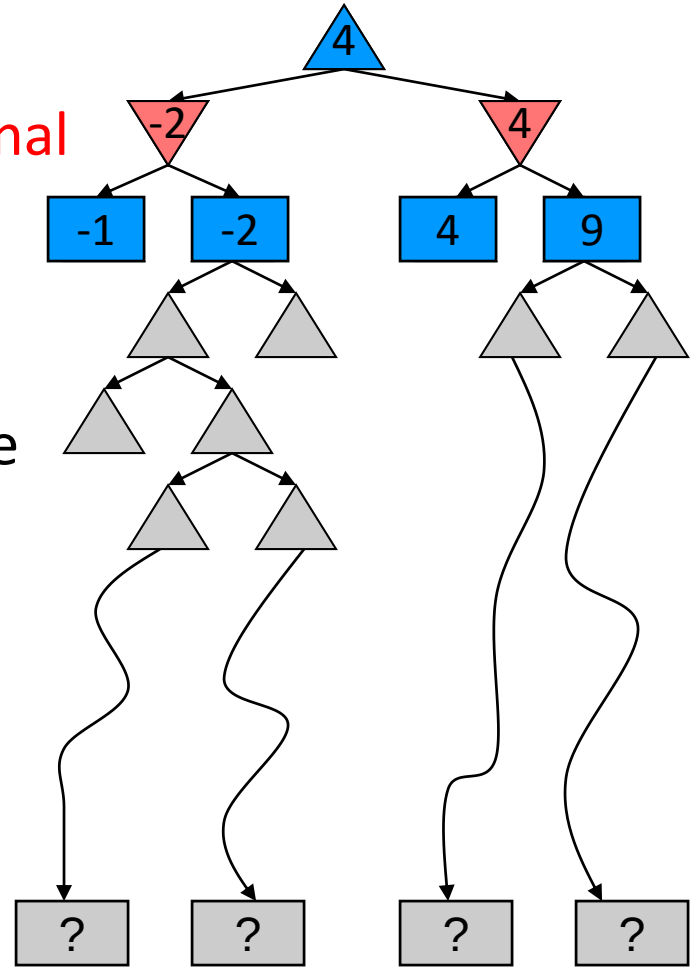
Evaluation Functions

Resource Limits

- The minimax algorithm generates the entire game search space, whereas the alpha–beta algorithm allows us to prune large parts of it.

- However, alpha–beta still has to search all the way to terminal states for at least a portion of the search space

- we turn to evaluation functions, functions that take in a state and output an estimate of the true minimax value of that node



Resource Limits

- Evaluation functions are widely employed in depth-limited minimax.

- where we treat non-terminal nodes located at our maximum solvable depth as terminal nodes.

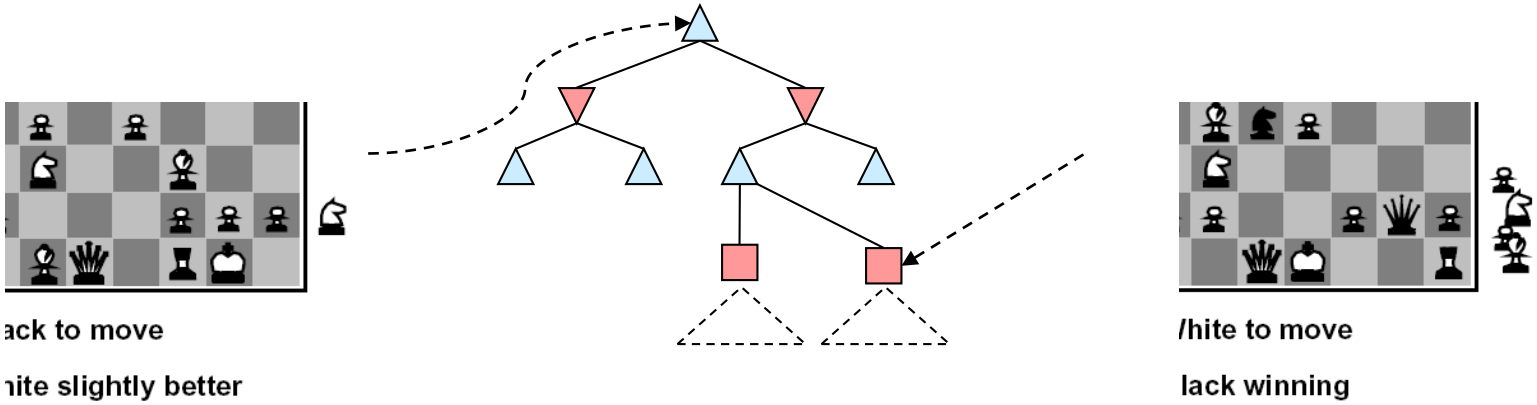
- Assigning mock terminal utilities as determined by a carefully selected evaluation function.

- Replace terminal utilities with an evaluation function fo non-terminal positions (Similar to heuristics in Informed search)

- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - β-αreaches about depth 8 – decent chess program
  - what is depth 8 means in chess ?
- evaluation functions can only yield estimates of the values of non-terminal utilities .Guarantee of optimal play is gone
- More depth makes a BIG difference

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:

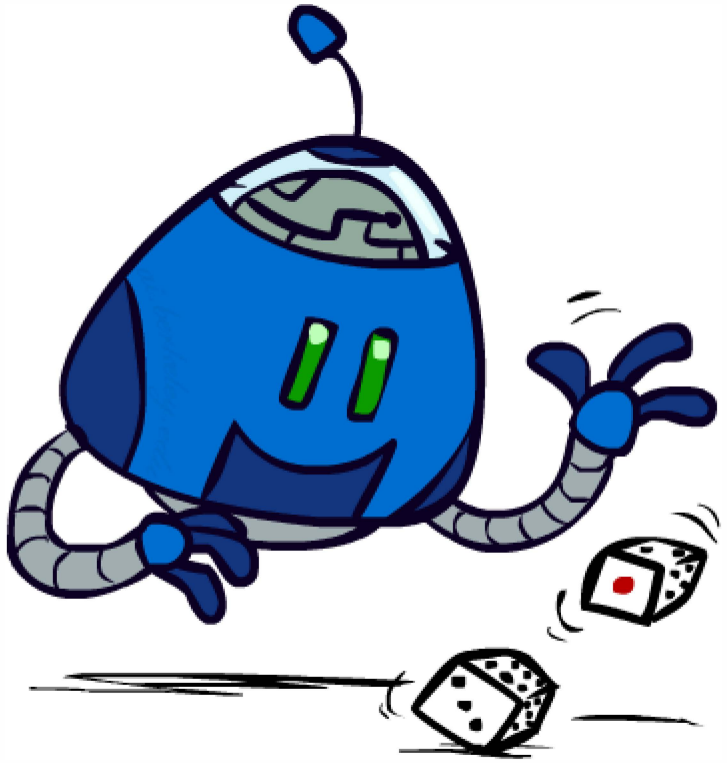
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

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

Depth Matters

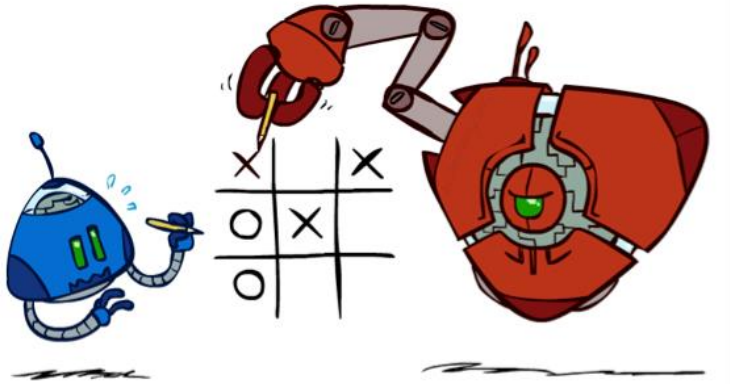
- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation

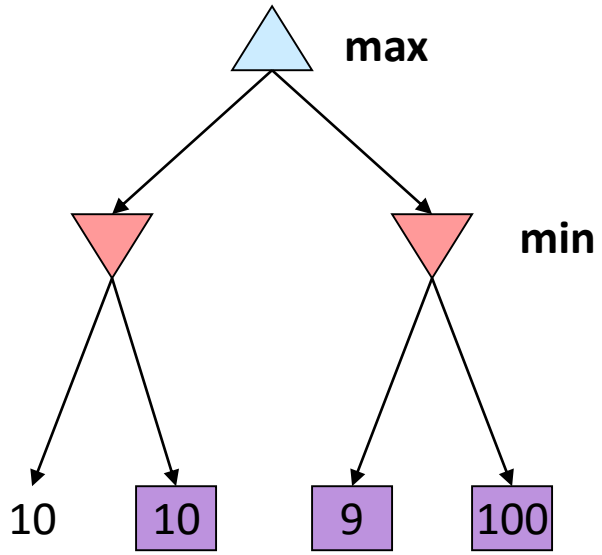
Expectimax Search

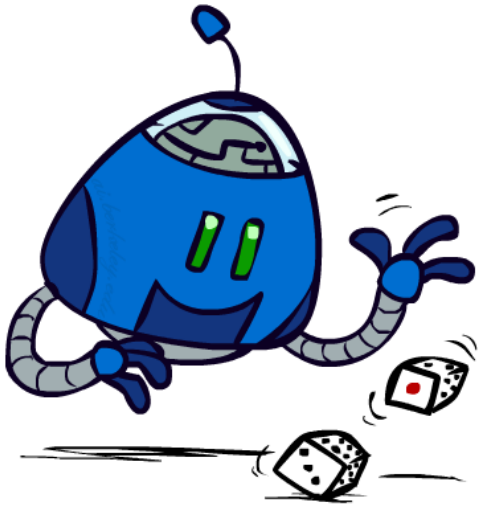


UNCERTAIN OUTCOMES

Worst-Case vs. Average Case





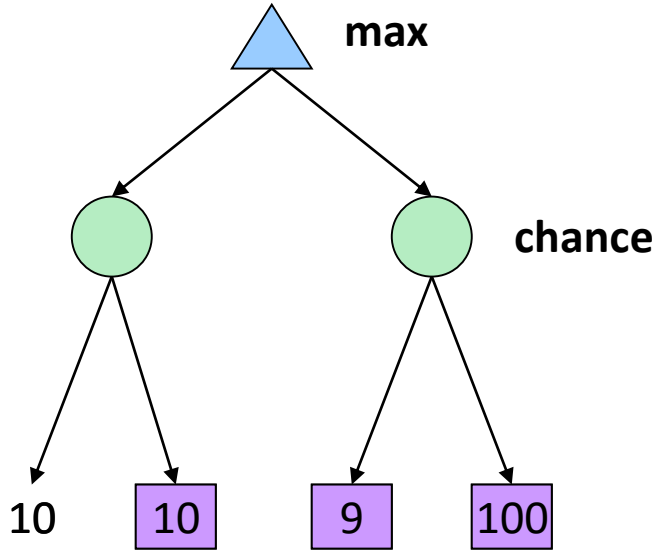


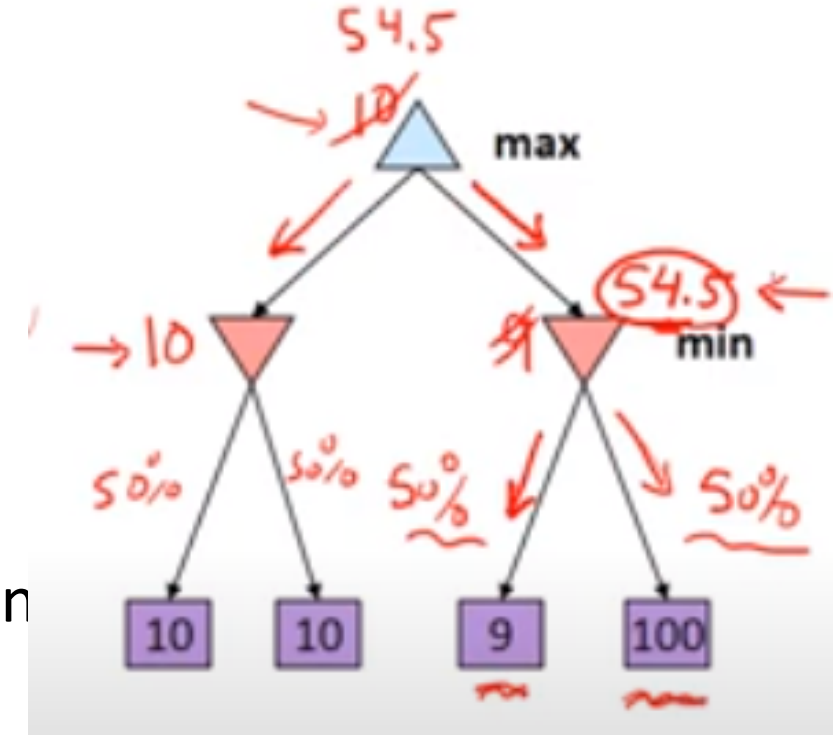
Idea: Uncertain outcomes controlled by chance, not an adversary!

In this example we will never get 100. we say never because we are assuming playing against optimal player. (Worst case)

Expectimax Search

- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip
- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes
- Expectimax search: compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
    - i.e., take weighted average (expectation) of children





Expectimax Pseudocode

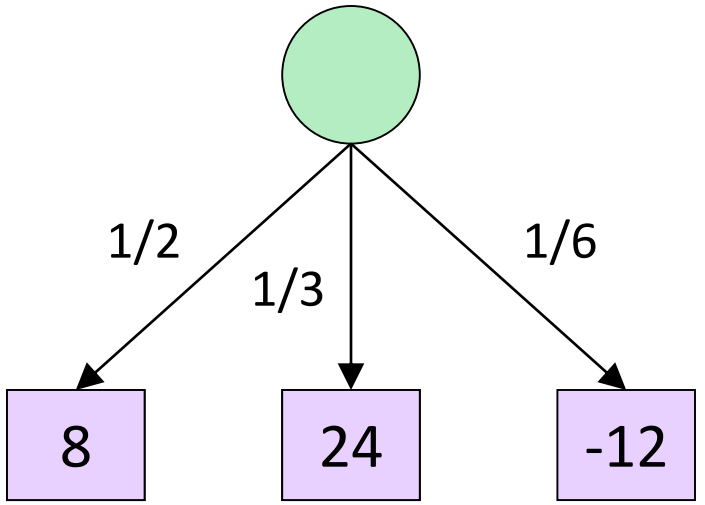
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

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

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

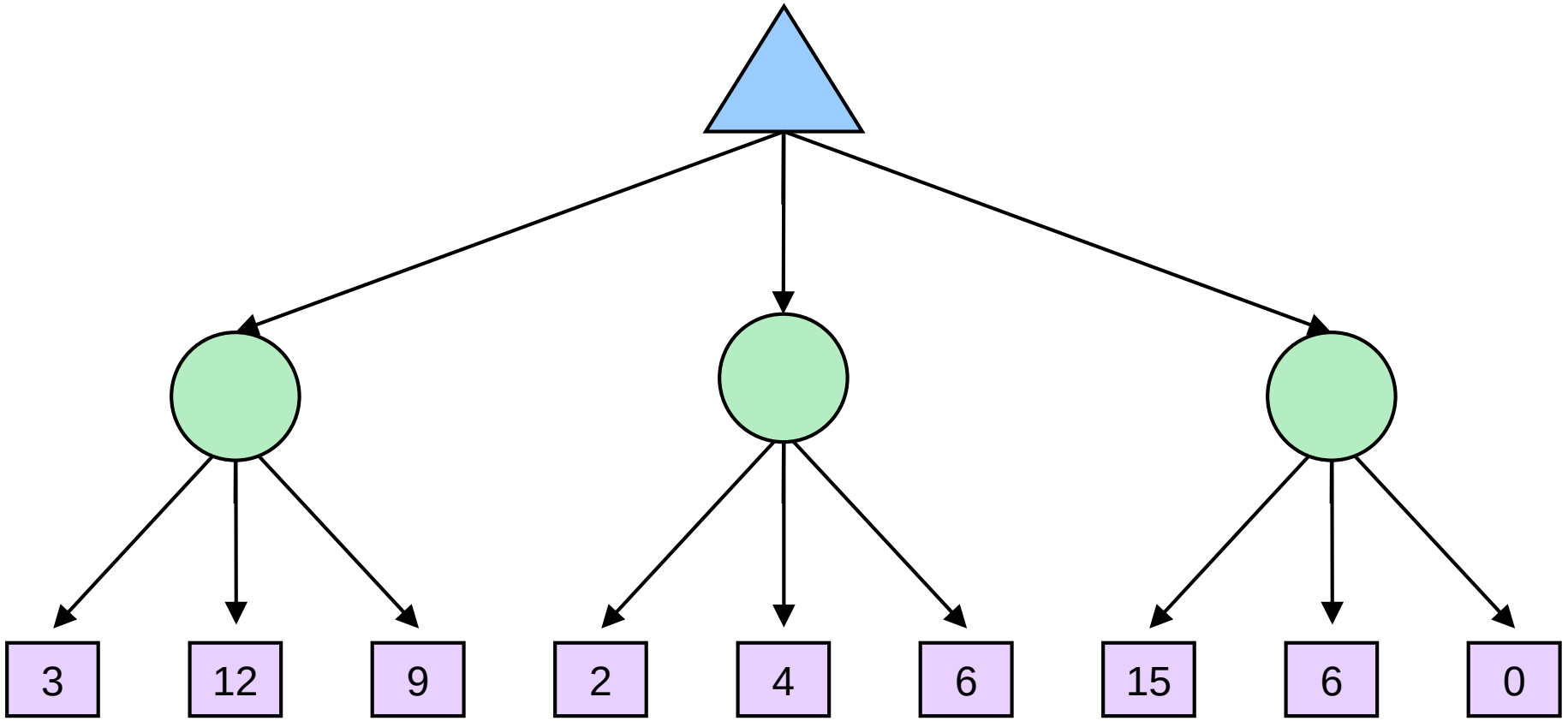
Expectimax Pseudocode

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

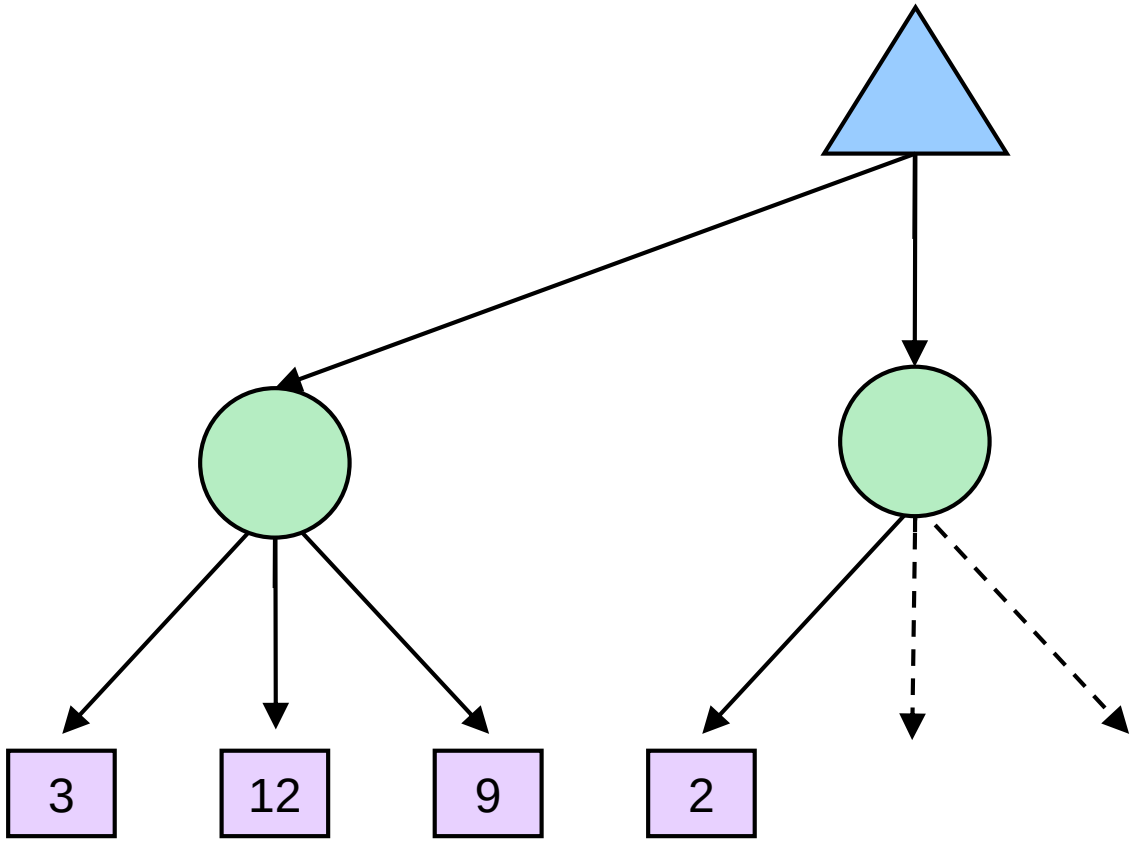


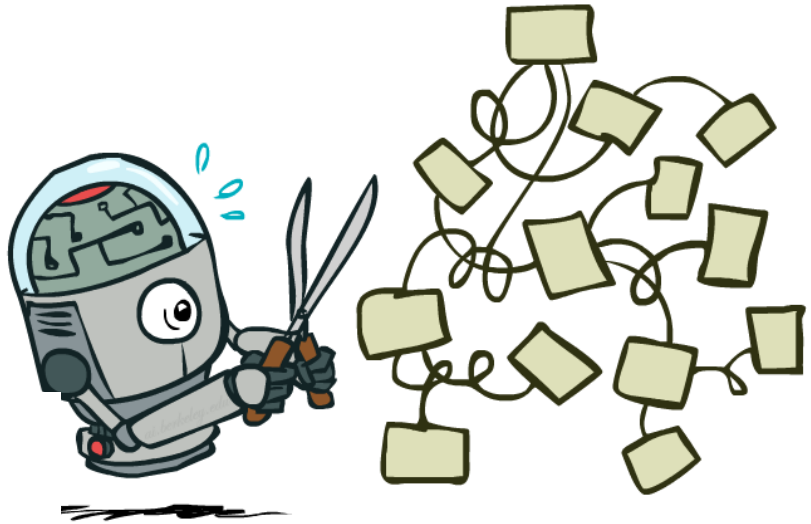
$$v = (1/2)\,(8) + (1/3)\,(24) + (1/6)\,(-12) = 10$$

Expectimax Example



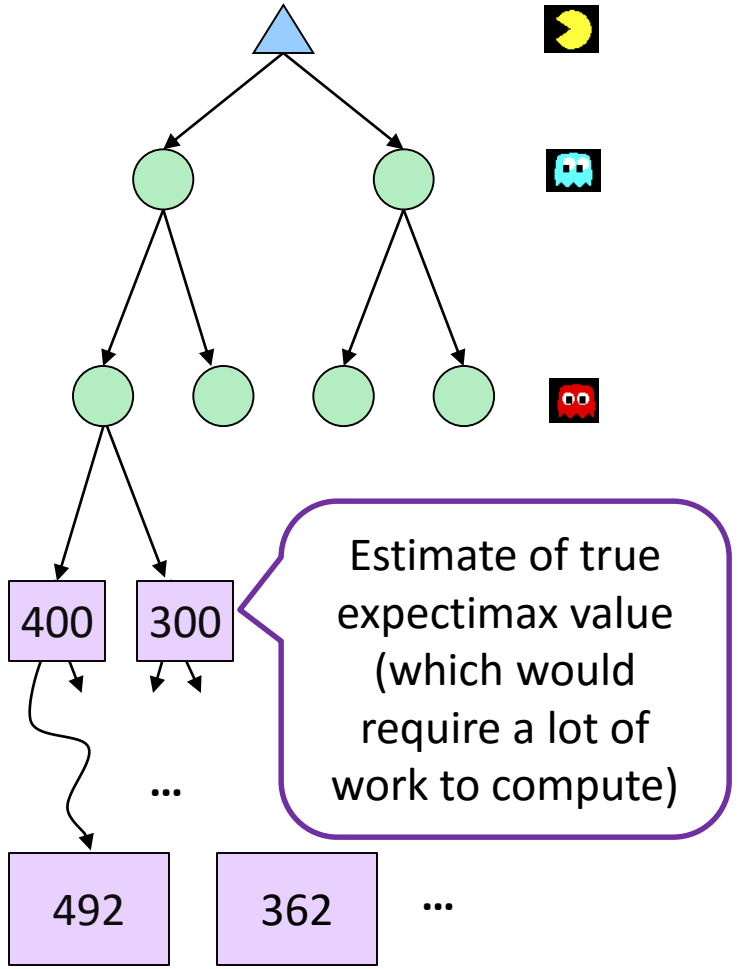
Expectimax Pruning?





Can we Prune ?

Depth-Limited Expectimax



can we depth Limited Search Here?