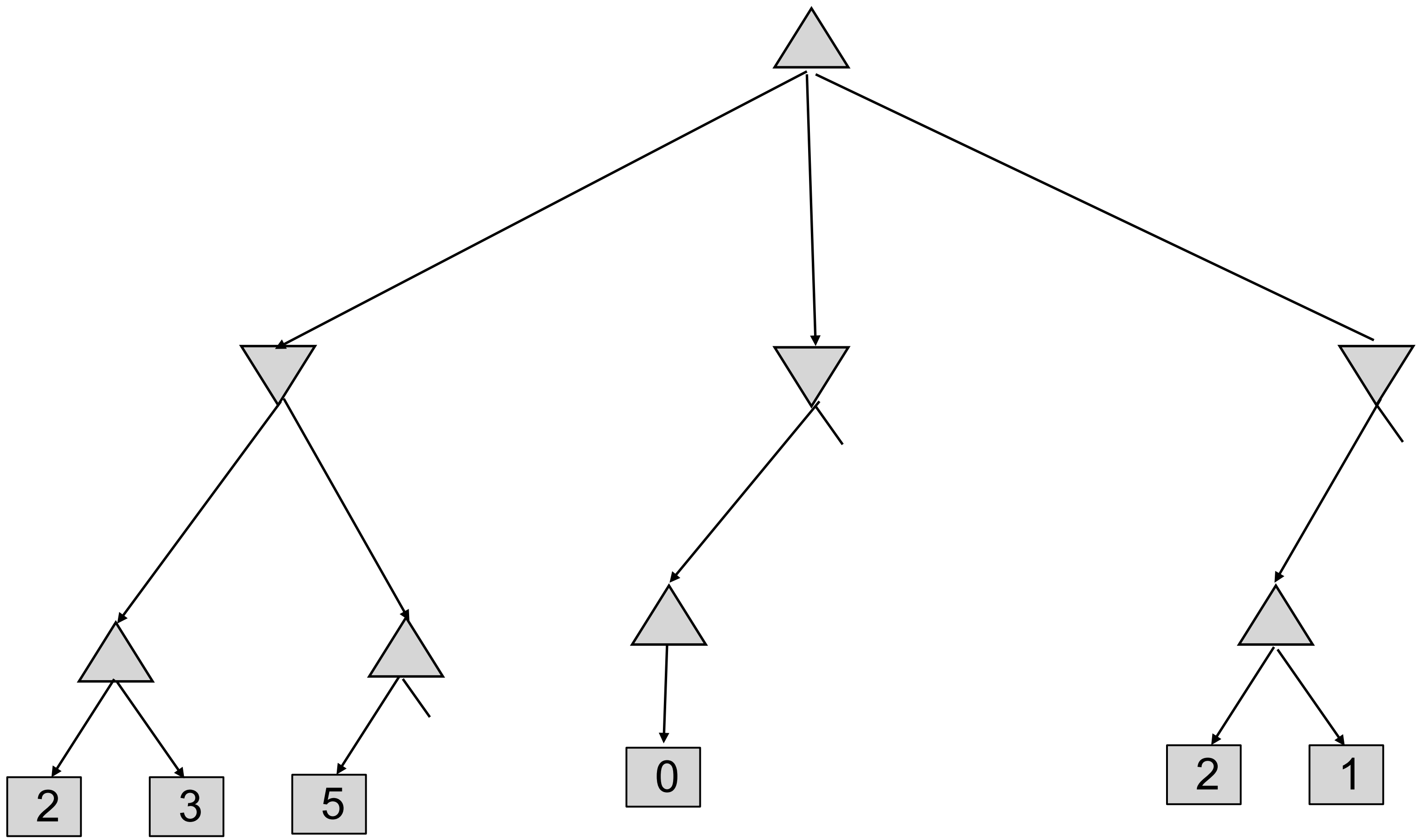
α is MAX's best alternative here or above
β is MIN's best alternative here or above

Alpha-Beta Pruning Example



α is MAX's best alternative here or above
β is MIN's best alternative here or above

Alpha-Beta Pruning Example



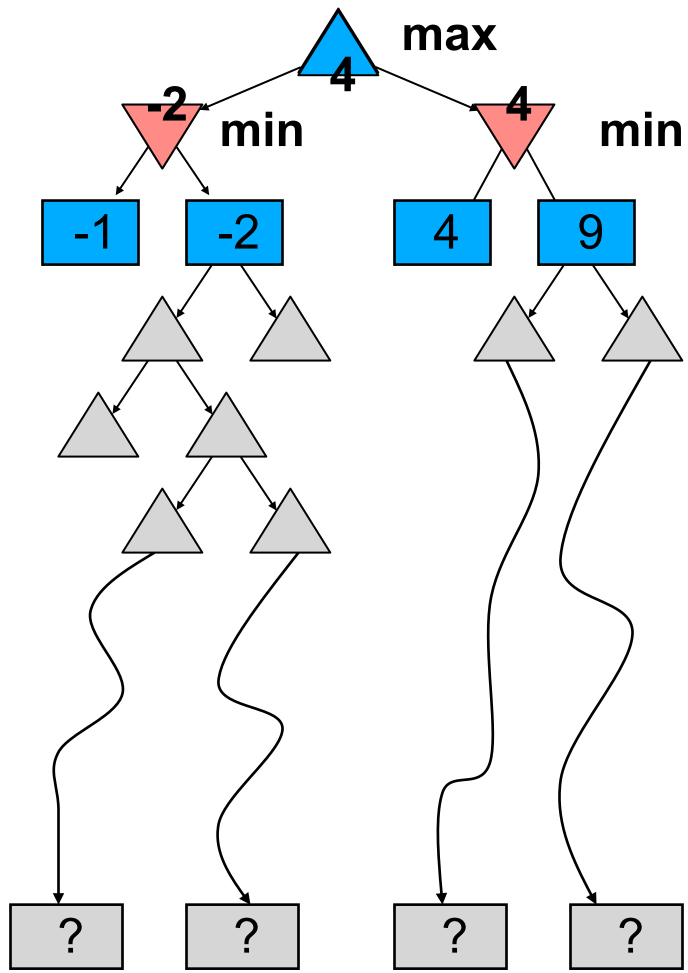
α is MAX's best alternative here or above
β is MIN's best alternative here or above

Alpha-Beta Pruning Properties

- This pruning has no effect on final result at the root
- Values of intermediate nodes might be wrong!
  - but, they are bounds
- Good child ordering improves effectiveness of pruning
- With “perfect ordering”:
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…

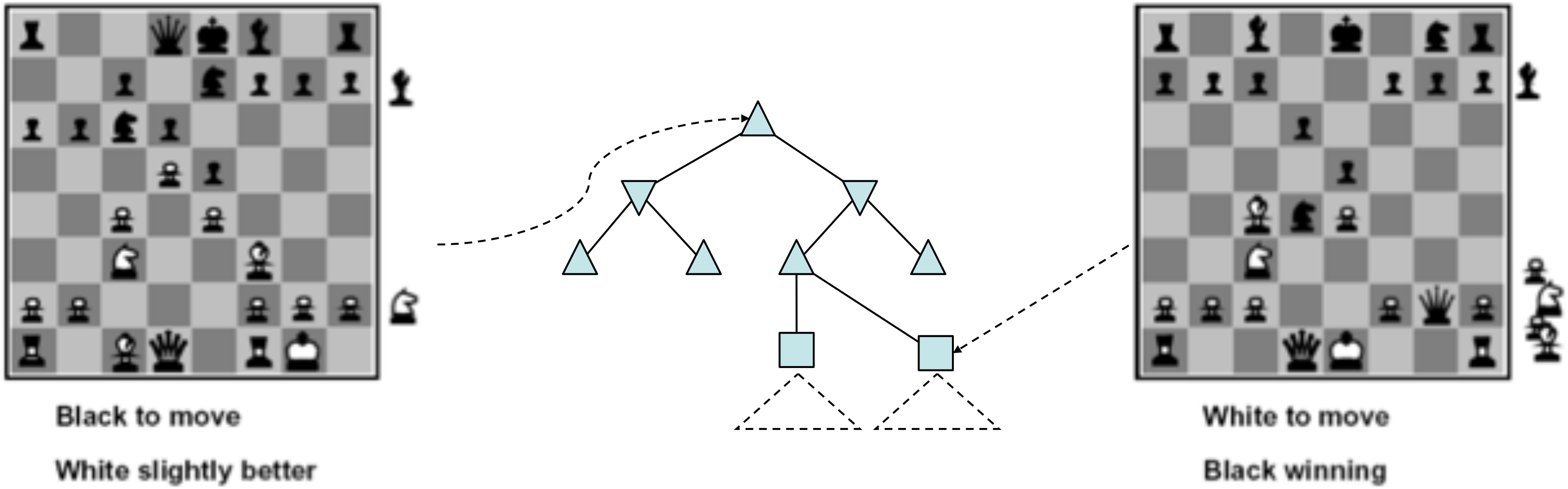
Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program



Evaluation Functions

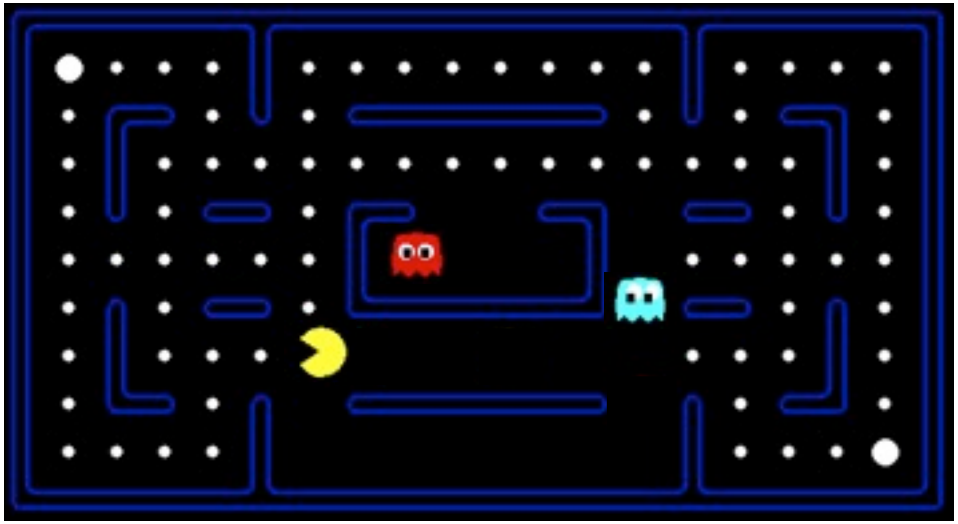
- Function which scores non-terminals



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

- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:
  - e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Evaluation for Pacman

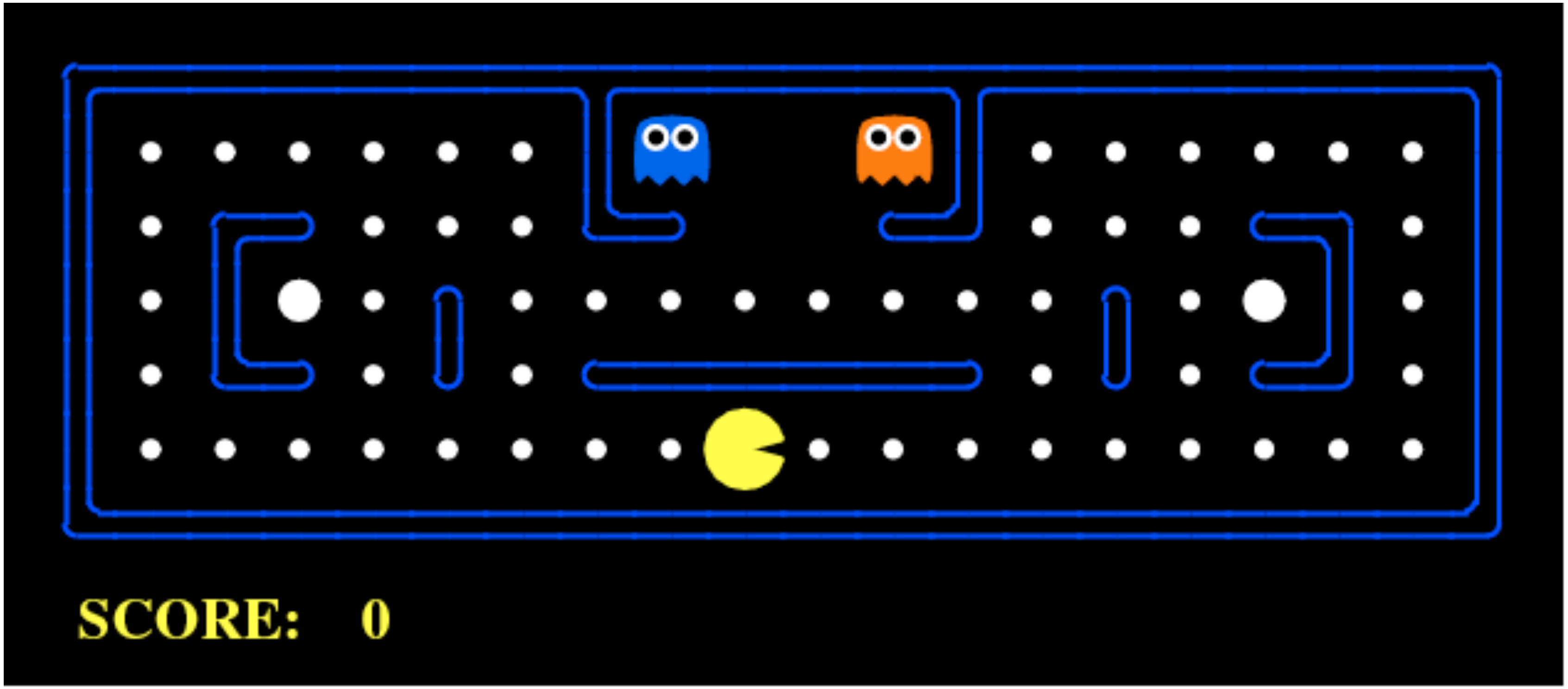


What features would be good for Pacman?

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

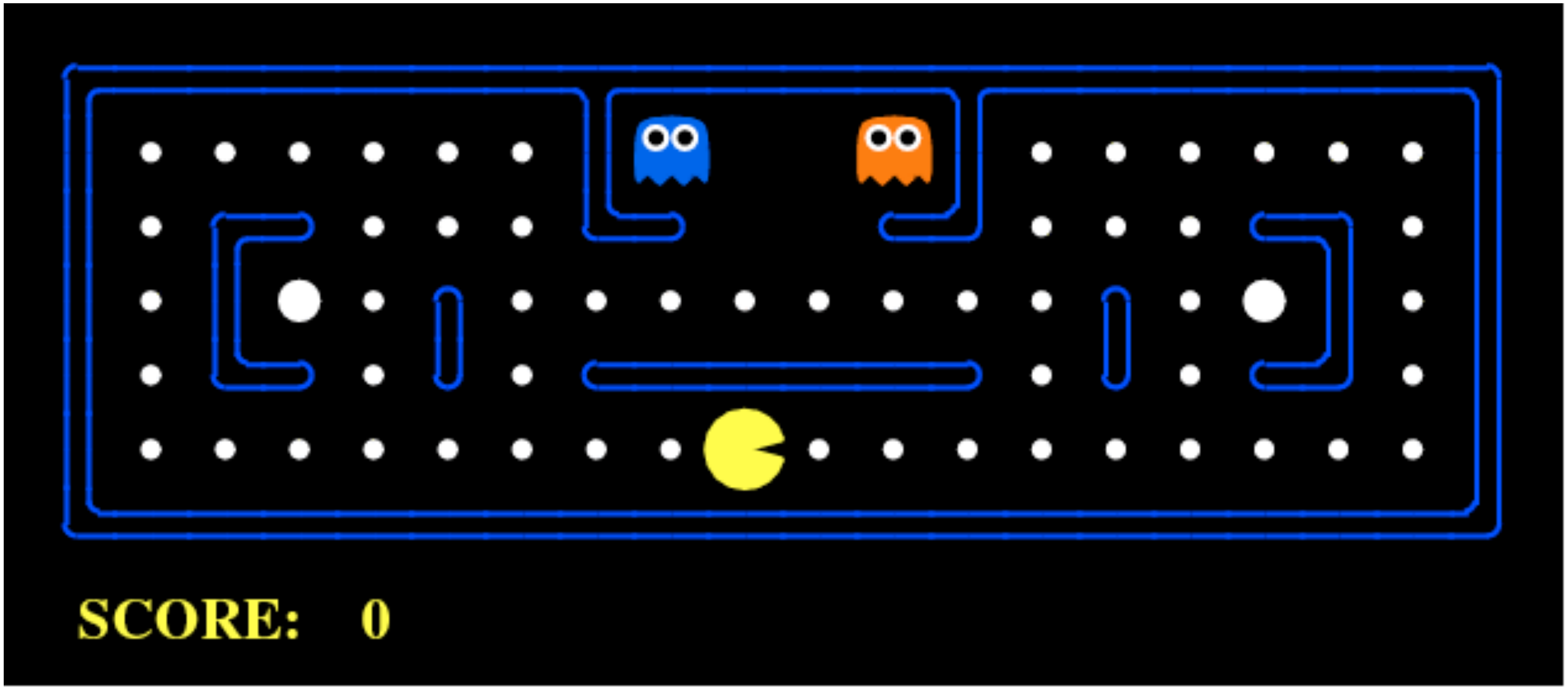
Which algorithm?

α-β, depth 4, simple eval fun



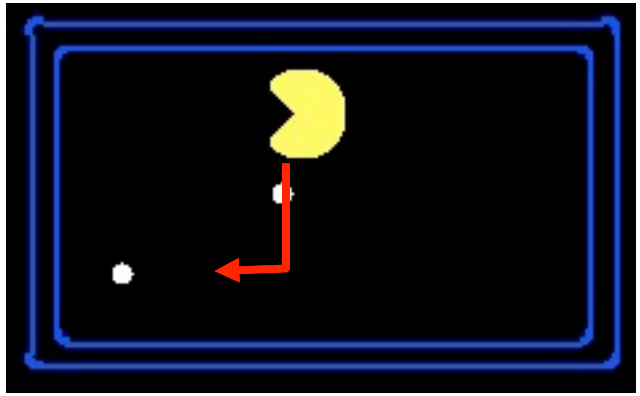
Which algorithm?

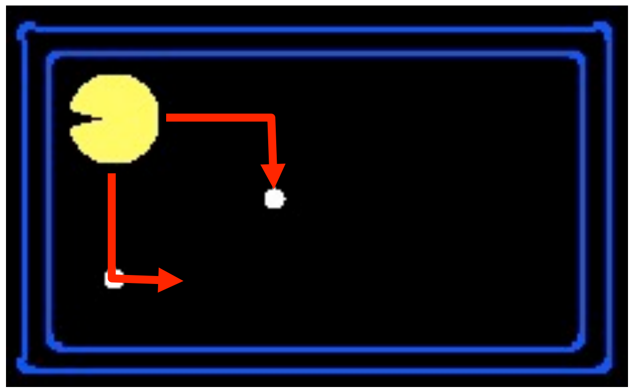
α-β, depth 4, better eval fun



Why Pacman Starves

- He knows his score will go up by eating the dot now
- He knows his score will go up just as much by eating the dot later on
- There are no point-scoring opportunities after eating the dot
- Therefore, waiting seems just as good as eating





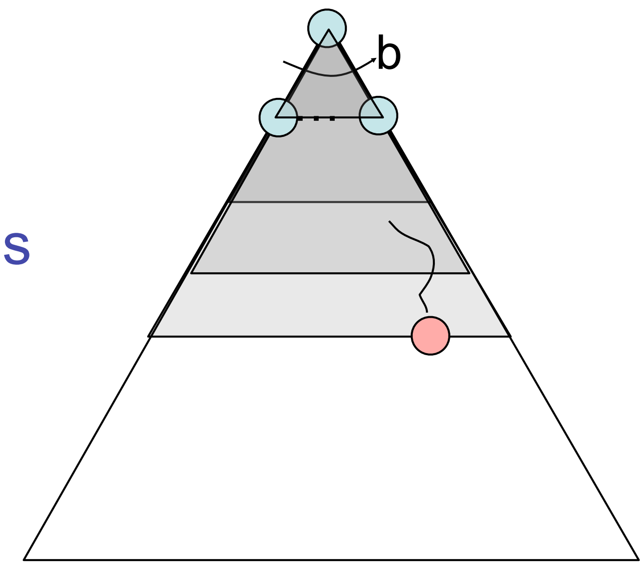
Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If "1" failed, do a DFS which only searches paths of length 2 or less.
3. If "2" failed, do a DFS which only searches paths of length 3 or less.

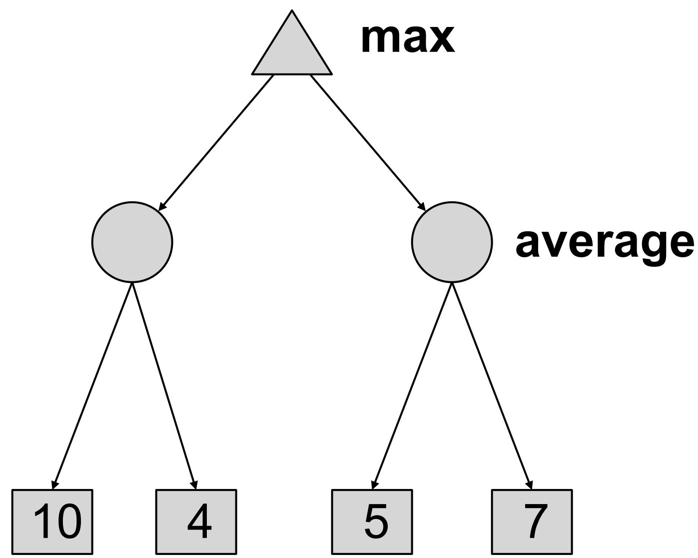
….and so on.

Why do we want to do this for multiplayer games?



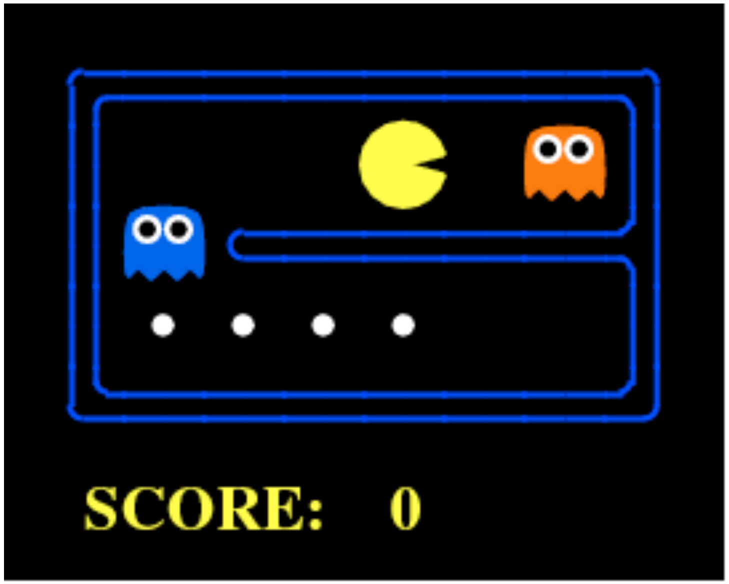
Stochastic Single-Player

- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, mine locations
- Can do **expectimax search**
  - Chance nodes, like actions except the environment controls the action chosen
  - Max nodes as before
  - Chance nodes take average (expectation) of value of children

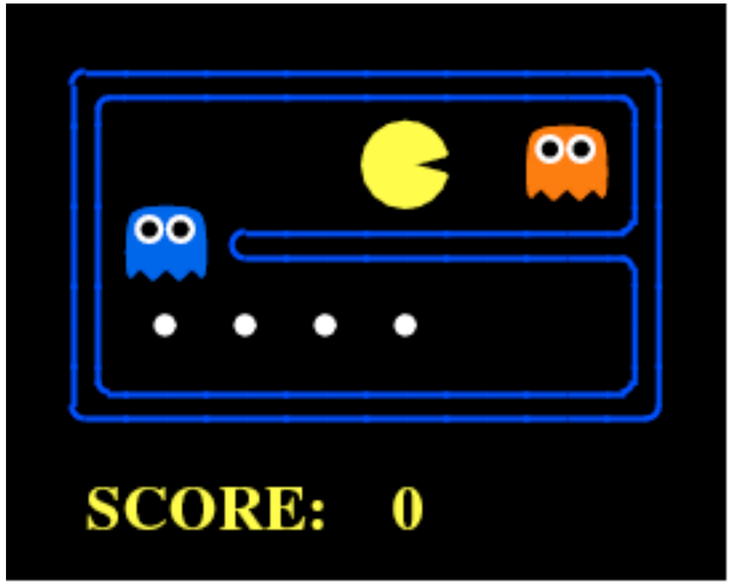


Which Algorithms?

Expectimax



Minimax



3 ply look ahead, ghosts move randomly

Maximum Expected Utility

- Why should we average utilities? Why not minimax?

- Principle of maximum expected utility: an agent should chose the action which maximizes its expected utility, given its knowledge
  - General principle for decision making
  - Often taken as the definition of rationality
  - We'll see this idea over and over in this course!

- Let's decompress this definition…

Reminder: Probabilities

- A random variable represents an event whose outcome is unknown
- A probability distribution is an assignment of weights to outcomes

- Example: traffic on freeway?
  - Random variable: T = whether there's traffic
  - Outcomes: T in {none, light, heavy}
  - Distribution: P(T=none) = 0.25, P(T=light) = 0.55, P(T=heavy) = 0.20

- Some laws of probability (more later):
  - Probabilities are always non-negative
  - Probabilities over all possible outcomes sum to one

- As we get more evidence, probabilities may change:
  - P(T=heavy) = 0.20, P(T=heavy | Hour=8am) = 0.60
  - We'll talk about methods for reasoning and updating probabilities later

What are Probabilities?

- Objectivist / frequentist answer:
  - Averages over repeated *experiments*
  - E.g. empirically estimating P(rain) from historical observation
  - E.g. pacman's estimate of what the ghost will do, given what it has done in the past
  - Assertion about how future experiments will go (in the limit)
  - Makes one think of *inherently random* events, like rolling dice

- Subjectivist / Bayesian answer:
  - Degrees of belief about unobserved variables
  - E.g. an agent's belief that it's raining, given the temperature
  - E.g. pacman's belief that the ghost will turn left, given the state
  - Often *learn* probabilities from past experiences (more later)
  - New evidence *updates beliefs* (more later)

Uncertainty Everywhere

- Not just for games of chance!
  - I’m sick: will I sneeze this minute?
  - Email contains “FREE!”: is it spam?
  - Tooth hurts: have cavity?
  - 60 min enough to get to the airport?
  - Robot rotated wheel three times, how far did it advance?
  - Safe to cross street? (Look both ways!)

- Sources of uncertainty in random variables:
  - Inherently random process (dice, etc)
  - Insufficient or weak evidence
  - Ignorance of underlying processes
  - Unmodeled variables
  - The world’s just noisy – it doesn’t behave according to plan!

Reminder: Expectations

- We can define function f(X) of a random variable X
- The expected value of a function is its average value, weighted by the probability distribution over inputs
- Example: How long to get to the airport?
  - Length of driving time as a function of traffic:
    L(none) = 20, L(light) = 30, L(heavy) = 60
  - What is my expected driving time?
    - Notation: $E_{P(T)}[\ L(T)\ ]$
    - Remember, P(T) = {none: 0.25, light: 0.5, heavy: 0.25}
    - E[L(T)] = L(none) * P(none) + L(light) * P(light) + L(heavy) * P(heavy)
    - E[L(T)] = (20 * 0.25) + (30 * 0.5) + (60 * 0.25) = 35

Utilities

- Utilities are functions from outcomes (states of the world) to real numbers that describe an agent's preferences

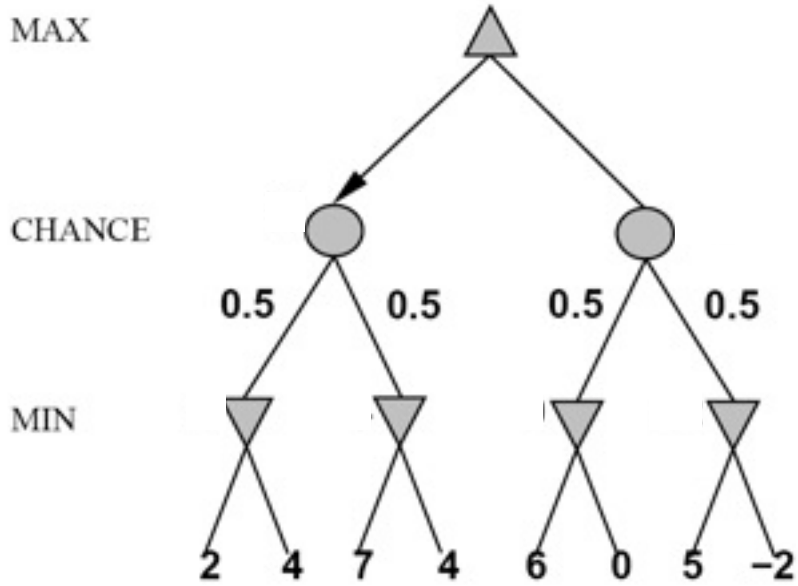
- Where do utilities come from?
  - In a game, may be simple (+1/-1)
  - Utilities summarize the agent's goals
  - Theorem: any set of preferences between outcomes can be summarized as a utility function (provided the preferences meet certain conditions)

- In general, we hard-wire utilities and let actions emerge (why don't we let agents decide their own utilities?)

- More on utilities soon…

Stochastic Two-Player

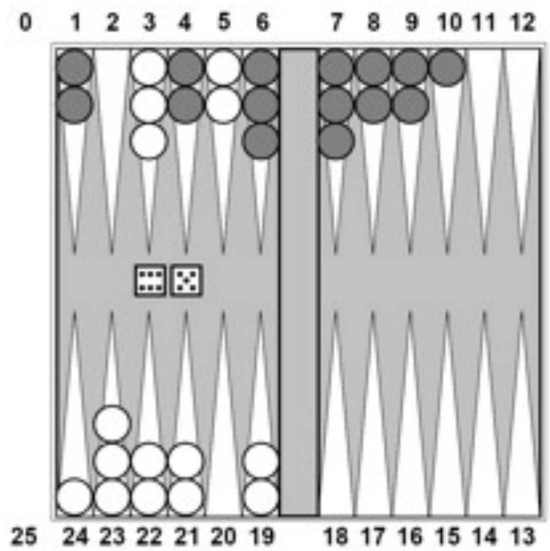
- E.g. backgammon
- Expectiminimax (!)
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax



if *state* is a MAX node **then**
  return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a MIN node **then**
  return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a chance node **then**
  return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

Stochastic Two-Player

- Dice rolls increase *b*: 21 possible rolls with 2 dice
  - Backgammon ≈ 20 legal moves
  - Depth 4 = 20 x $(21 \times 20)^3$ = 1.2 x 10^9
- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible…
- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play

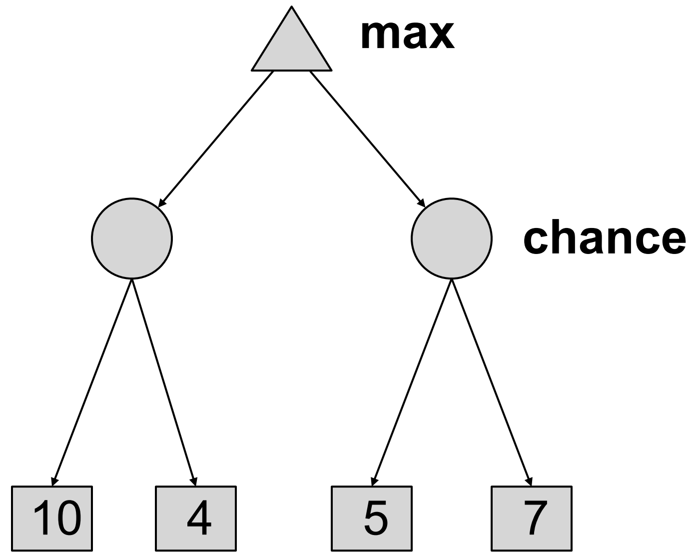


Expectimax Search Trees

- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In minesweeper, mine locations
  - In pacman, the ghosts act randomly

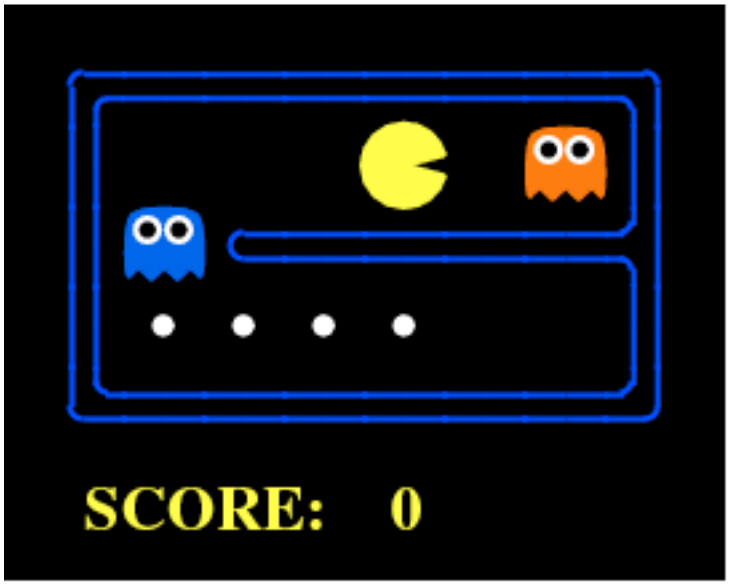
- Can do **expectimax search**
  - Chance nodes, like min nodes, except the outcome is uncertain
  - Calculate expected utilities
  - Max nodes as in minimax search
  - Chance nodes take average (expectation) of value of children

- Later, we'll learn how to formalize the underlying problem as a **Markov Decision Process**



Which Algorithm?

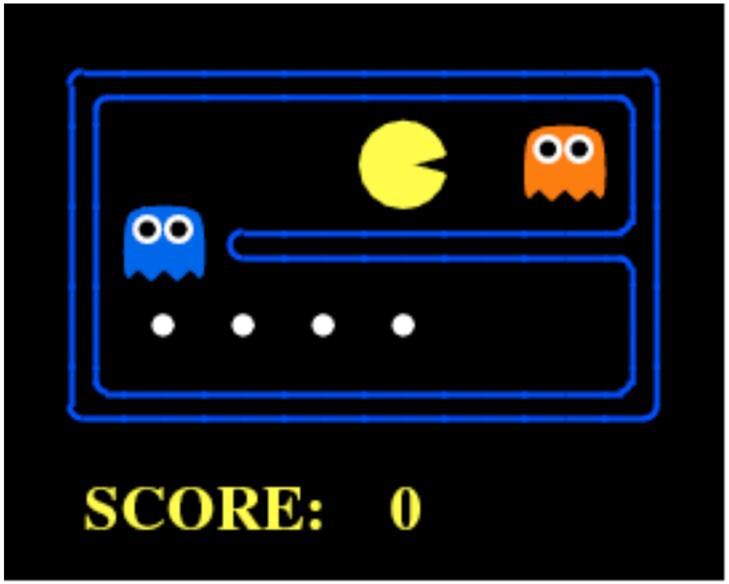
Minimax: no point in trying



3 ply look ahead, ghosts move randomly

Which Algorithm?

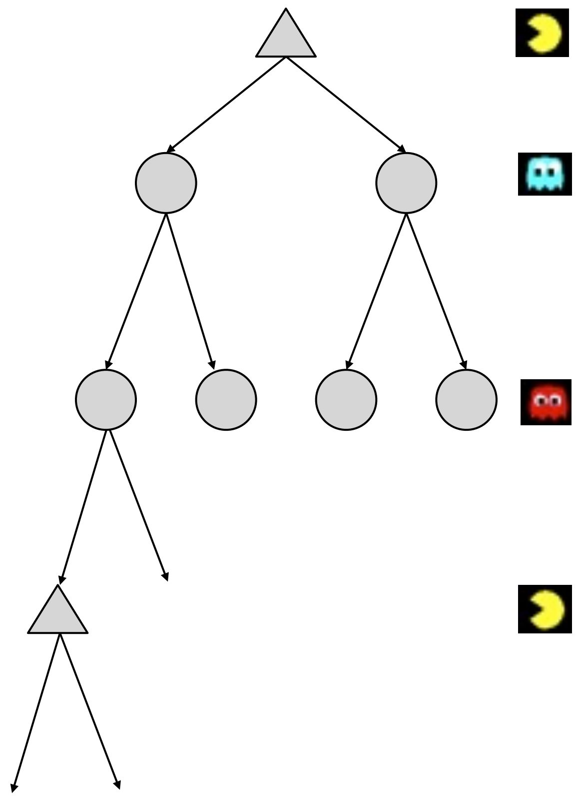
Expectimax: wins some of the time



3 ply look ahead, ghosts move randomly

Expectimax Search

- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a node for every outcome out of our control: opponent or environment
  - The model might say that adversarial actions are likely!
- For now, assume for any state we magically have a distribution to assign probabilities to opponent actions / environment outcomes

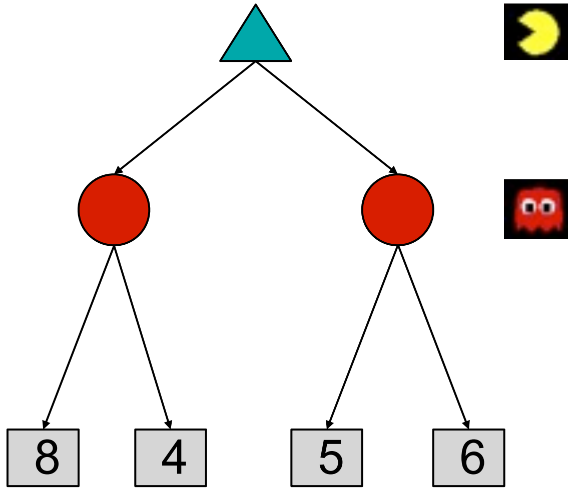


Expectimax Pseudocode

```
def value(s)
    if s is a max node return maxValue(s)
    if s is an exp node return expValue(s)
    if s is a terminal node return evaluation(s)

def maxValue(s)
    values = [value(s’) for s’ in successors(s)]
    return max(values)

def expValue(s)
    values = [value(s’) for s’ in successors(s)]
    weights = [probability(s, s’) for s’ in successors(s)]
    return expectation(values, weights)
```



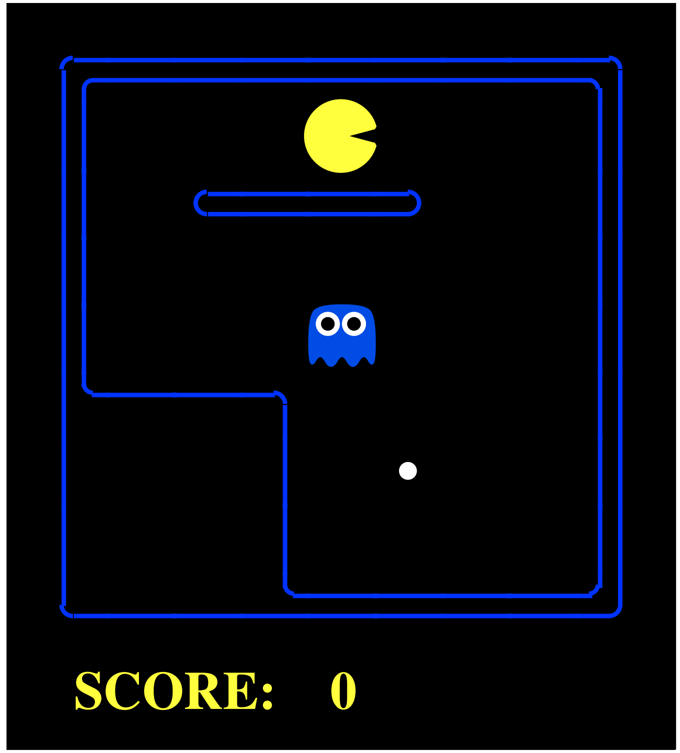
Expectimax for Pacman

- Notice that we've gotten away from thinking that the ghosts are trying to minimize pacman's score
- Instead, they are now a part of the environment
- Pacman has a belief (distribution) over how they will act
- Quiz: Can we see minimax as a special case of expectimax?
- Quiz: what would pacman's computation look like if we assumed that the ghosts were doing 1-ply minimax and taking the result 80% of the time, otherwise moving randomly?

Expectimax for Pacman

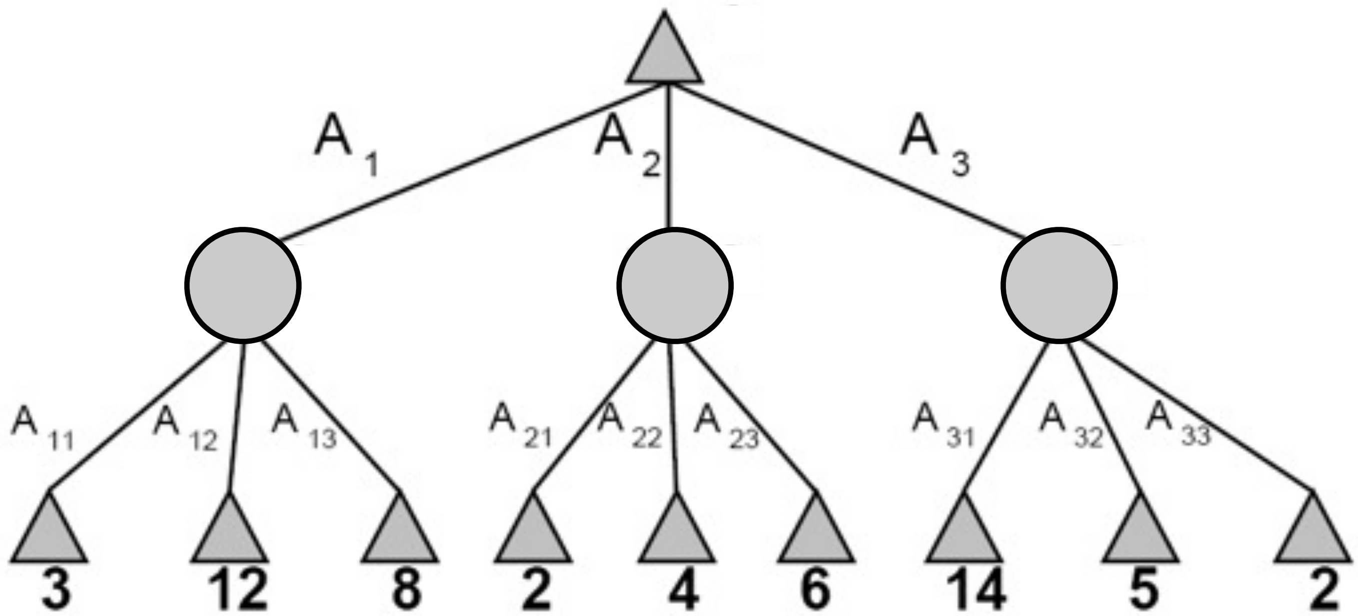
Results from playing 5 games

	Minimizing Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 493	Won 5/5 Avg. Score: 483
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503



Pacman does depth 4 search with an eval function that avoids trouble
Minimizing ghost does depth 2 search with an eval function that seeks Pacman

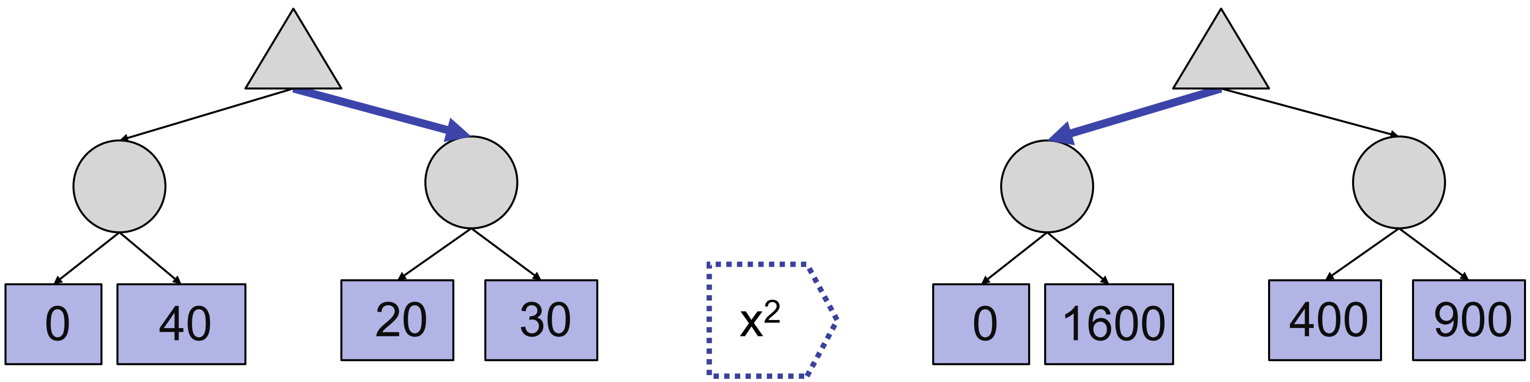
Expectimax Pruning?



- Not easy
  - exact: need bounds on possible values
  - approximate: sample high-probability branches

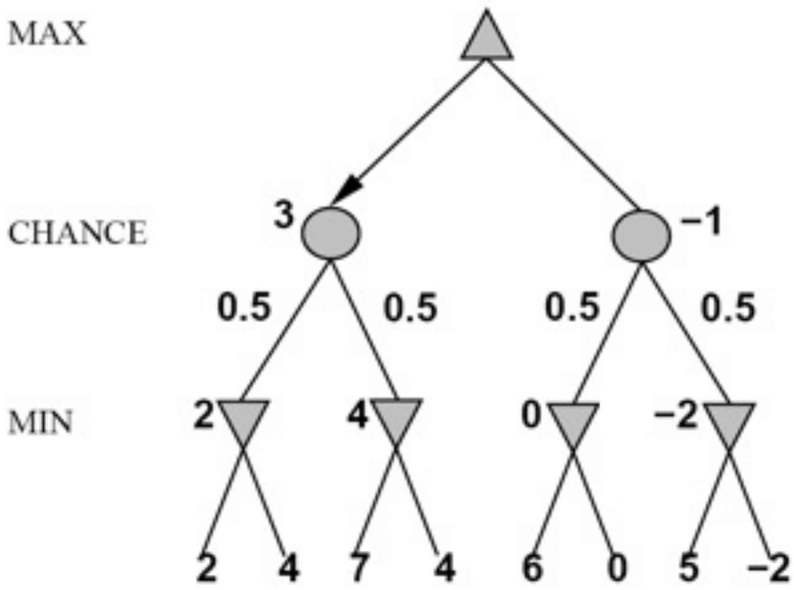
Expectimax Evaluation

- Evaluation functions quickly return an estimate for a node's true value (which value, expectimax or minimax?)
- For minimax, evaluation function scale doesn't matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations
- For expectimax, we need *magnitudes* to be meaningful



Mixed Layer Types

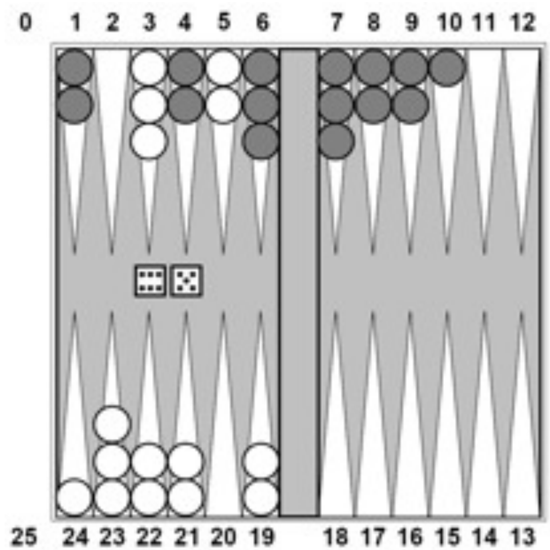
- E.g. Backgammon
- Expectiminimax
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax



if *state* is a MAX node **then**
  return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a MIN node **then**
  return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a chance node **then**
  return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

Stochastic Two-Player

- Dice rolls increase *b*: 21 possible rolls with 2 dice
  - Backgammon ≈ 20 legal moves
  - Depth 4 = 20 x (21 x 20)^3 1.2 x 10^9
- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible…
- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play



Multi-player Non-Zero-Sum Games

- Similar to minimax:
  - Utilities are now tuples
  - Each player maximizes their own entry at each node
  - Propagate (or back up) nodes from children
  - Can give rise to cooperation and competition dynamically…