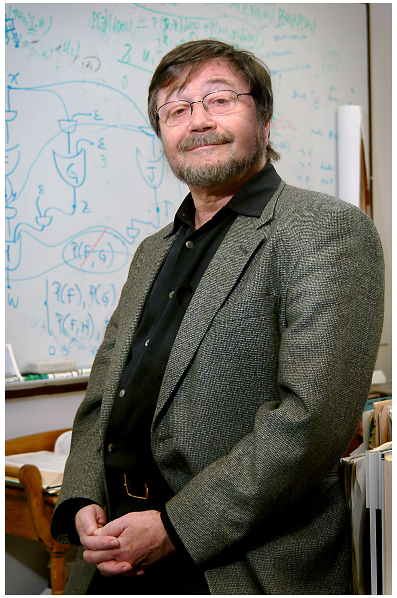
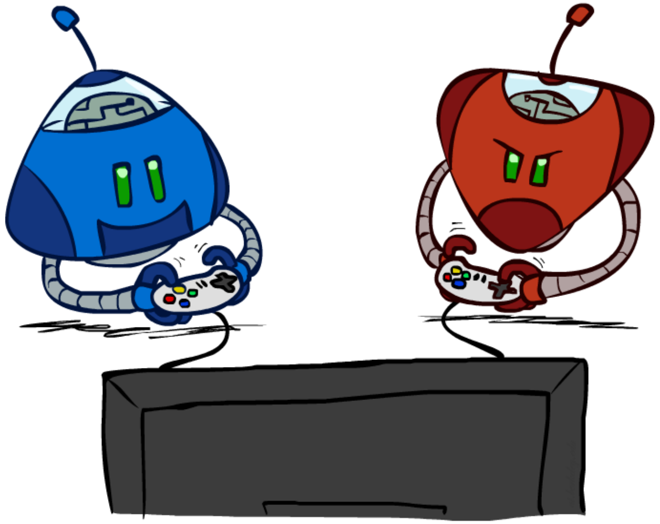


Artificial Intelligence



Adversarial Search

CS 444 – Spring 2021

Dr. Kevin Molloy

Department of Computer Science

James Madison University

JMU JAMES MADISON UNIVERSITY.

Much of this lecture is taken from
Dan Klein and Pieter Abbeel AI class at UC Berkeley

Today

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

- **Go:** Human champions are now starting to be challenged by machines. In go, b > 300! Classic programs use pattern knowledge bases, but big recent advances use neural networks developed by Google's DeepMind research group (Alpha Go -- https://deepmind.com/research/case-studies/alphago-the-story-so-far)

- **Pacman**

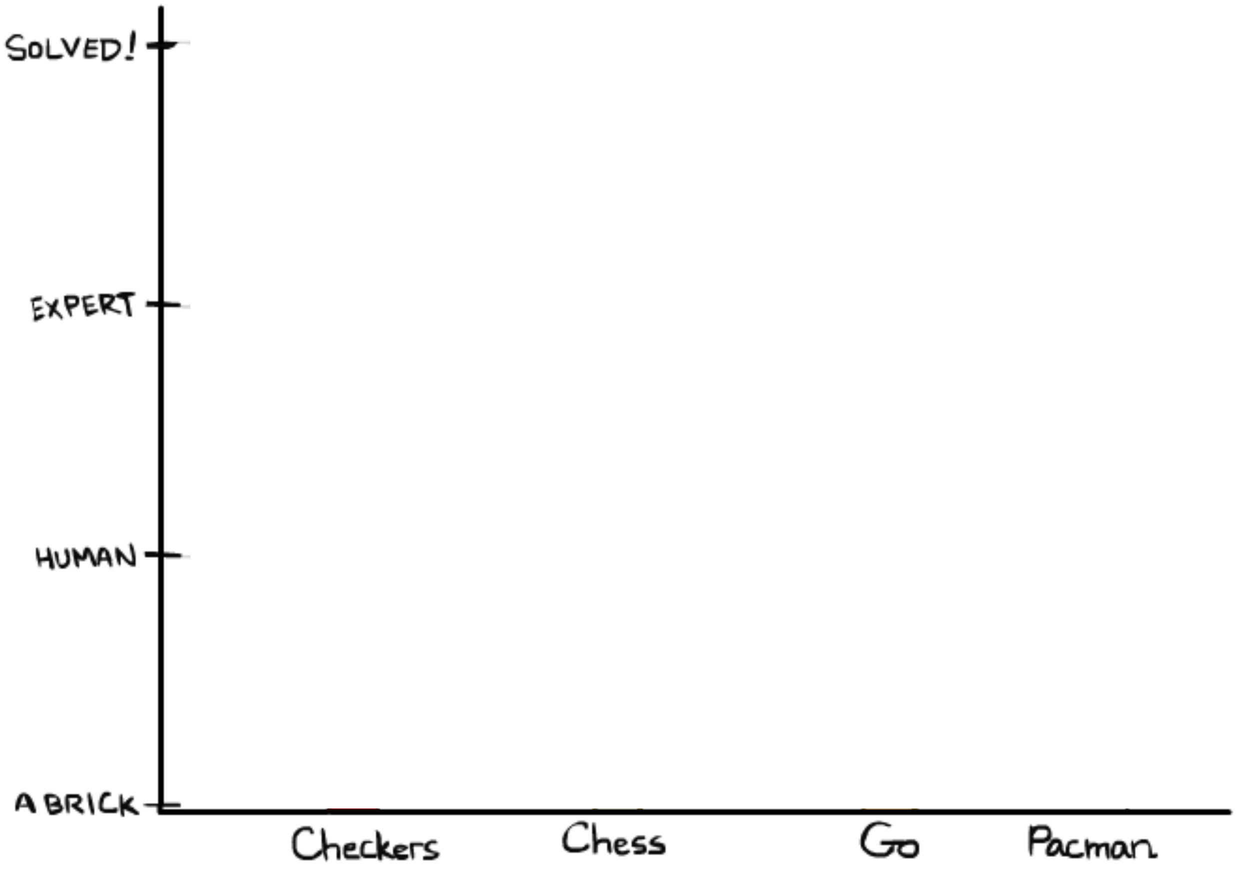


Figure from Berkley AI

What is Going On Under the Covers?

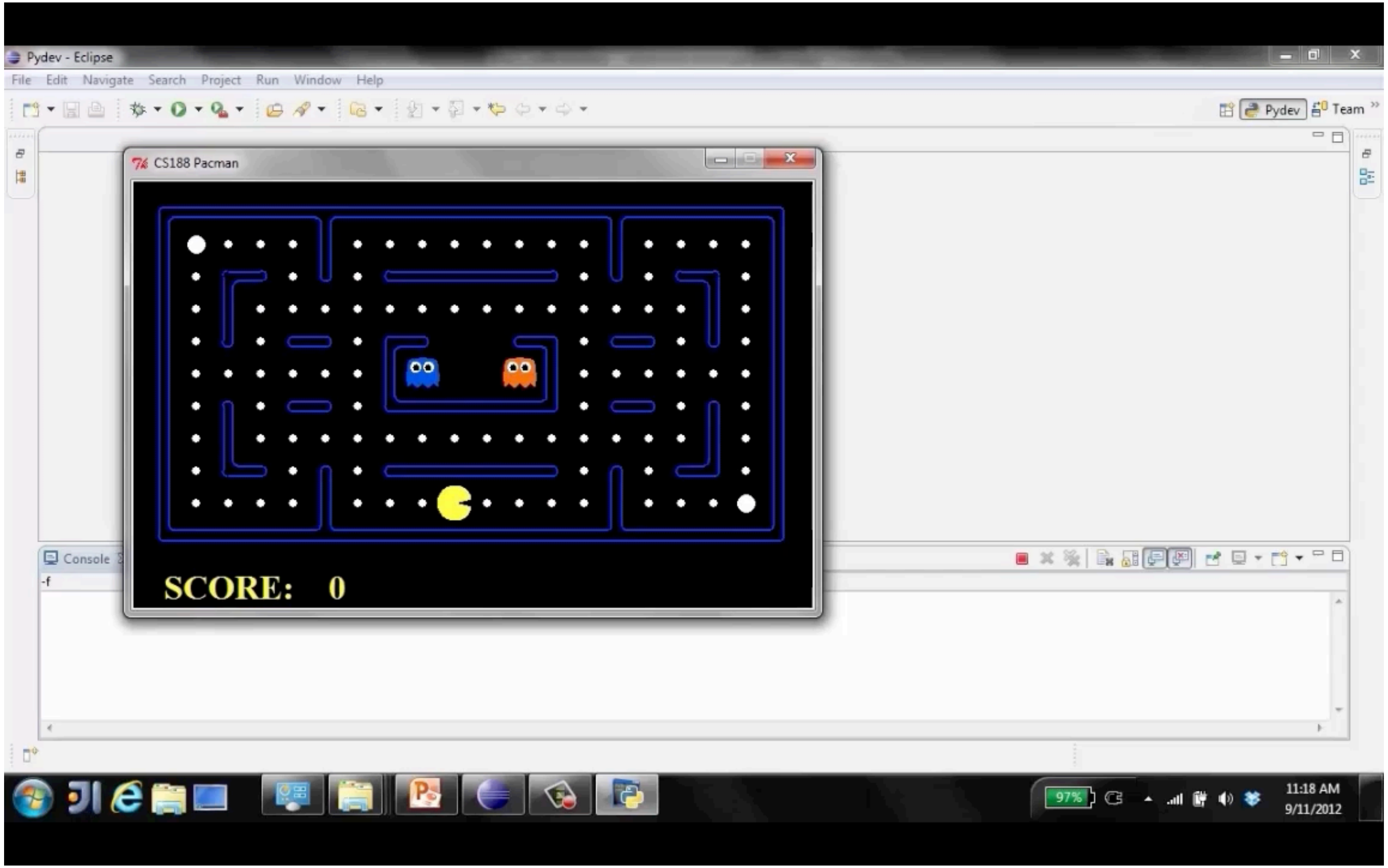


Figure from Berkley AI

Types of Games

- Many different kinds of games!

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

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

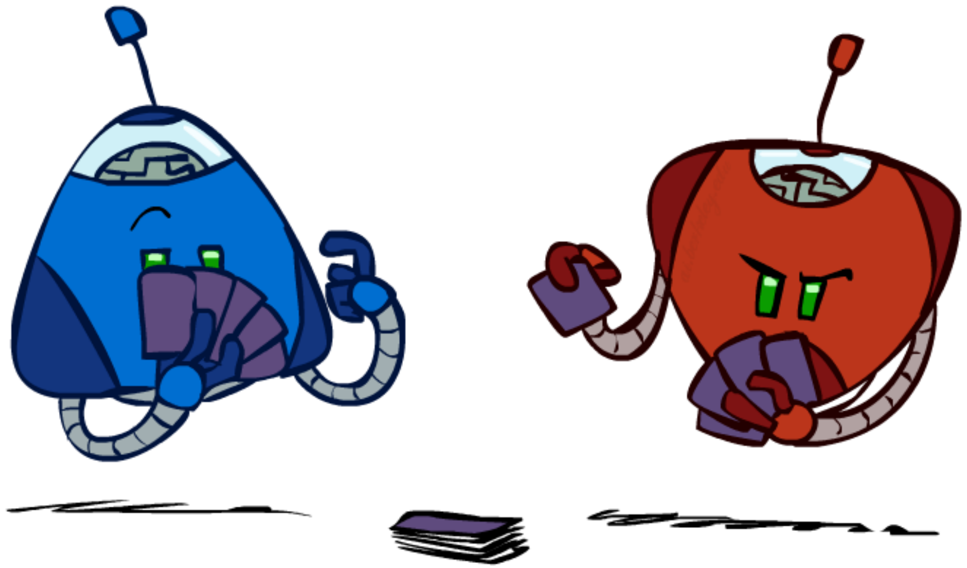


Figure from Berkley AI

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: SxA $\rightarrow$ S
  - Terminal Test: S $\rightarrow$ {t,f}
  - Terminal Utilities: SxP $\rightarrow$ R

- Solution for a player is a policy: S $\rightarrow$ A

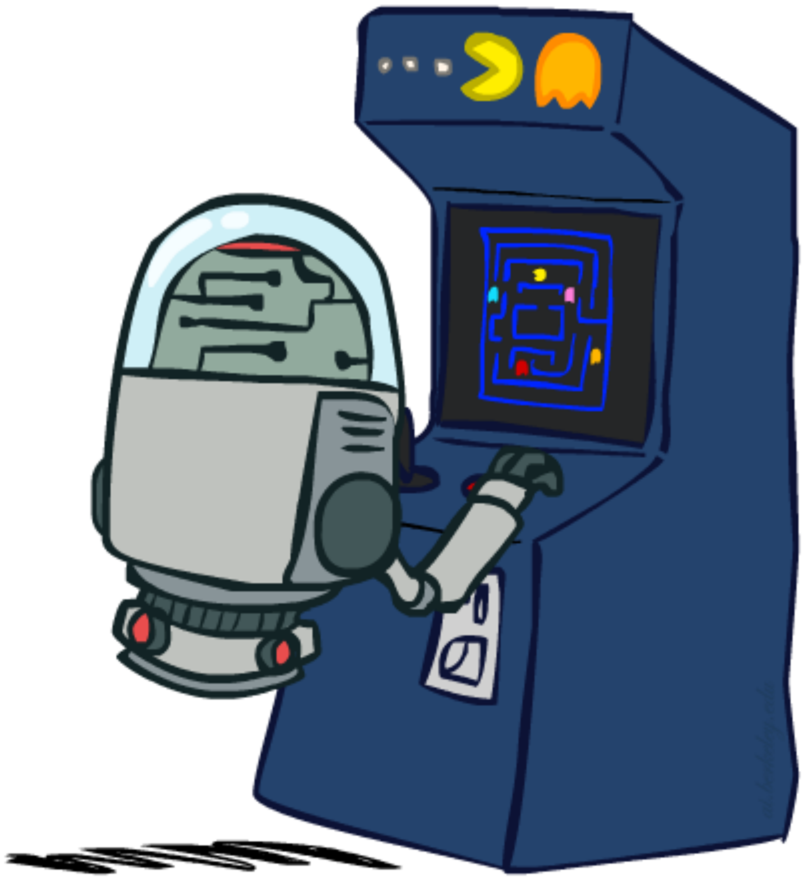
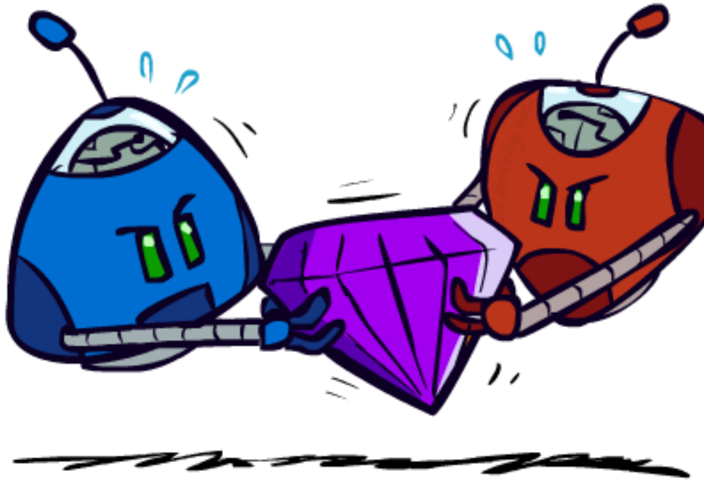
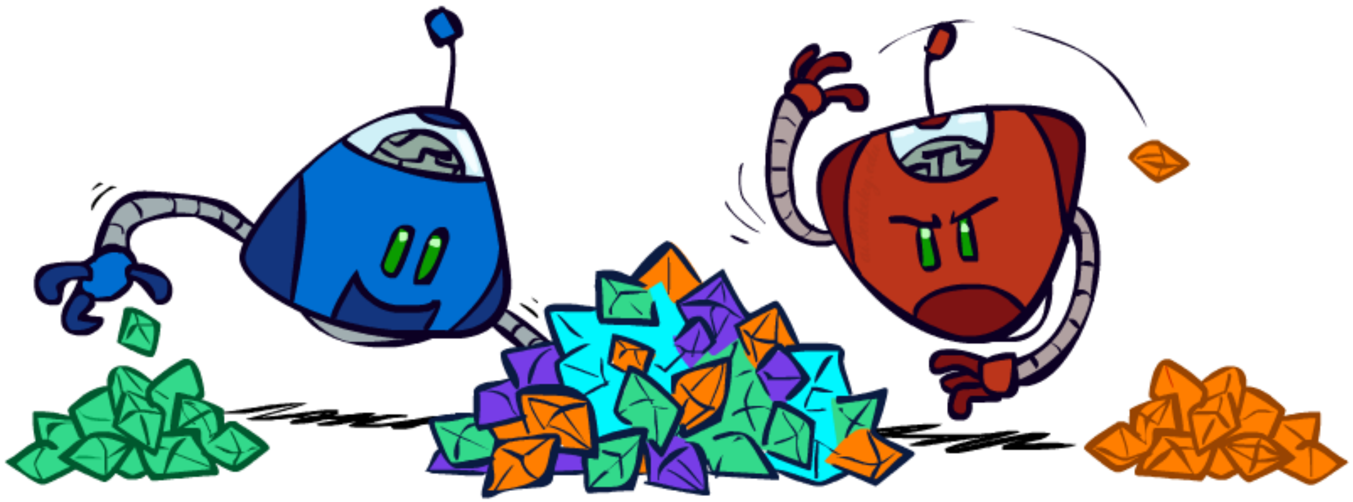


Figure from Berkley AI

Zero-Sum Games





- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition

- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
  - More later on non-zero-sum games

JAMES MADISON UNIVERSITY

Figure from Berkley AI

Adversarial Search

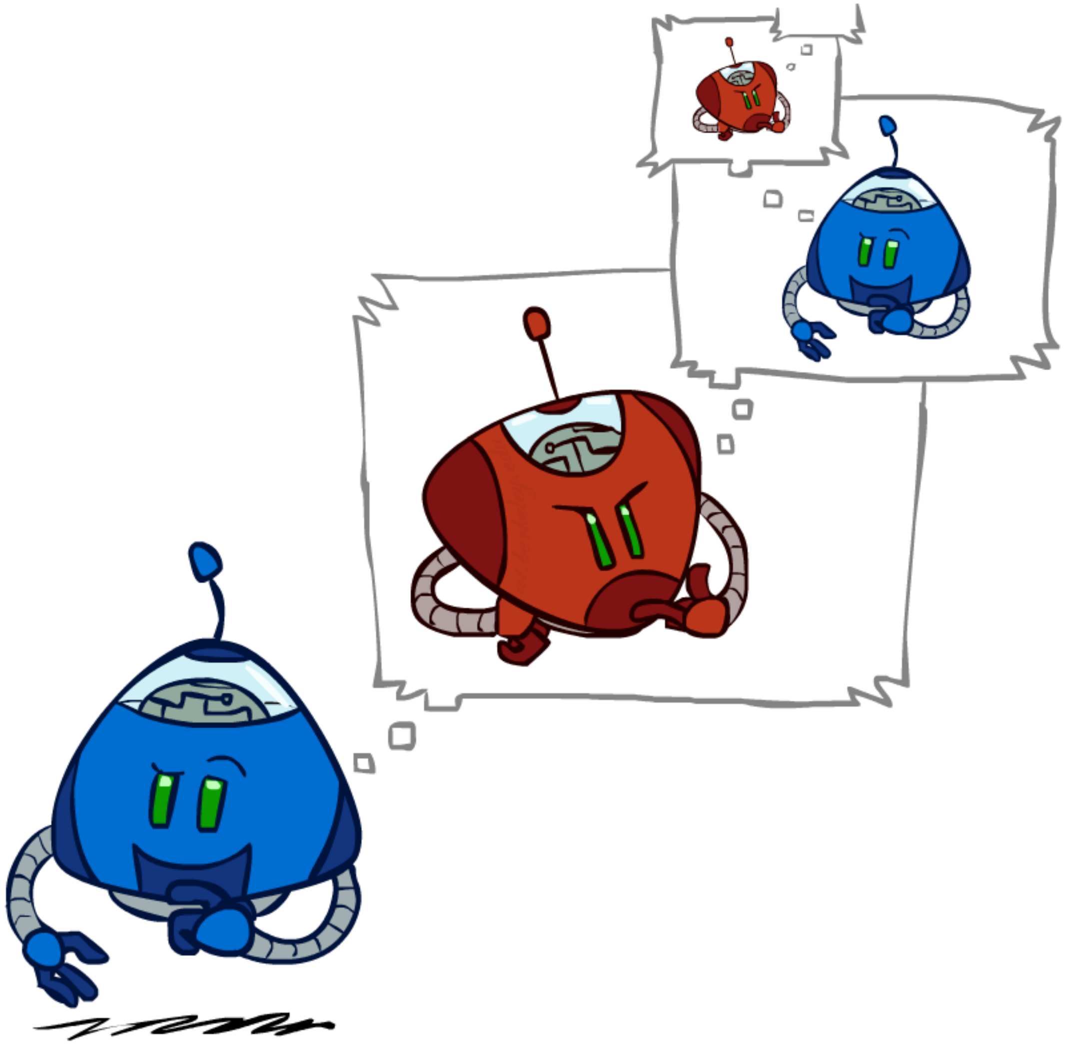


Figure from Berkley AI

Single-Agent Trees

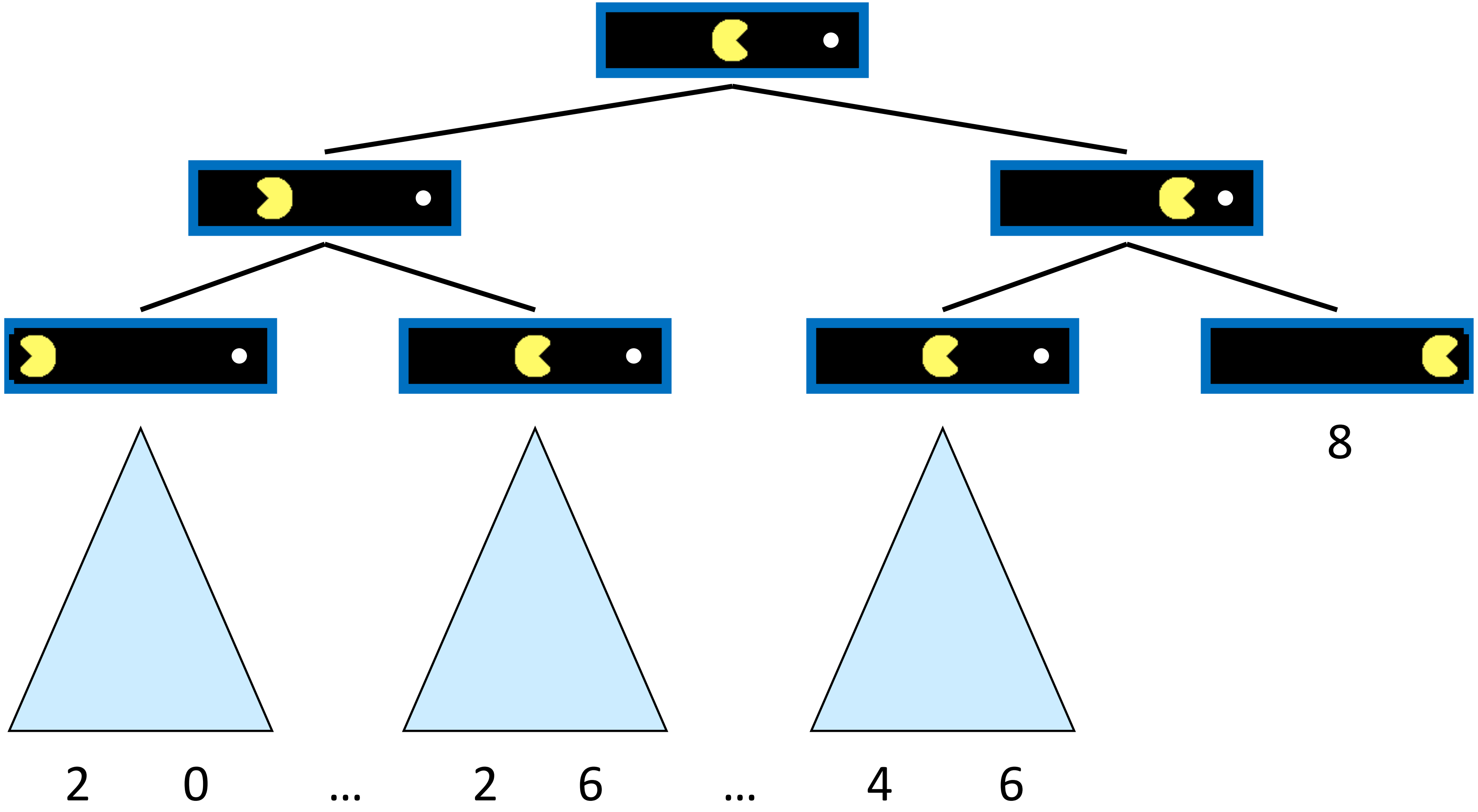


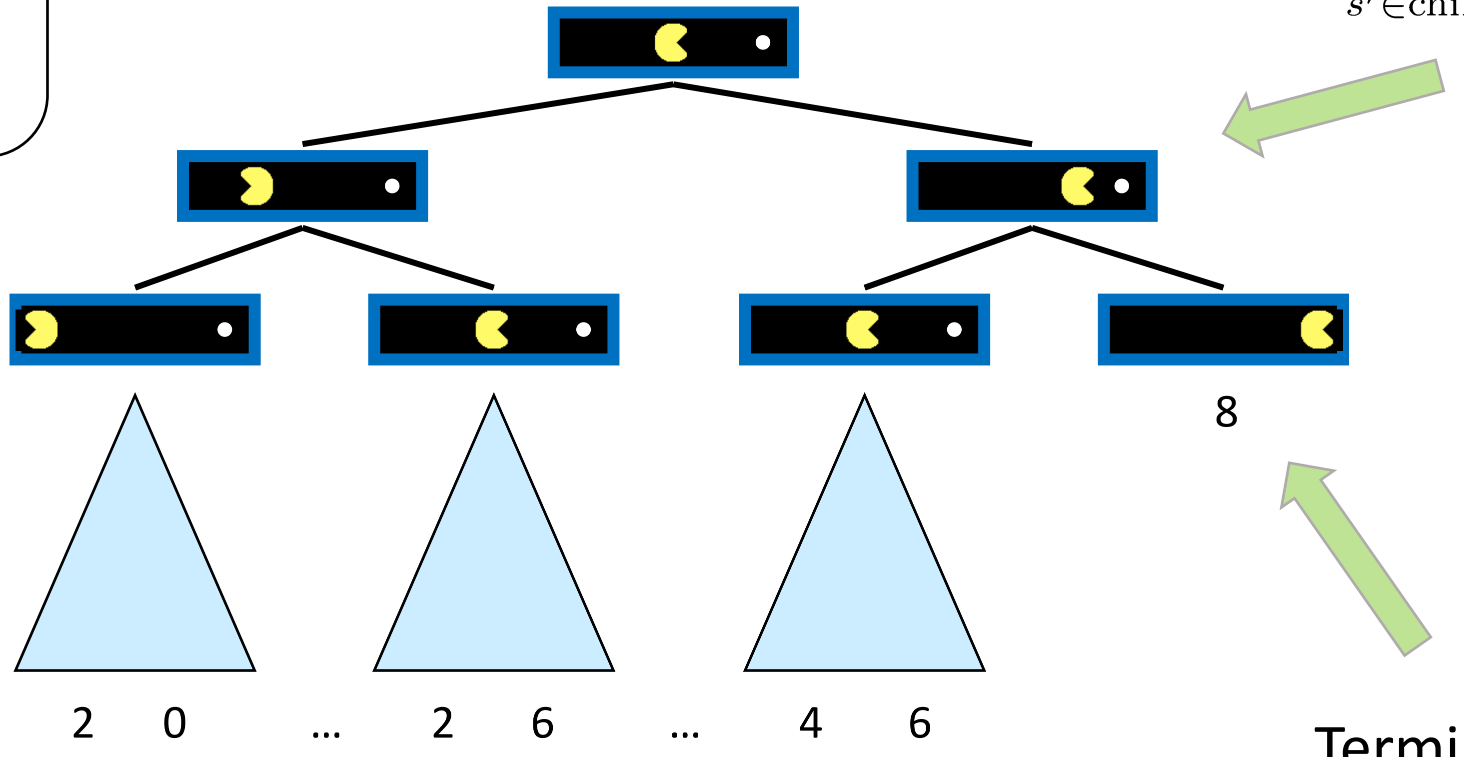
Figure from Berkley AI

Value of a State

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

Non-Terminal States:

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



Terminal States:

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

Figure from Berkley AI

Adversarial Game Trees

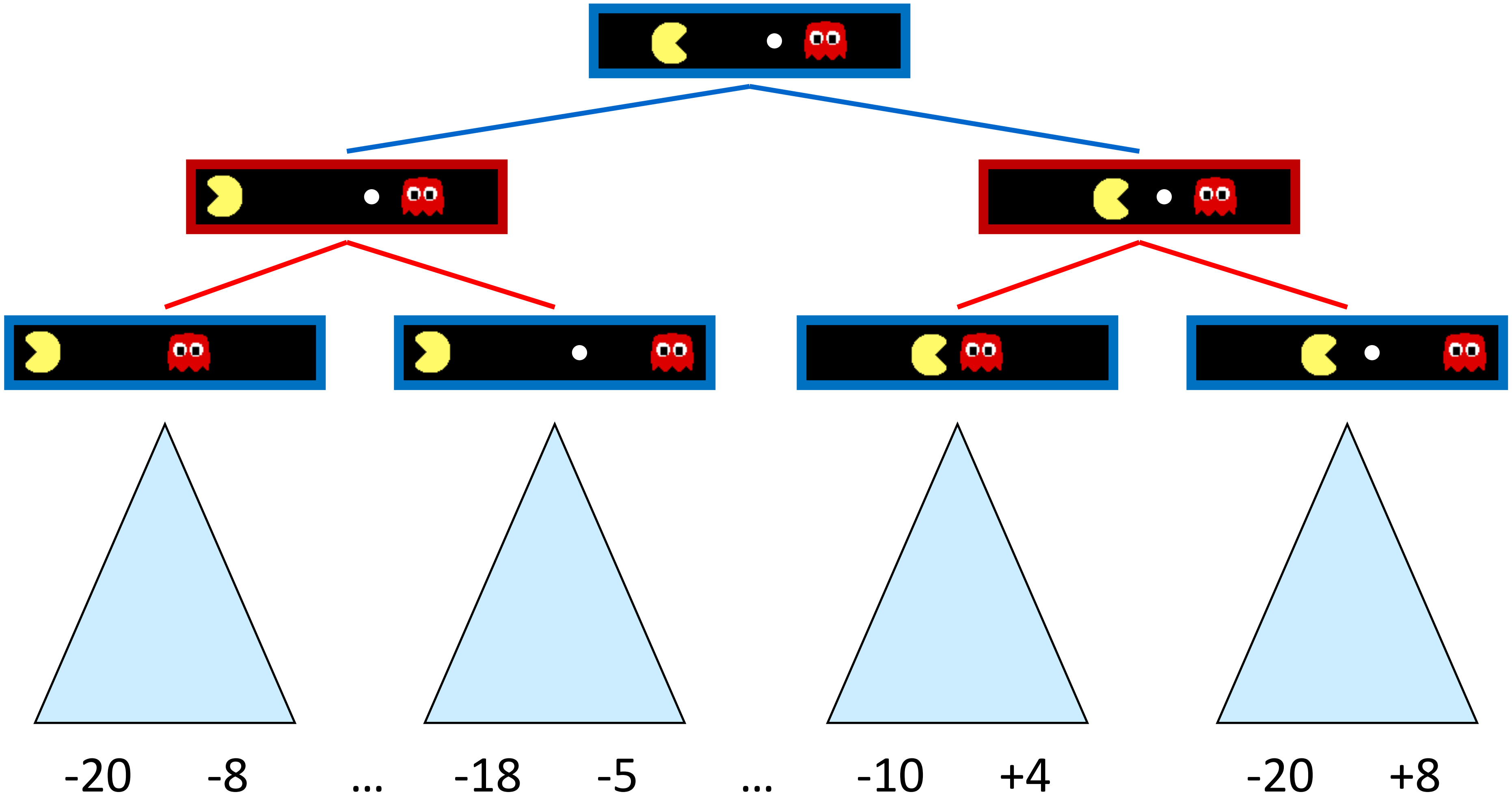


Figure from Berkley AI

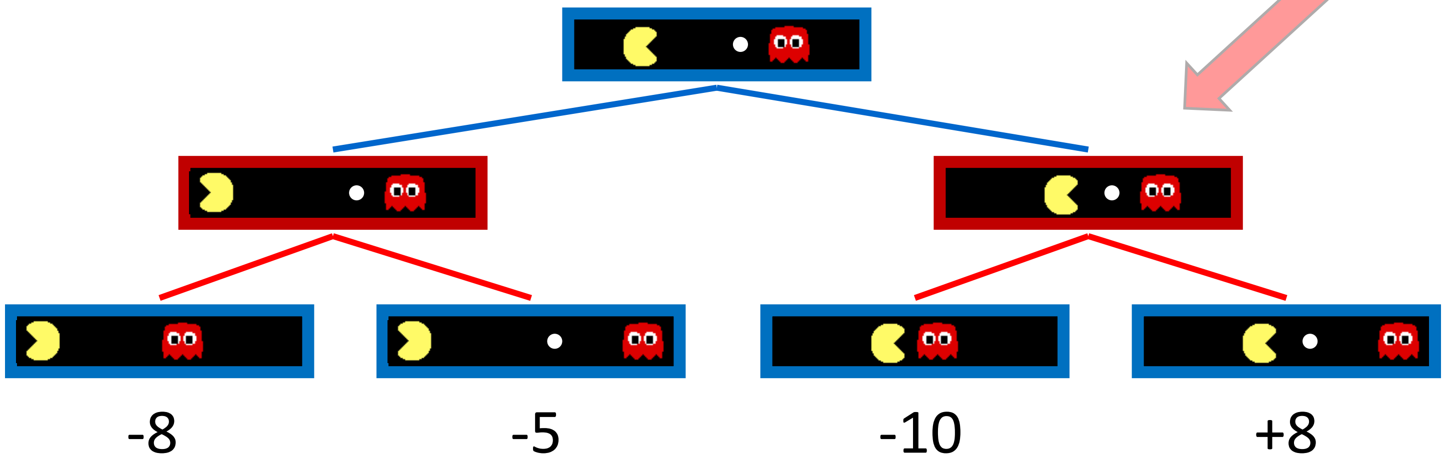
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

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



Terminal States:

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

Figure from Berkley AI

Tic-Tac-Toe Game Tree

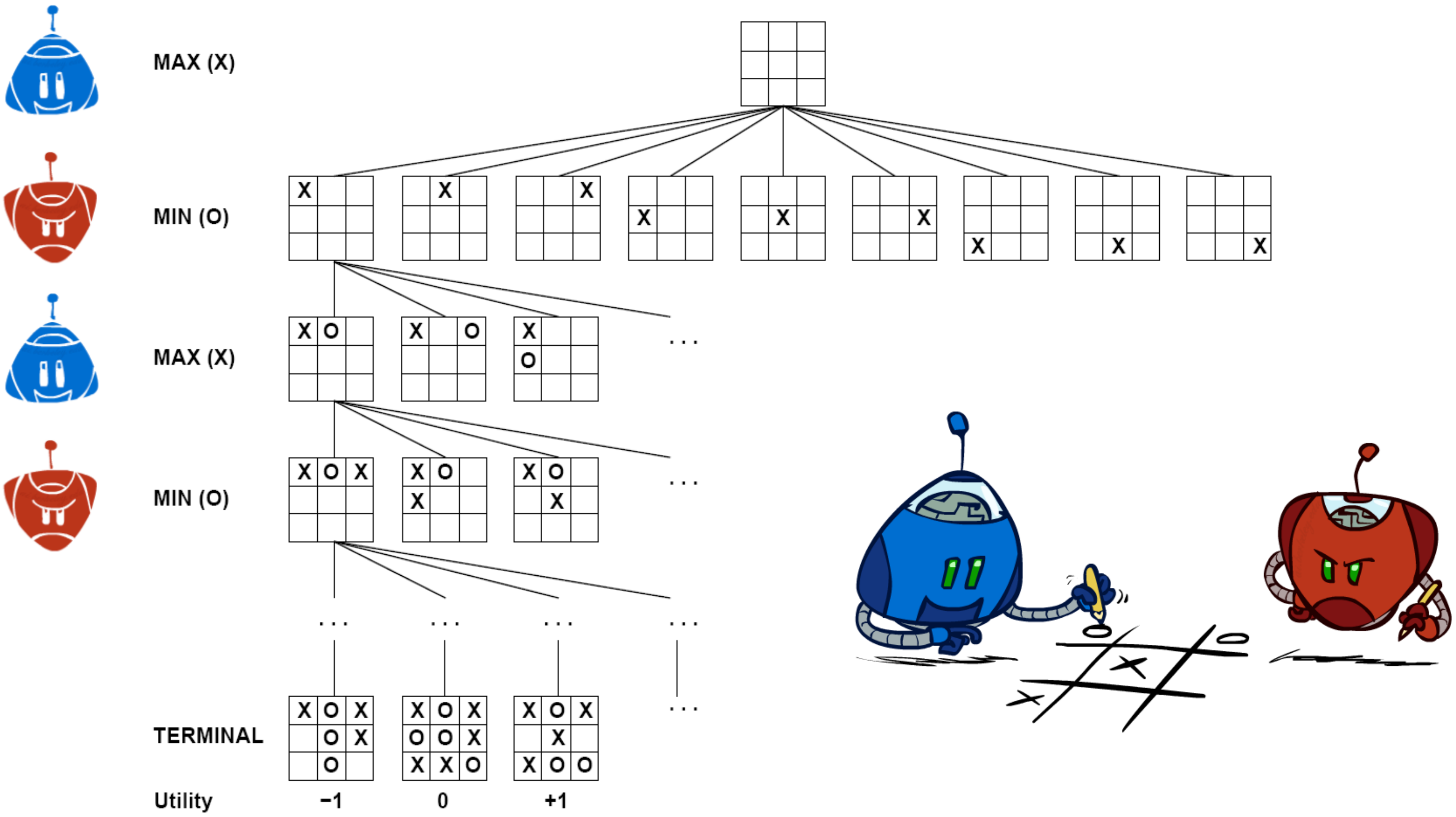


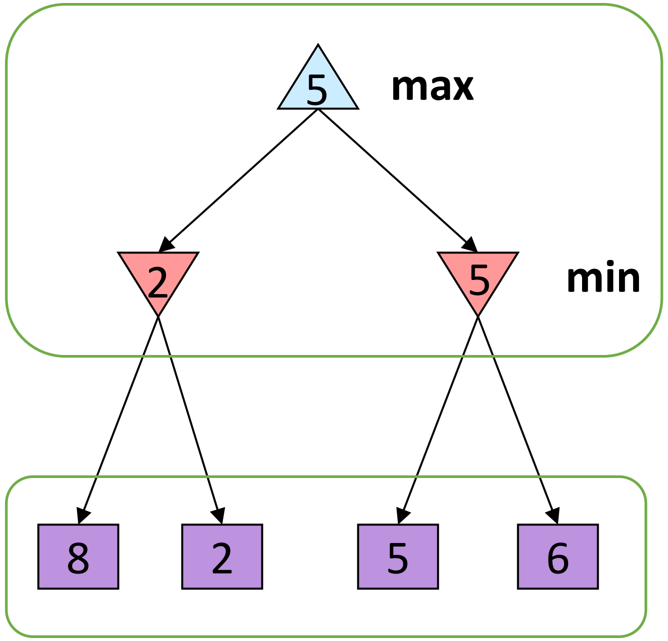
Figure from Berkley AI

Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result

- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node’s minimax value: the best achievable utility against a rational (optimal) adversary

Minimax values: computed recursively



Terminal values: part of the game

Minimax Implementation

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

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

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

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

Figure from Berkley AI

Minimax Implementation (Dispatch)

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

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

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


Figure from Berkley AI

Minimax Example

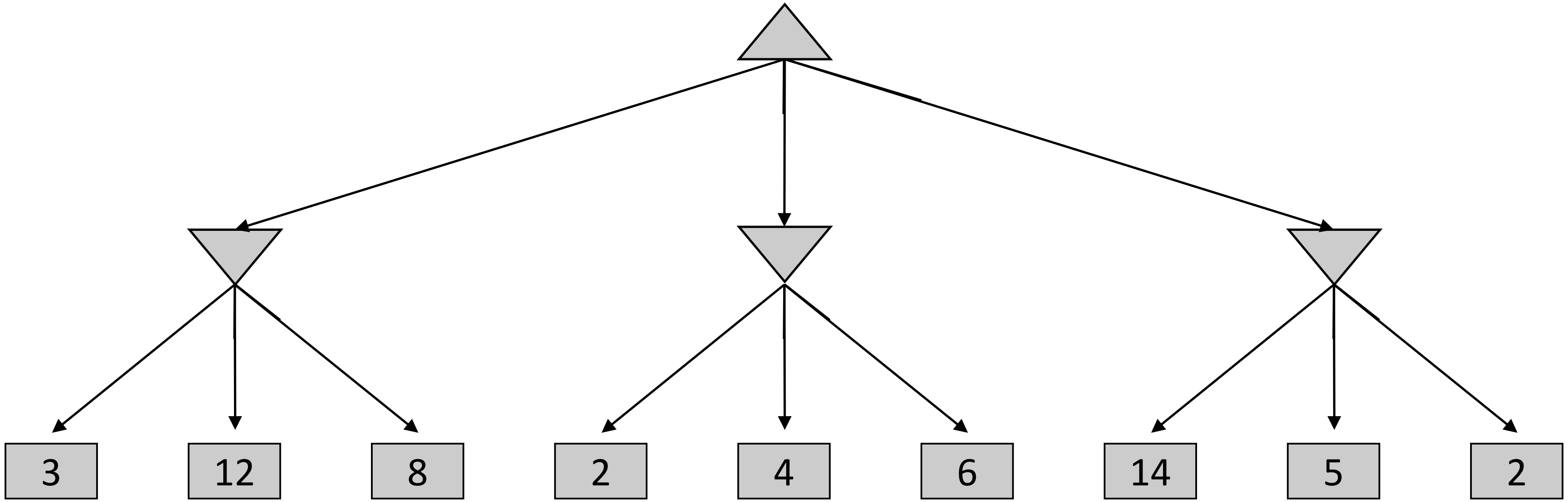


Figure from Berkley AI

Minimax Efficiency

- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

- Example: Chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - Do we need to explore the entire tree?

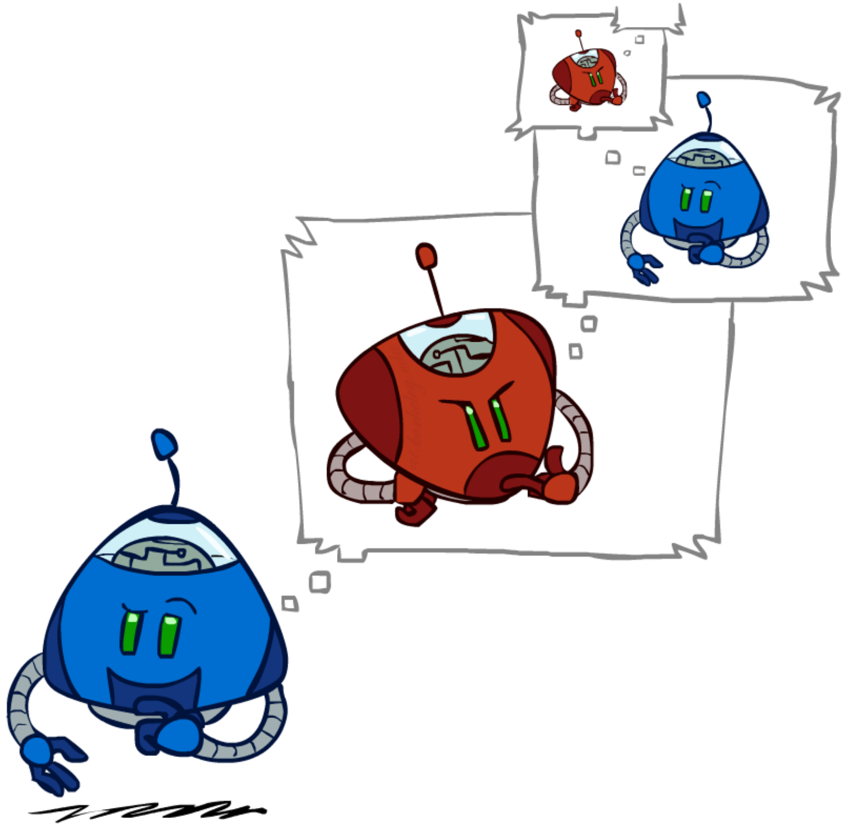
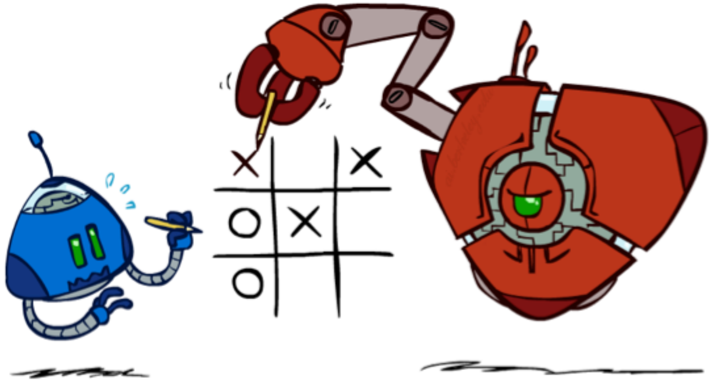
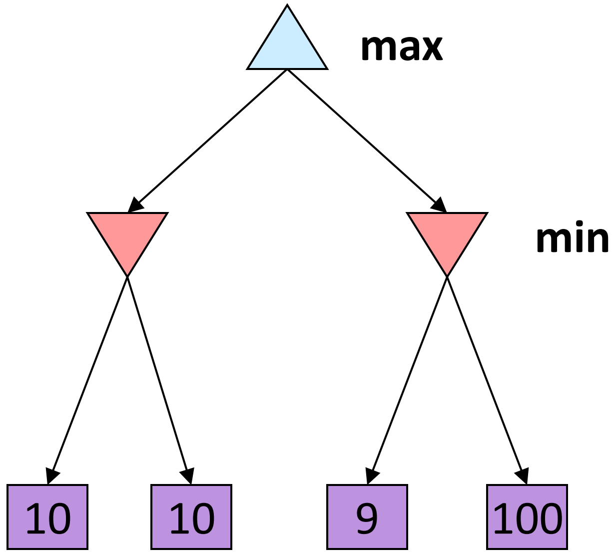
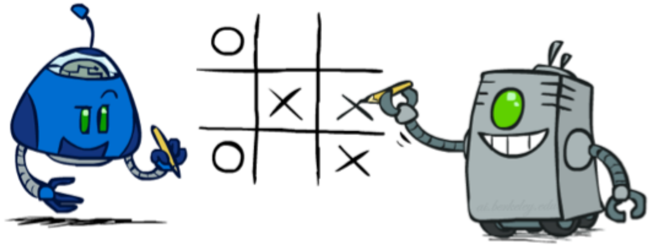


Figure from Berkley AI

Minimax Properties







Optimal against a perfect player. Otherwise?

Figure from Berkley AI

Video of Demo Min Vs Exp (Min)

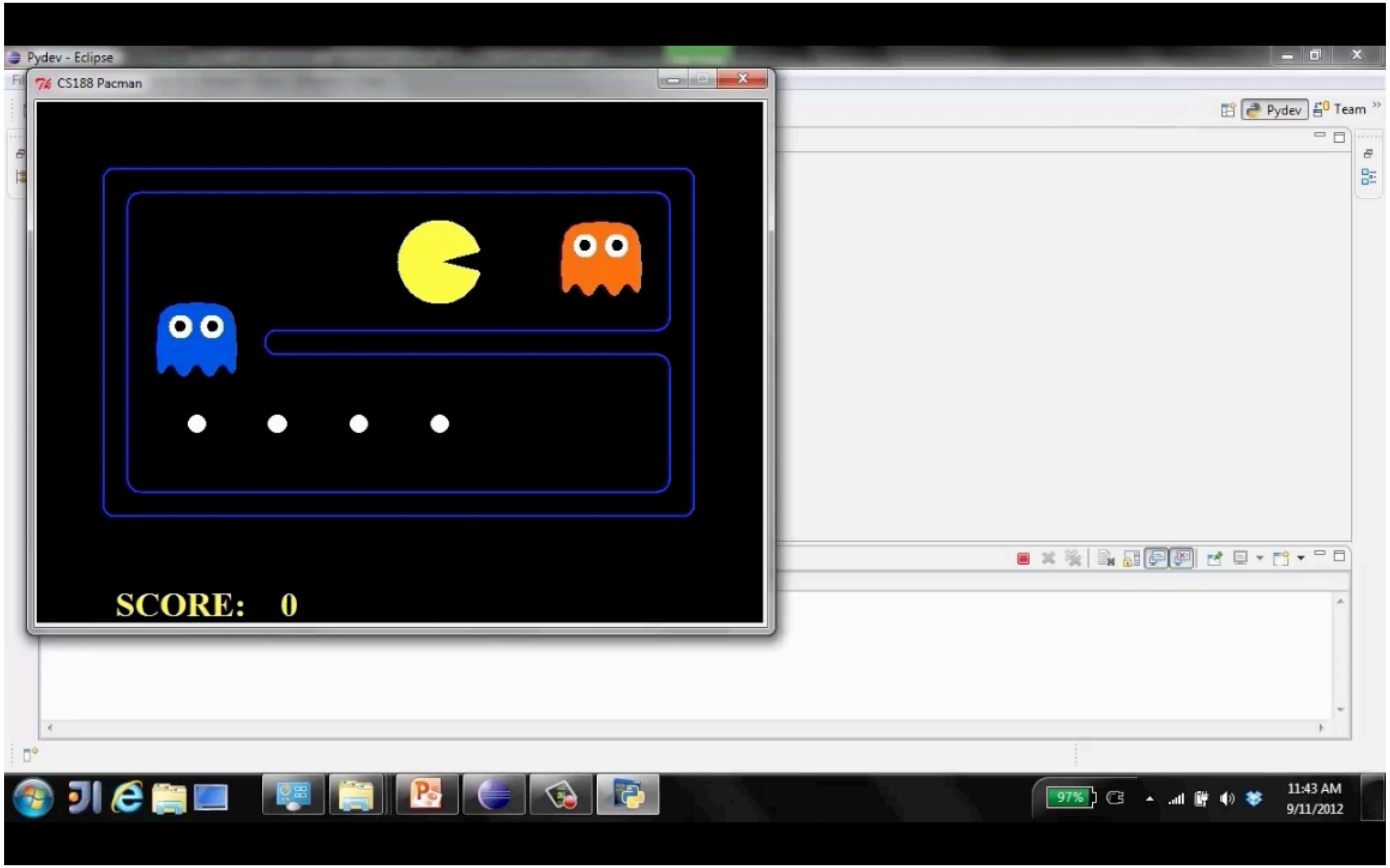
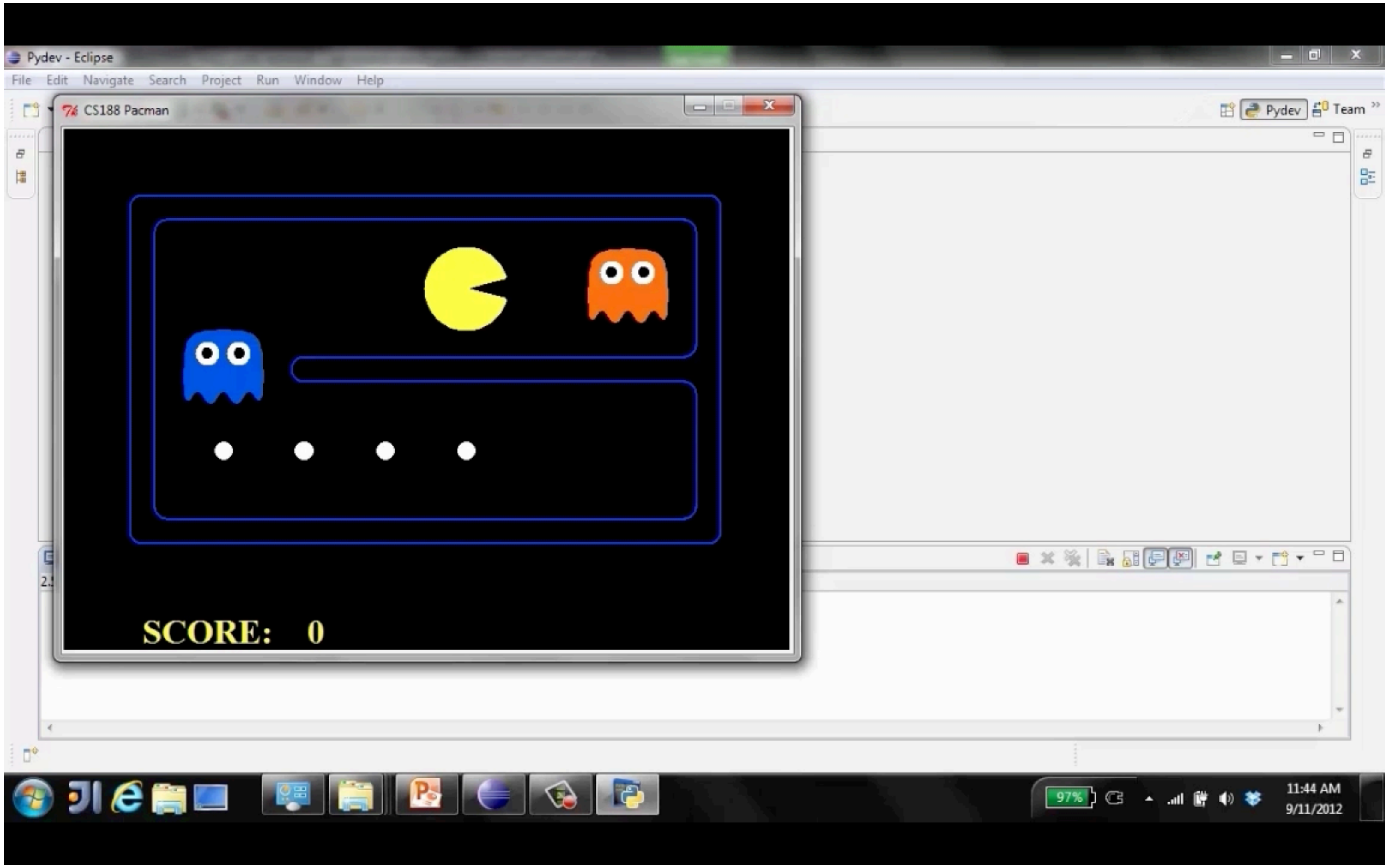


Figure from Berkley AI

Video of Demo Min vs Exp (Exp)



Resource Limits

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

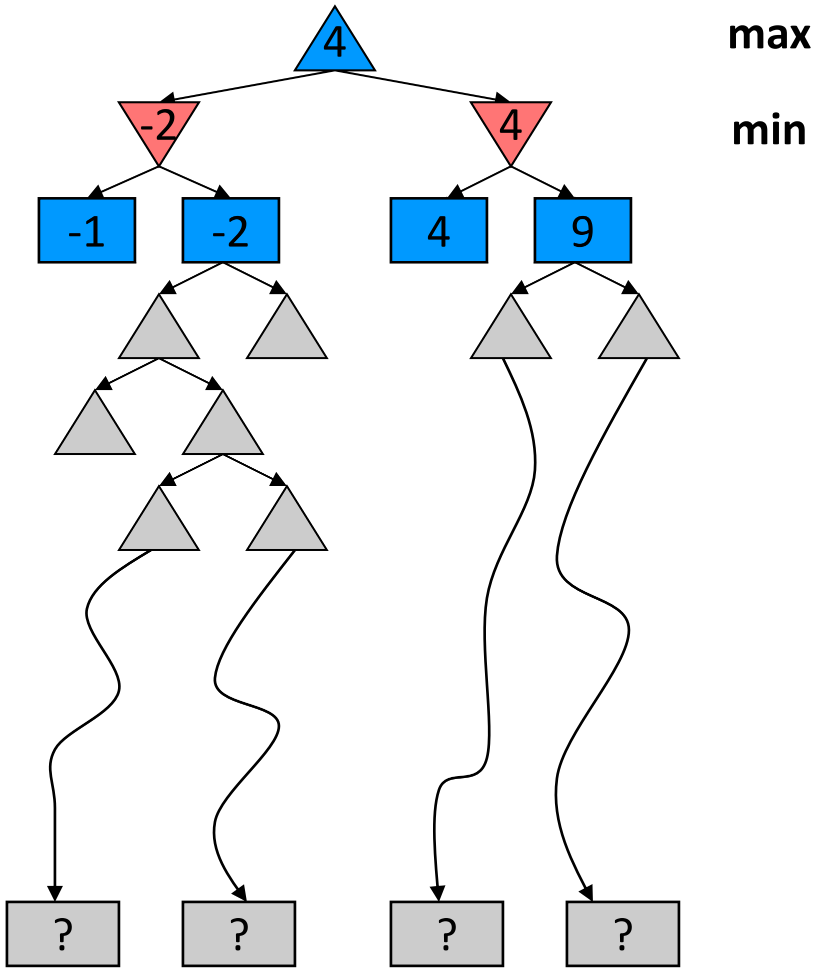
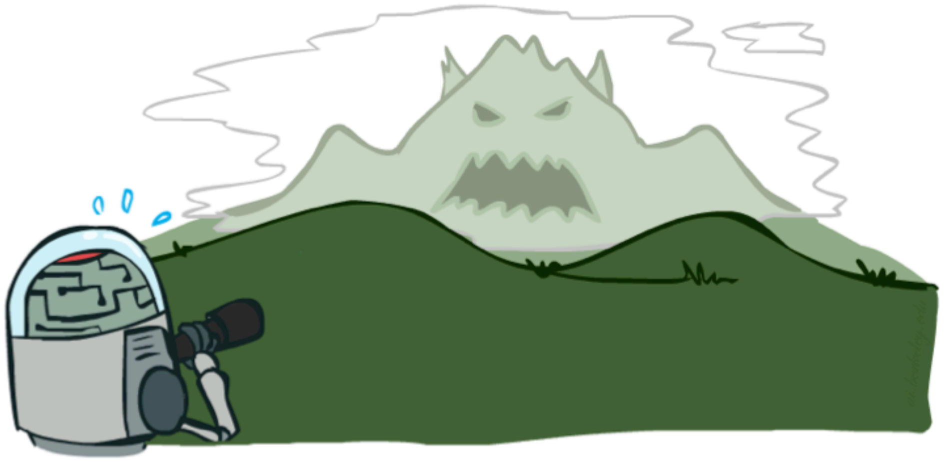


Figure from Berkley AI

Search Depth Matters

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



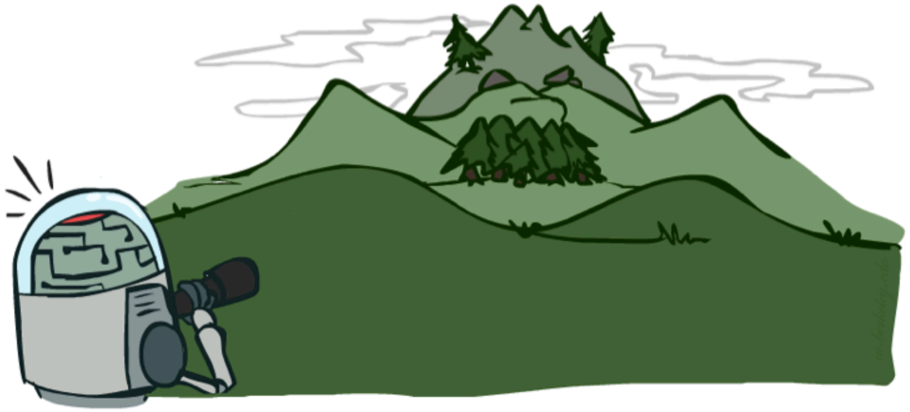


Figure from Berkley AI

Demo Limited Depth (2)

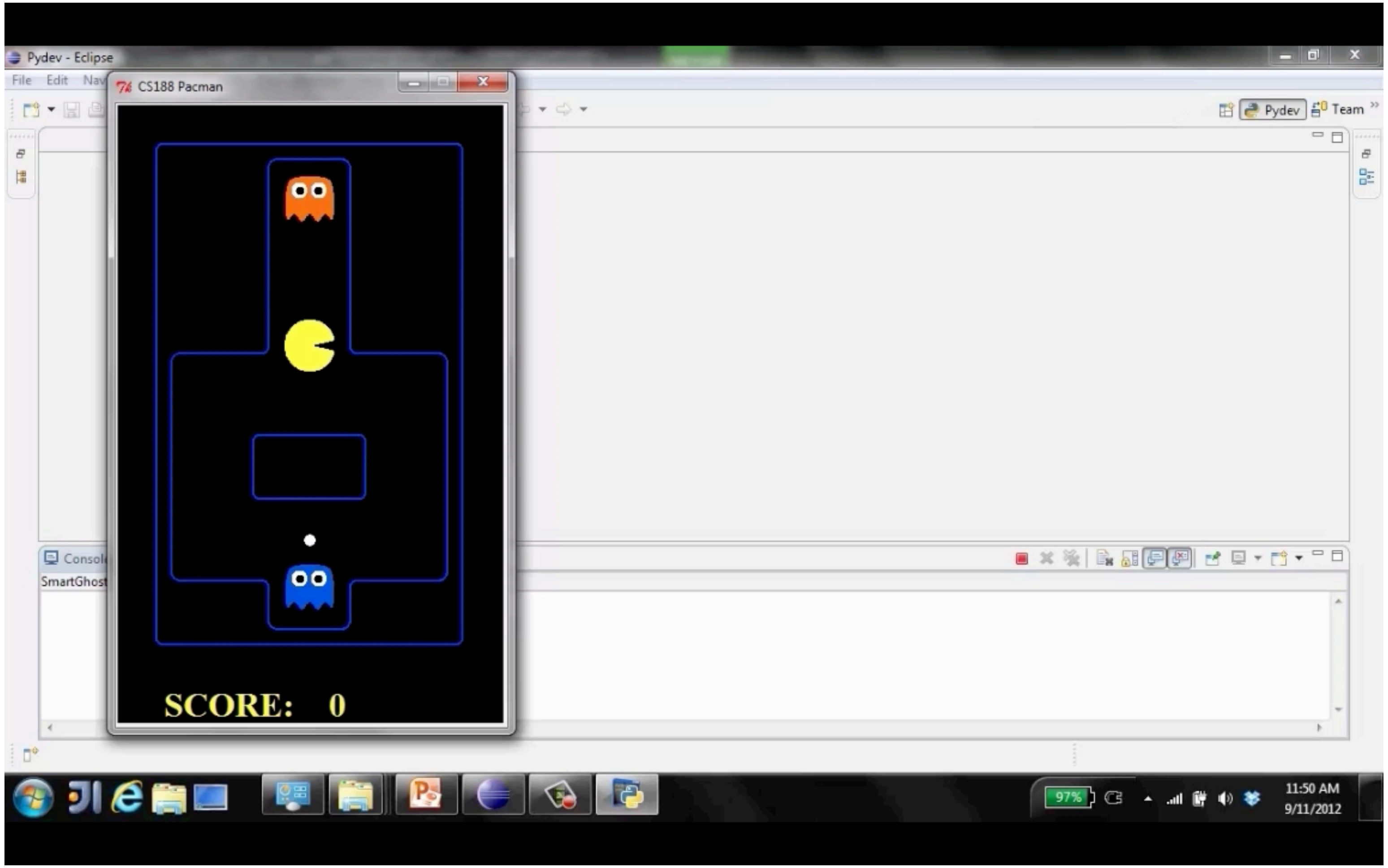


Figure from Berkley AI

Demo – Depth Limited (10)

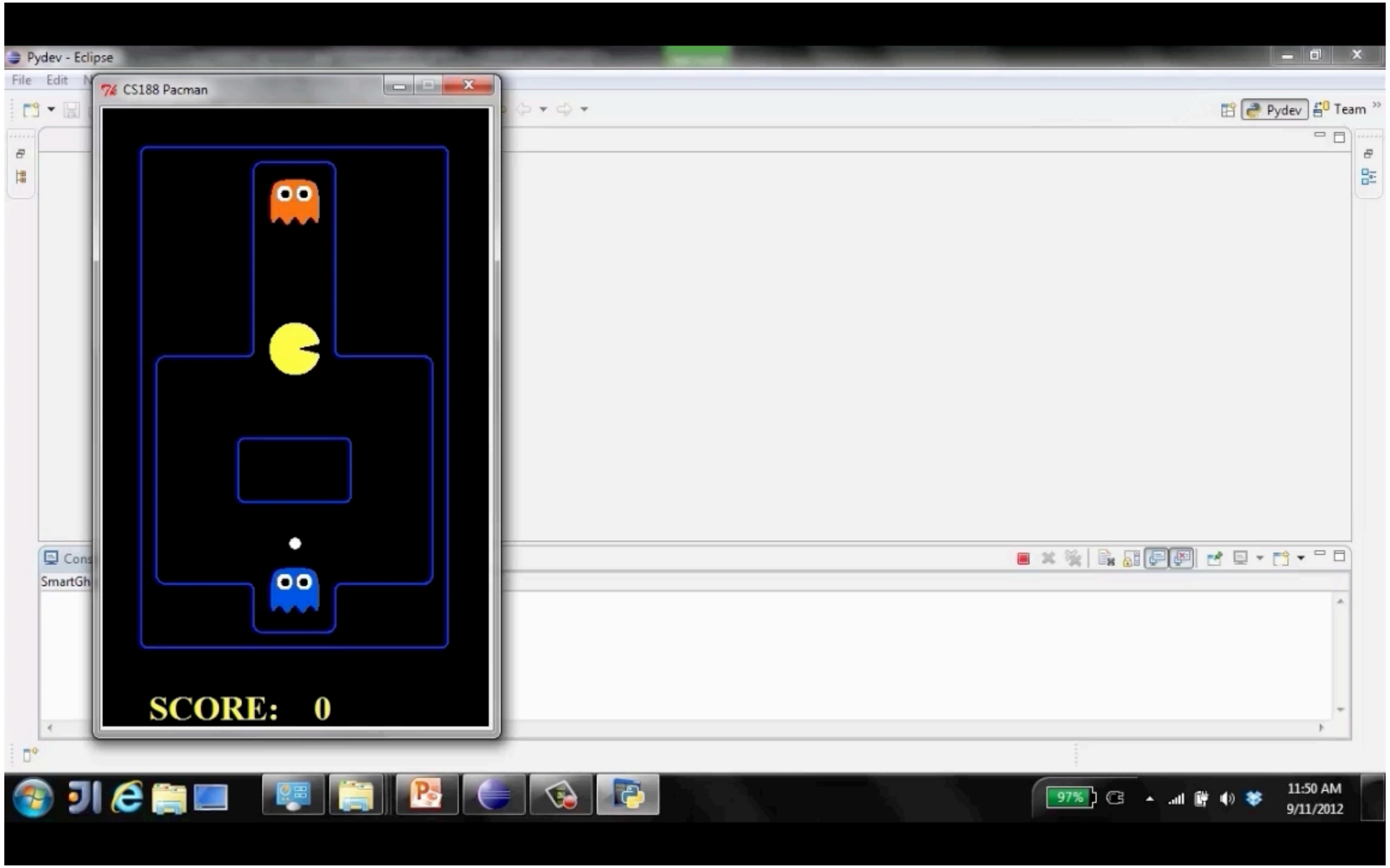
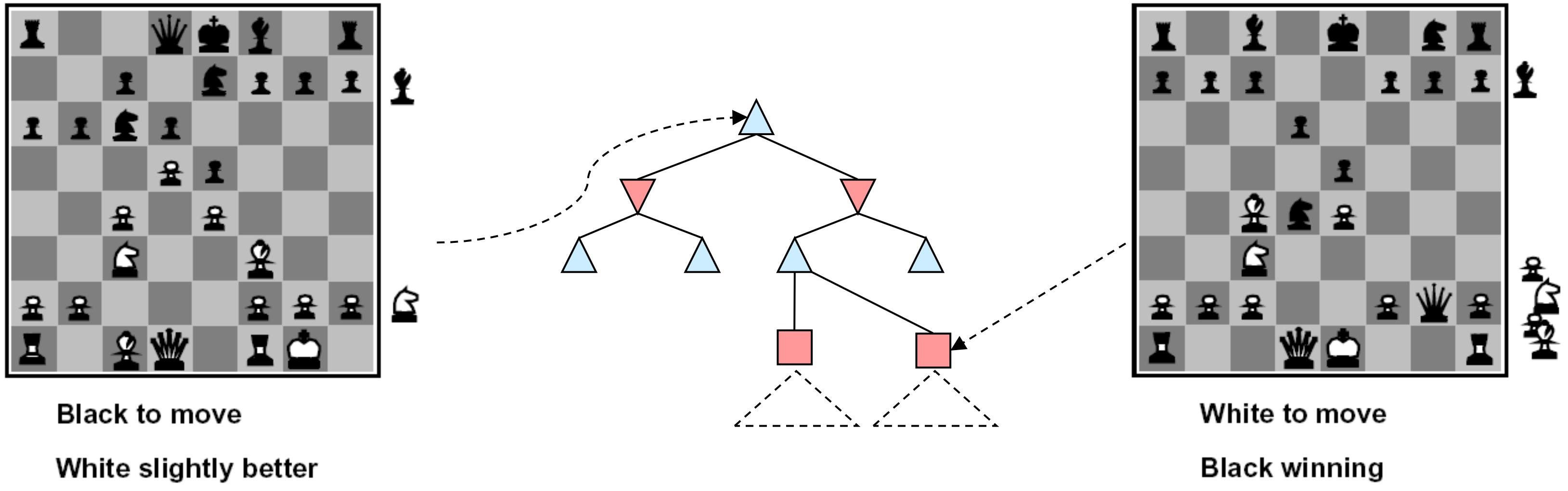


Figure from Berkley AI

Evaluation Functions

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



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

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

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

Evaluation for Pacman – Thrashing (d=2)

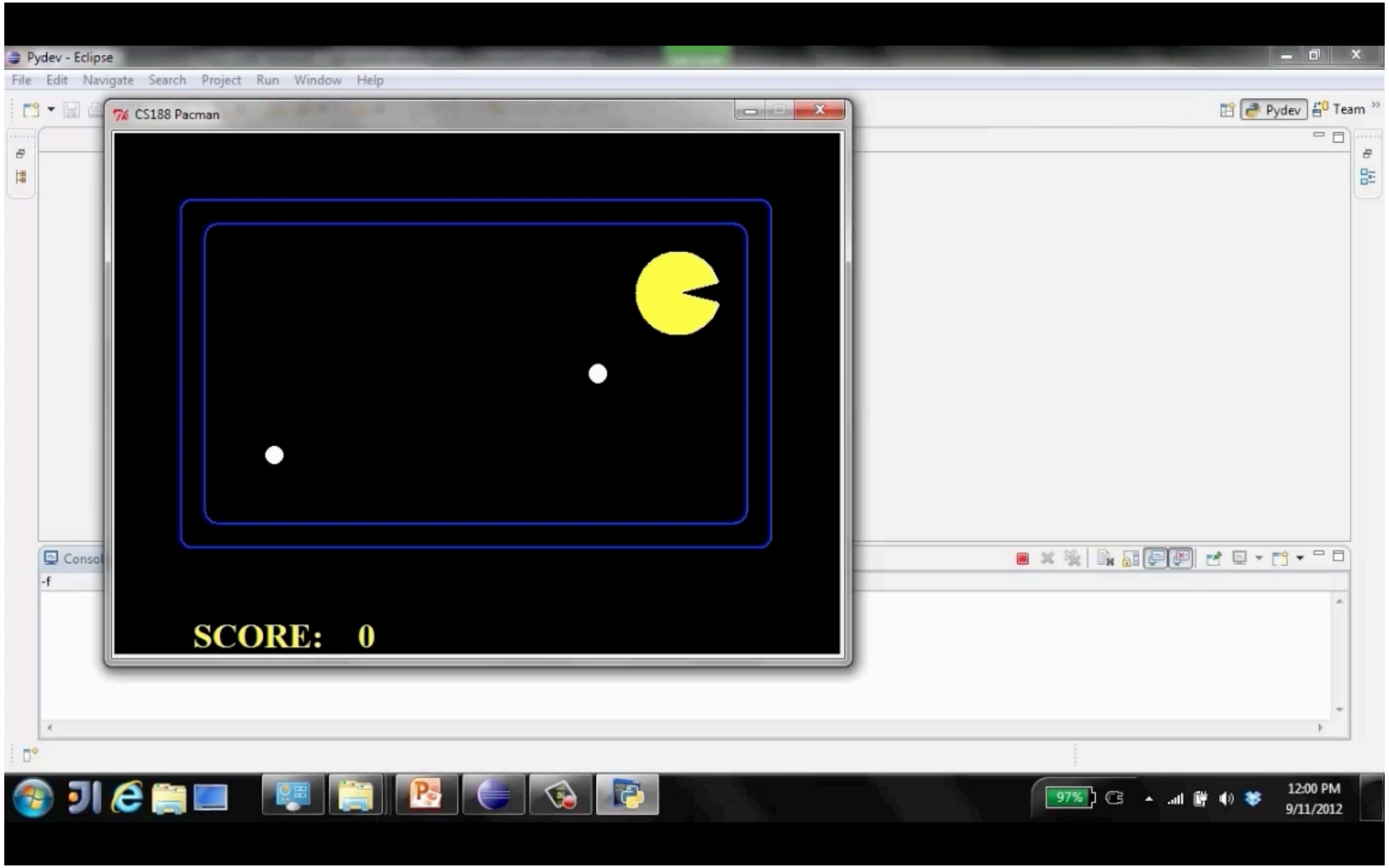
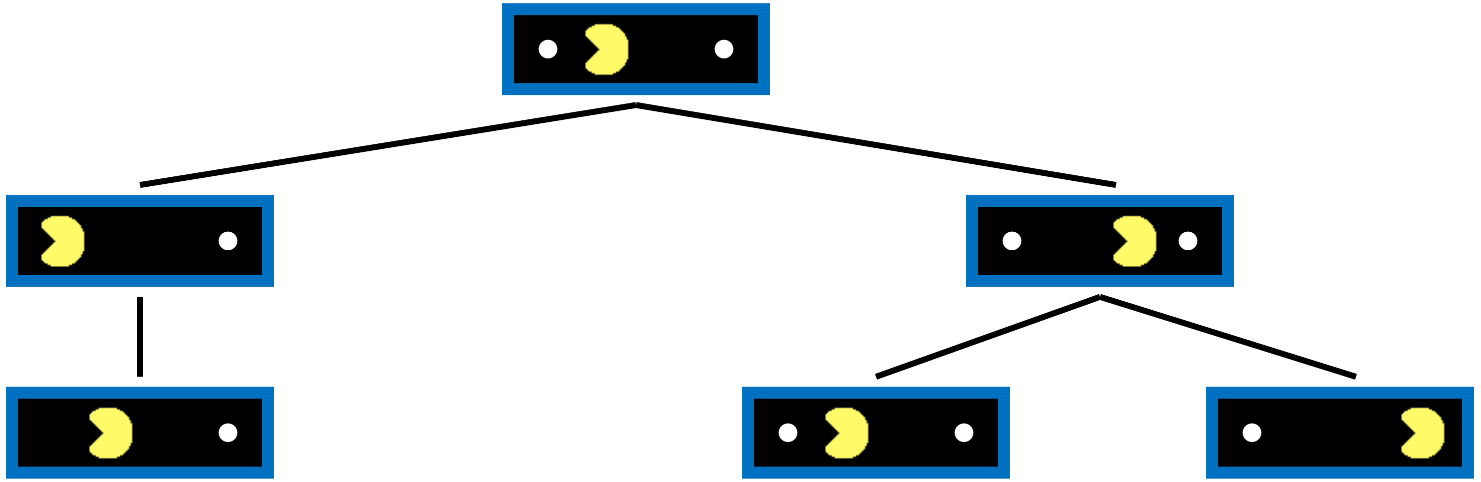


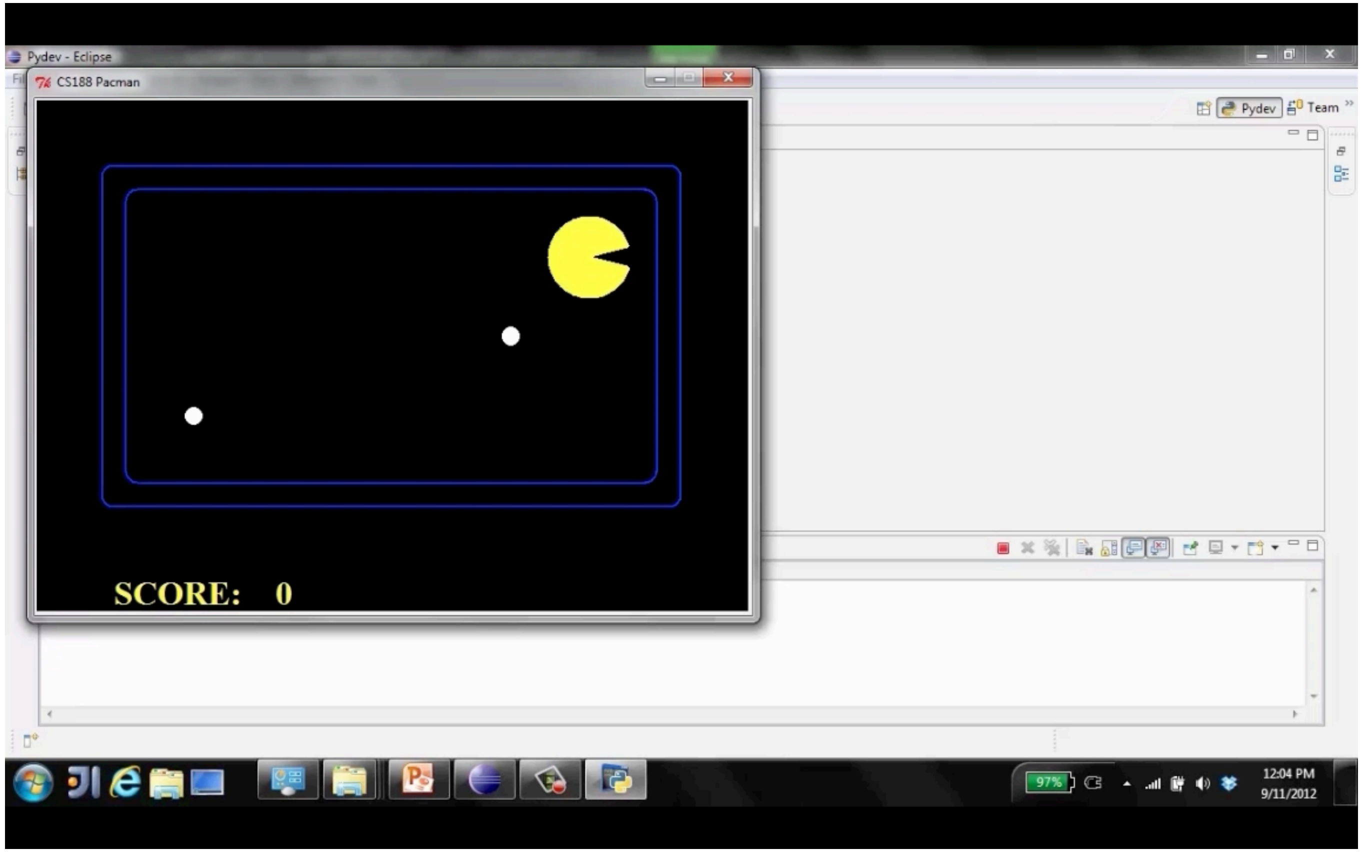
Figure from Berkley AI

Why does Pacman Thrash and Starve



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

Video – Thrashing Resolved – Fixed (d=2)



Evaluation function includes proximity to the nearest food pellet.

Figure from Berkley AI

Game Tree Pruning

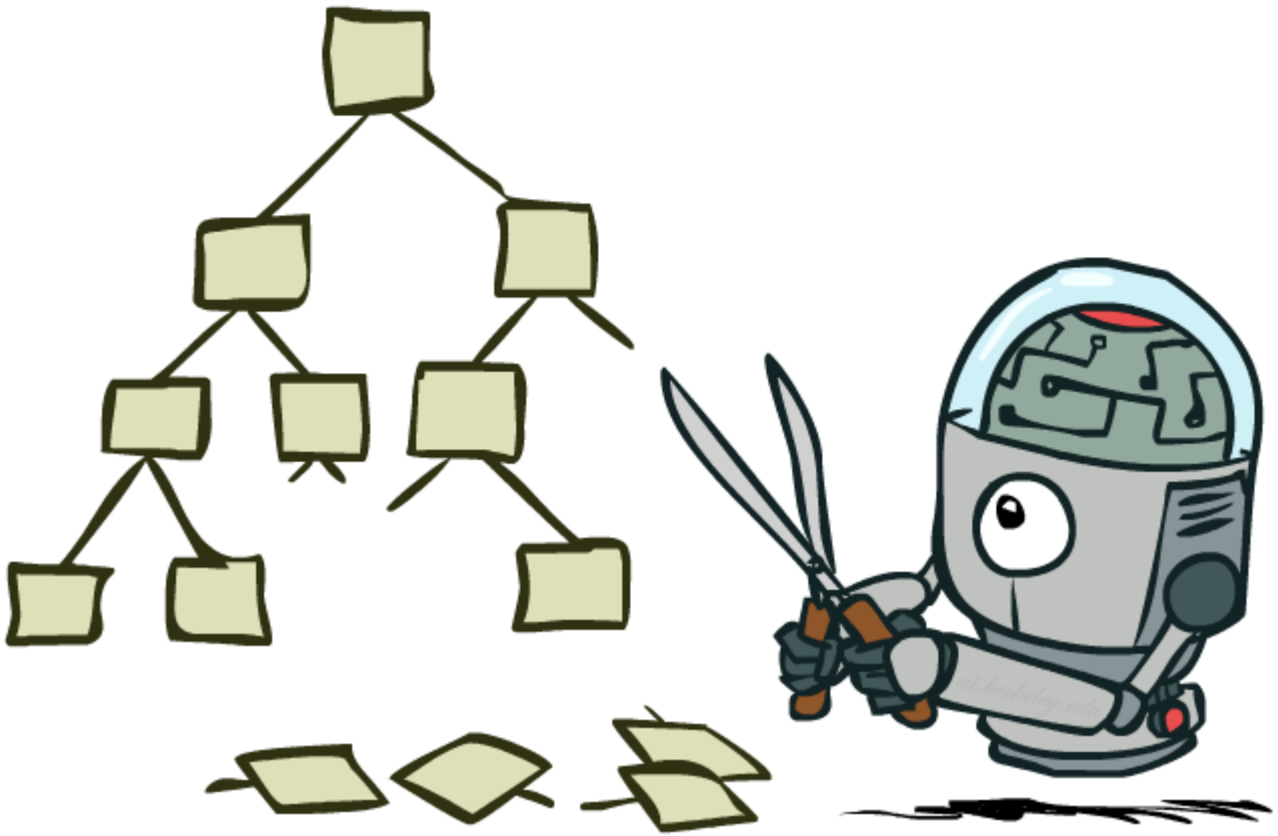


Figure from Berkley AI

Game Tree Pruning

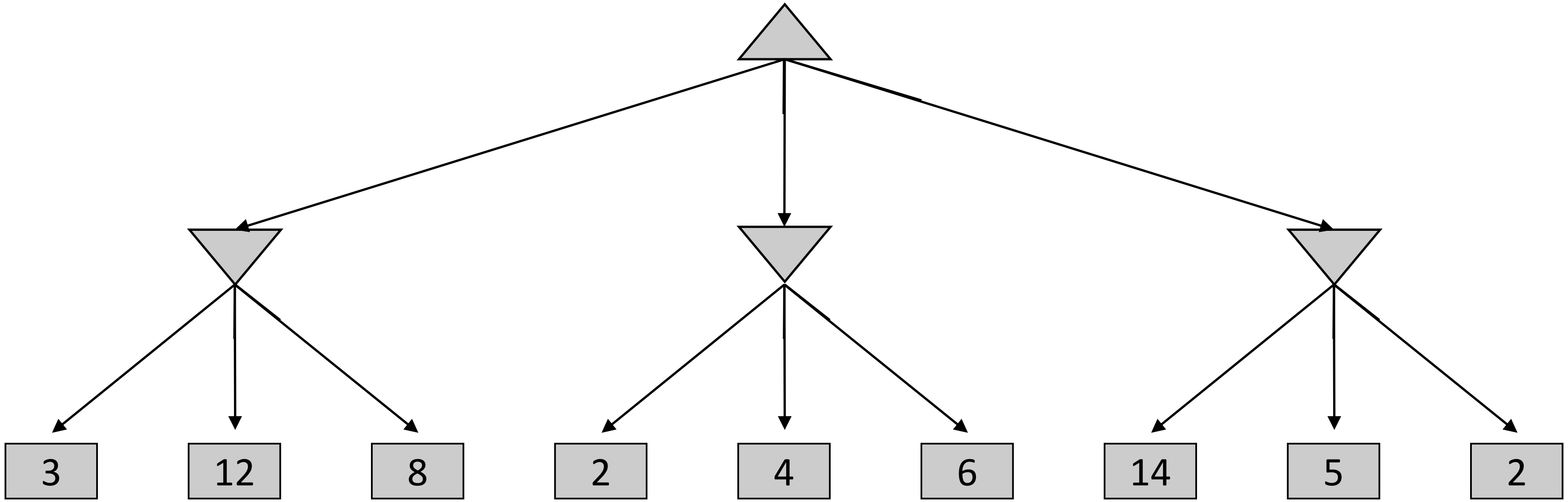


Figure from Berkley AI

Alpha-Beta Pruning

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

- MAX version is symmetric

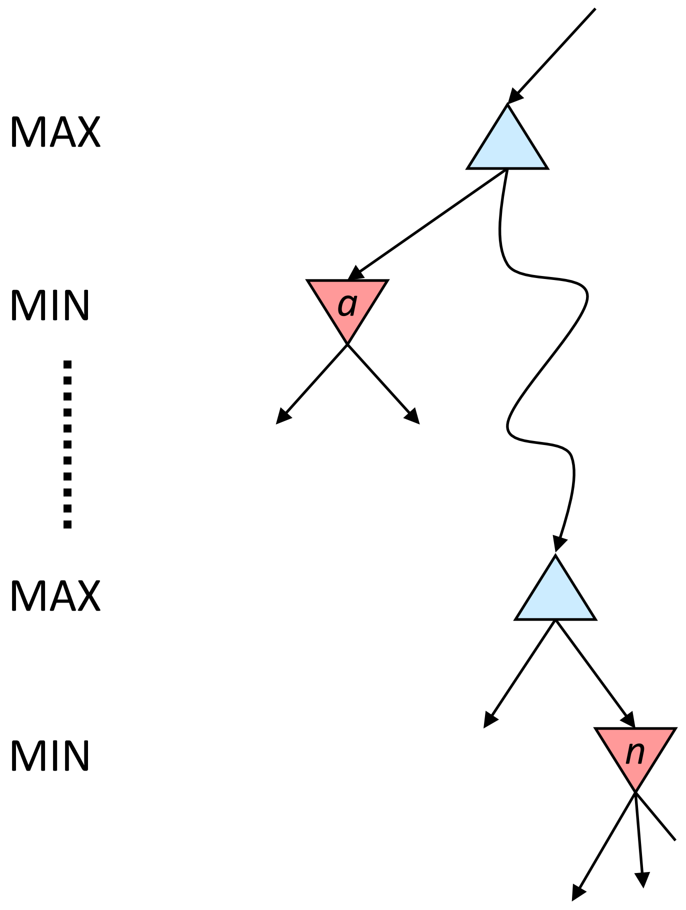


Figure from Berkley AI

Alpha-Beta Pruning

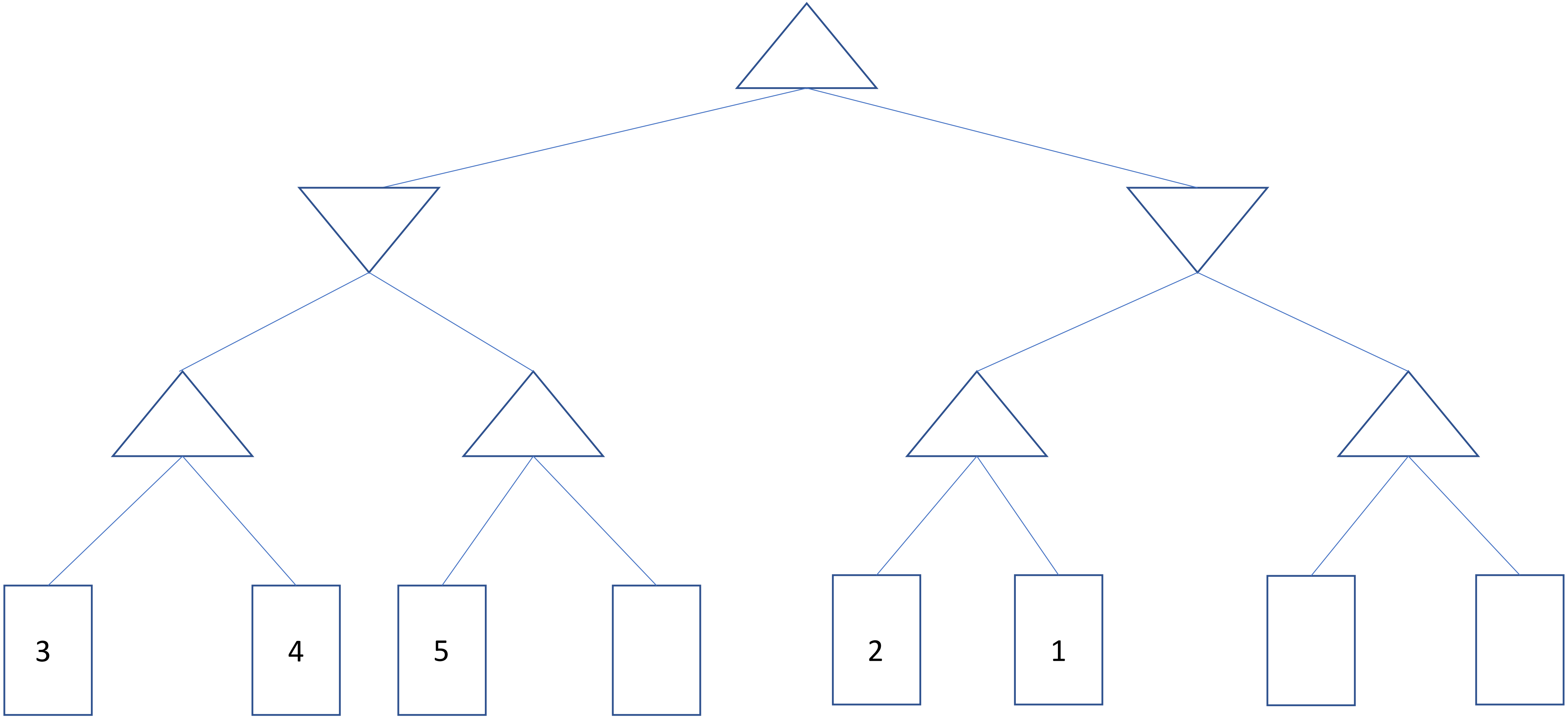


Figure from Berkley AI

Alpha-Beta Pruning Properties

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

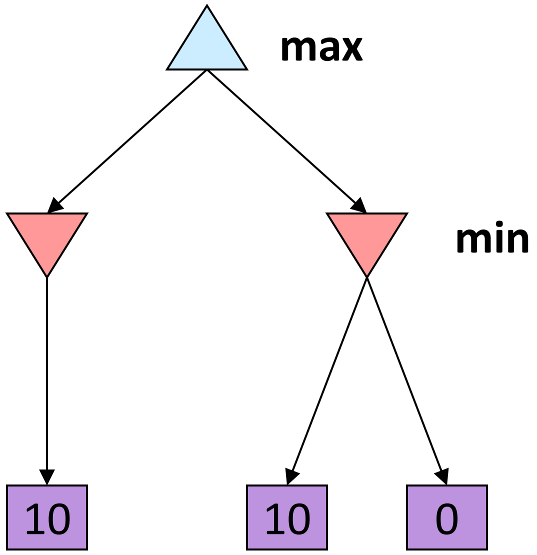


Figure from Berkley AI

Alpha Beta Quiz

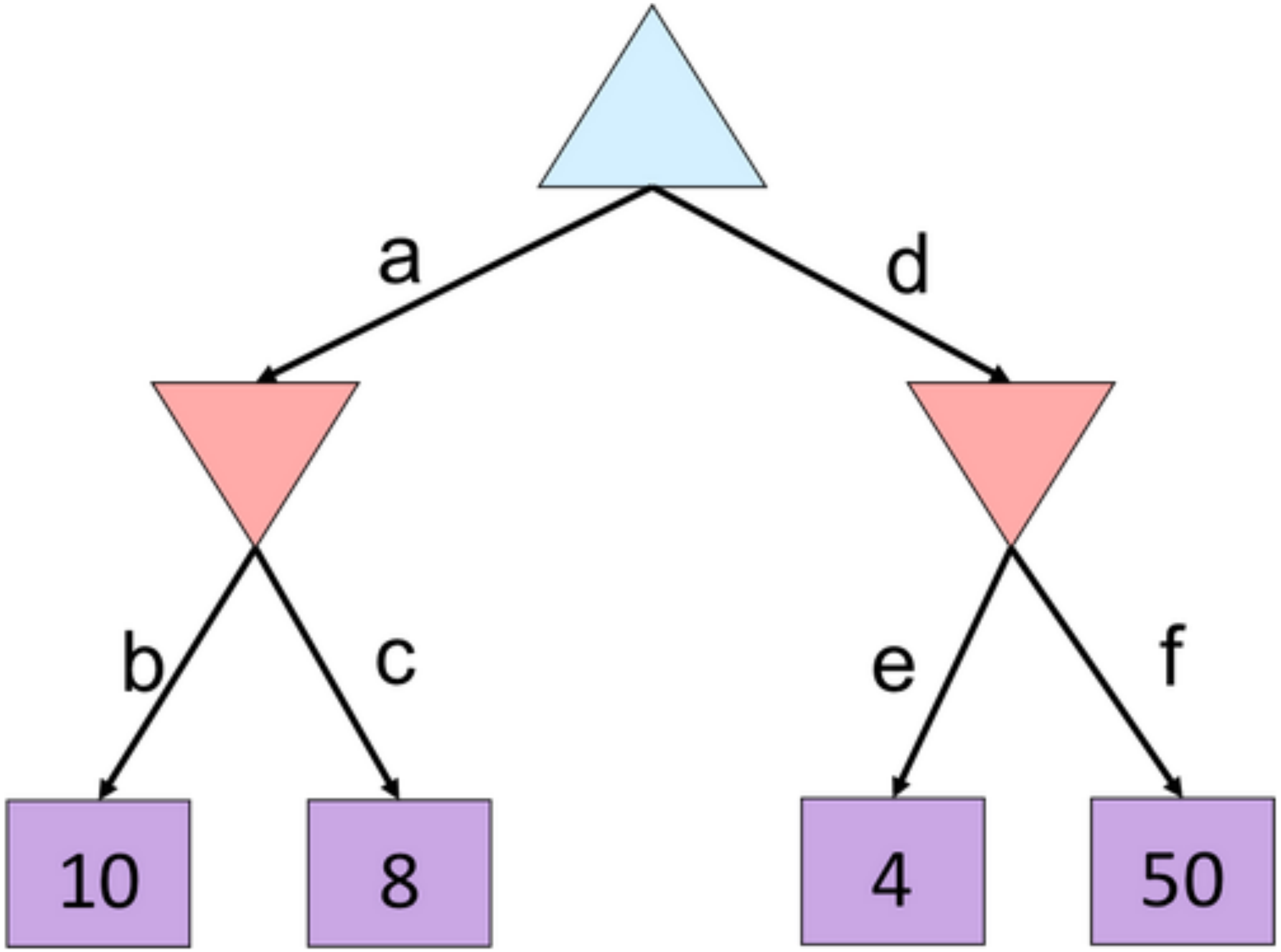


Figure from Berkley AI