Python Programming

```
>>> values = ChainMap()
>>> values['x'] = 1
>>> # Add a new mapping
>>> values = values.new_child()
>>> values['x'] = 2
>>> # Add a new mapping
>>> values = values.new_child()
>>> values['x'] = 3
>>> values
ChainMap({'x': 3}, {'x': 2}, {'x': 1})
>>> values['x']
3
>>> # Discard last mapping
>>> values = values.parents
>>> values['x']
2
```

```
>>> # Discard last mapping
>>> values = values.parents
>>> values['x']
1
>>> values
ChainMap({'x': 1})
```

```
>>> a = {'x': 1, 'z': 3 }
>>> b = {'y': 2, 'z': 4 }
>>> merged = dict(b)
>>> merged.update(a)
>>> merged['x']
1
>>> merged['y']
2
>>> merged['z']
3
```

```
>>> a['x'] = 13
>>> merged['x']
1
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> line = 'asdf fjdk; afed, fjek,asdf,      foo'
>>> import re
>>> re.split(r'[;,\s]\s*', line)
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']

>>> fields = re.split(r'(;|,|\s)\s*', line)
>>> fields
['asdf', ' ', 'fjdk', ';', 'afed', ',', 'fjek', ',', 'asdf', ',', 'foo']
>>> values = fields[::2]
>>> delimiters = fields[1::2] + ['']
>>> values
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
>>> delimiters
[' ', ';', ',', ',', ',', '']

>>> # Reform the line using the same delimiters
>>> ''.join(v+d for v,d in zip(values, delimiters))
'asdf fjdk;afed,fjek,asdf,foo'
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> re.split(r'(?:,|;|\s)\s*', line)
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
```

```
>>> filename = 'spam.txt'
>>> filename.endswith('.txt')
True
>>> filename.startswith('file:')
False
>>> url = 'http://www.python.org'
>>> url.startswith('http:')
True
```

```
>>> import os
>>> filenames = os.listdir('.')
>>> filenames
[ 'Makefile', 'foo.c', 'bar.py', 'spam.c', 'spam.h' ]
>>> [name for name in filenames if name.endswith(('.c', '.h')) ]
['foo.c', 'spam.c', 'spam.h'
>>> any(name.endswith('.py') for name in filenames)
True
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```python
from urllib.request import urlopen

def read_data(name):
    if name.startswith(('http:', 'https:', 'ftp:')):
        return urlopen(name).read()
    else:
        with open(name) as f:
            return f.read()
```

```python
>>> choices = ['http:', 'ftp:']
>>> url = 'http://www.python.org'
>>> url.startswith(choices)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: startswith first arg must be str or a tuple of str, not list
>>> url.startswith(tuple(choices))
True
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> filename = 'spam.txt'
>>> filename[-4:] == '.txt'
True
>>> url = 'http://www.python.org'
>>> url[:5] == 'http:' or url[:6] == 'https:' or url[:4] == 'ftp:'
True

>>> import re
>>> url = 'http://www.python.org'
>>> re.match('http:|https:|ftp:', url)
<_sre.SRE_Match object at 0x101253098>
>>>
```

```
if any(name.endswith(('.c', '.h')) for name in listdir(dirname)):
    ...
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> from fnmatch import fnmatch, fnmatchcase
>>> fnmatch('foo.txt', '*.txt')
True
>>> fnmatch('foo.txt', '?oo.txt')
True
>>> fnmatch('Dat45.csv', 'Dat[0-9]*')
True
>>> names = ['Dat1.csv', 'Dat2.csv', 'config.ini', 'foo.py']
>>> [name for name in names if fnmatch(name, 'Dat*.csv')]
['Dat1.csv', 'Dat2.csv']
>>>
```

```
>>> # On OS X (Mac)
>>> fnmatch('foo.txt', '*.TXT')
False
```

```
>>> # On Windows
>>> fnmatch('foo.txt', '*.TXT')
True
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> fnmatchcase('foo.txt', '*.TXT')
False

addresses = [
    '5412 N CLARK ST',
    '1060 W ADDISON ST',
    '1039 W GRANVILLE AVE',
    '2122 N CLARK ST',
    '4802 N BROADWAY',
]

>>> from fnmatch import fnmatchcase
>>> [addr for addr in addresses if fnmatchcase(addr, '* ST')]
['5412 N CLARK ST', '1060 W ADDISON ST', '2122 N CLARK ST']
>>> [addr for addr in addresses if fnmatchcase(addr, '54[0-9][0-9] *CLARK*')]
['5412 N CLARK ST']
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> text1 = '11/27/2012'
>>> text2 = 'Nov 27, 2012'
>>>
>>> import re
>>> # Simple matching: \d+ means match one or more digits
>>> if re.match(r'\d+/\d+/\d+', text1):
...     print('yes')
... else:
...     print('no')
...
yes
>>> if re.match(r'\d+/\d+/\d+', text2):
...     print('yes')
... else:
...     print('no')
...
no
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
>>> datepat = re.compile(r'\d+/\d+/\d+')
>>> if datepat.match(text1):
...     print('yes')
... else:
...     print('no')
...
yes
>>> if datepat.match(text2):
...     print('yes')
... else:
...     print('no')
...
no
```

```
>>> text = 'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> datepat.findall(text)
['11/27/2012', '3/13/2013']
>>> datepat = re.compile(r'(\d+)/(\d+)/(\d+)')
>>> m = datepat.match('11/27/2012')
>>> m
<_sre.SRE_Match object at 0x1005d2750>
>>> m.group(0)
'11/27/2012'
>>> m.group(1)
'11'
>>> m.group(2)
'27'
>>> m.group(3)
'2012'
>>> m.groups()
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

Python Programming

```
('11', '27', '2012')
>>> month, day, year = m.groups()
>>>

>>> # Find all matches (notice splitting into tuples)
>>> text
'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> datepat.findall(text)
[('11', '27', '2012'), ('3', '13', '2013')]
>>> for month, day, year in datepat.findall(text):
...     print('{}-{}-{}'.format(year, month, day))
...
2012-11-27
2013-3-13
```

```
>>> for m in datepat.finditer(text):
...     print(m.groups())
...
('11', '27', '2012')
('3', '13', '2013')
```

Source: Beazley, David; Jones, Brian K. (2013). *Python Cookbook* (3rd ed.).

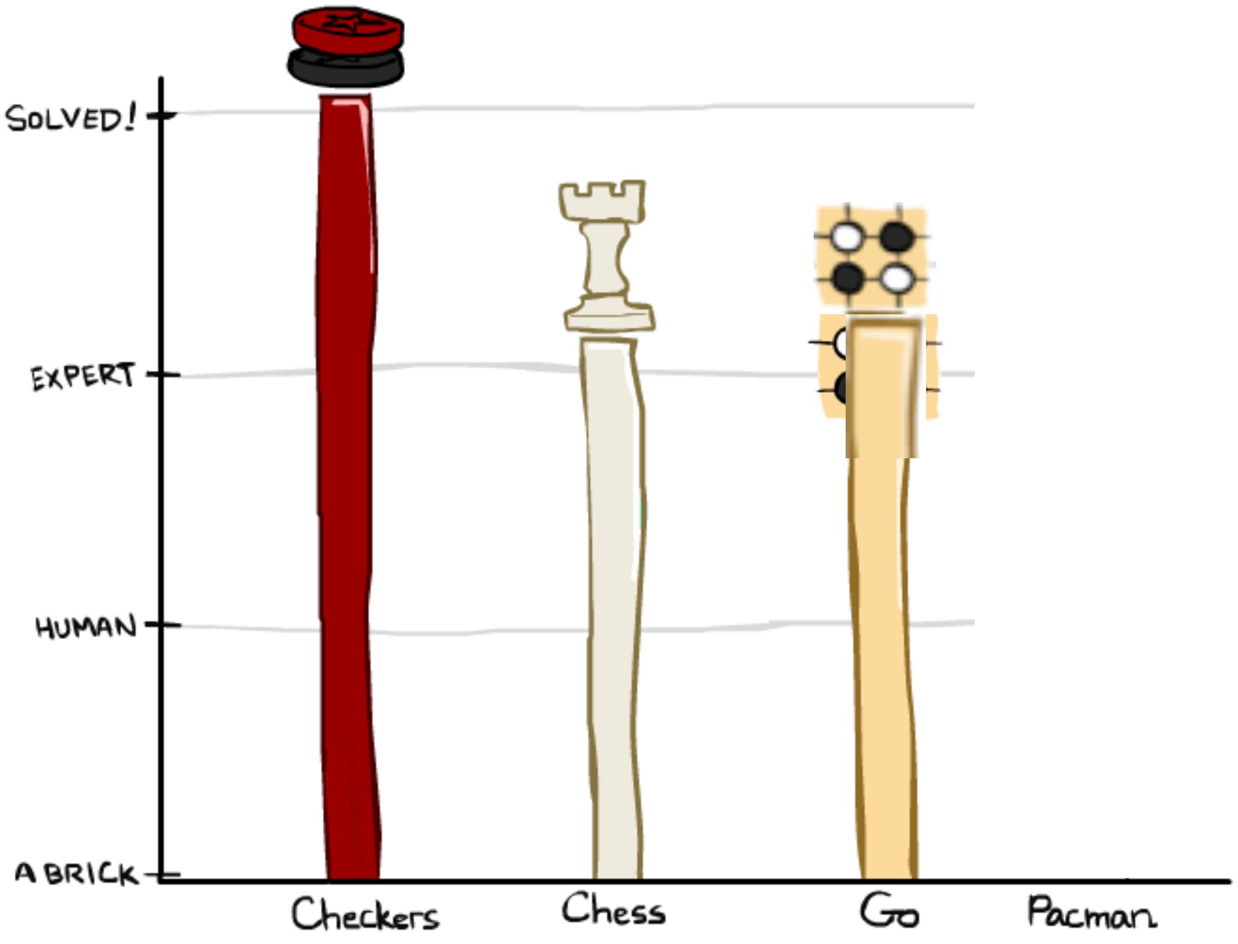
Game Playing State-of-the-Art

- **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!

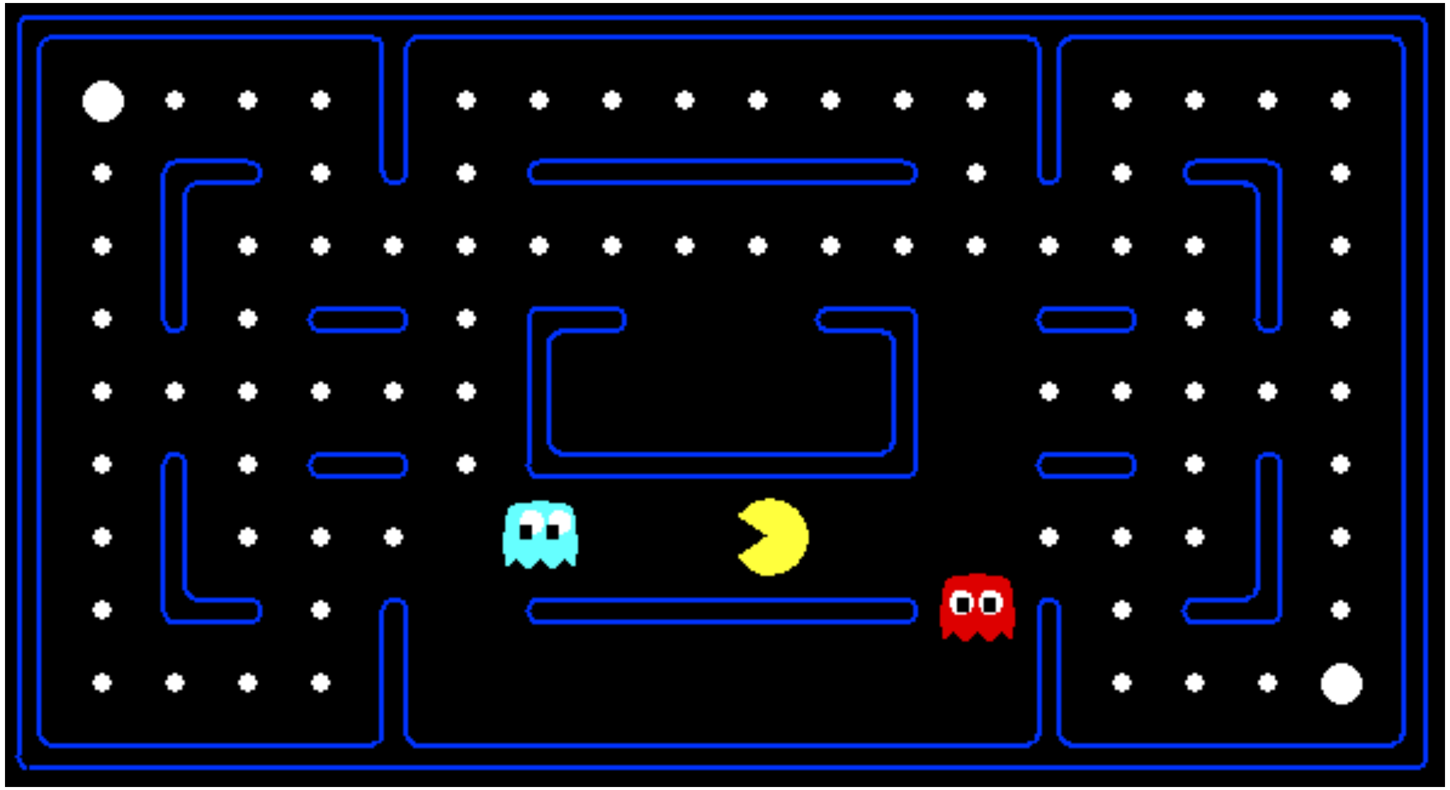
- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

- **Go: 2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search, learned evaluation function.**

- **Pacman**

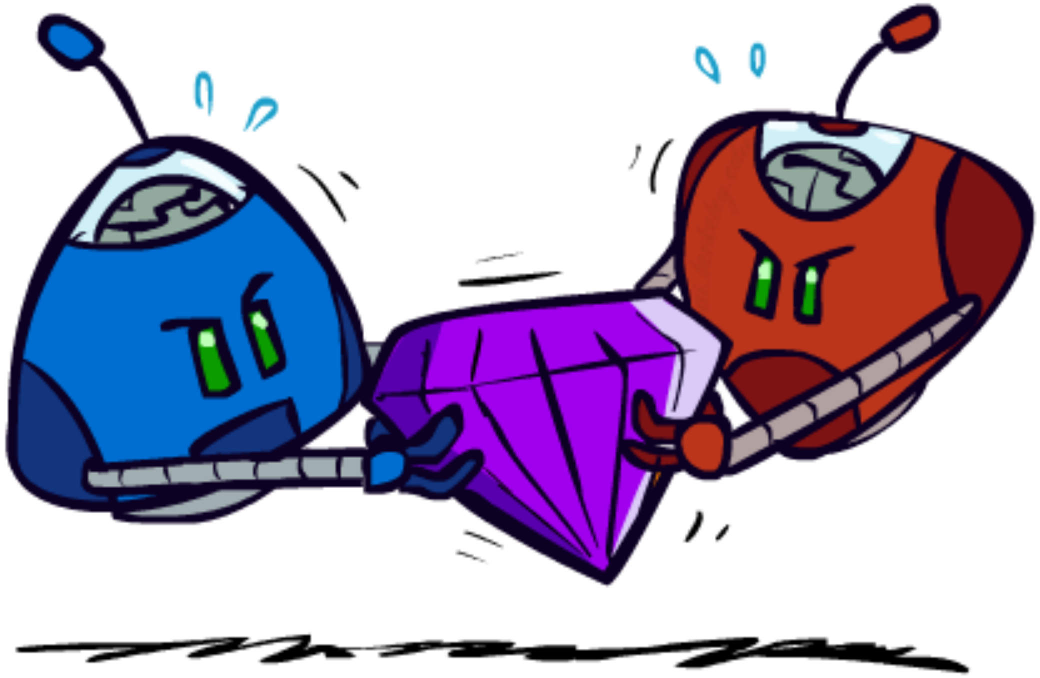


Behavior from Computation



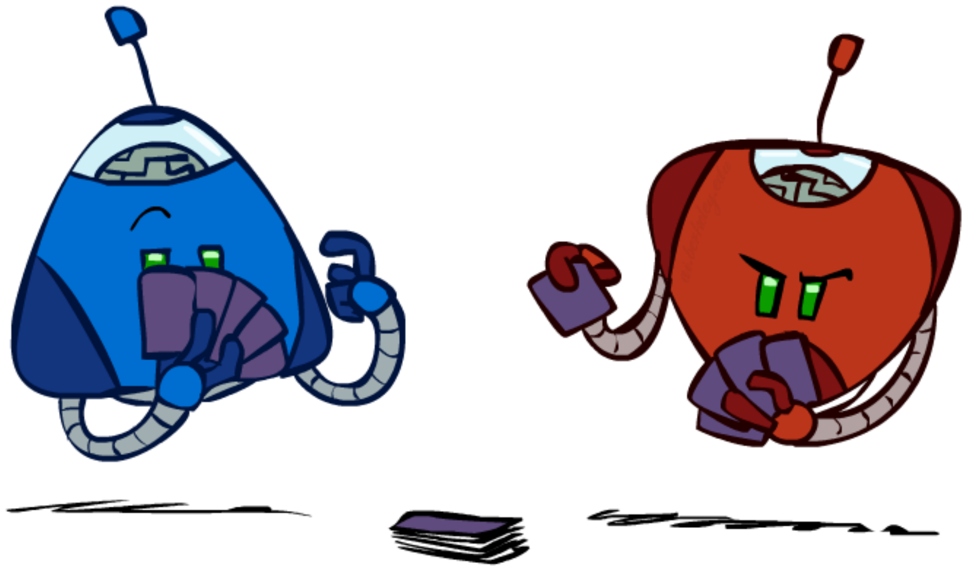
[Demo: mystery pacman (L6D1)]

Adversarial Games



Types of Games

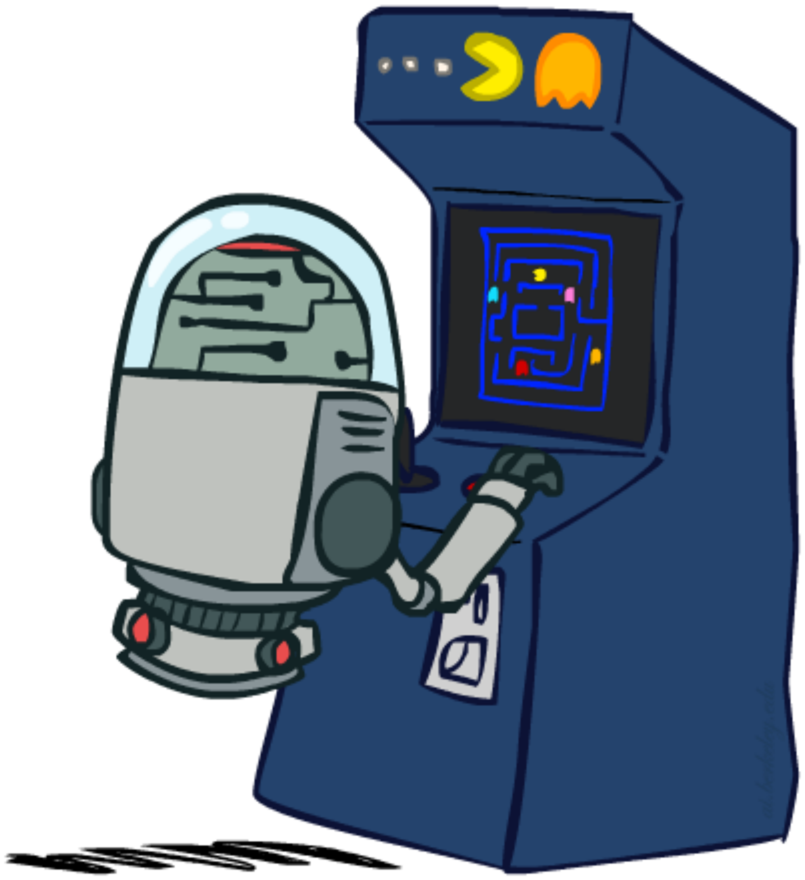
- Many different kinds of games!
- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?



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

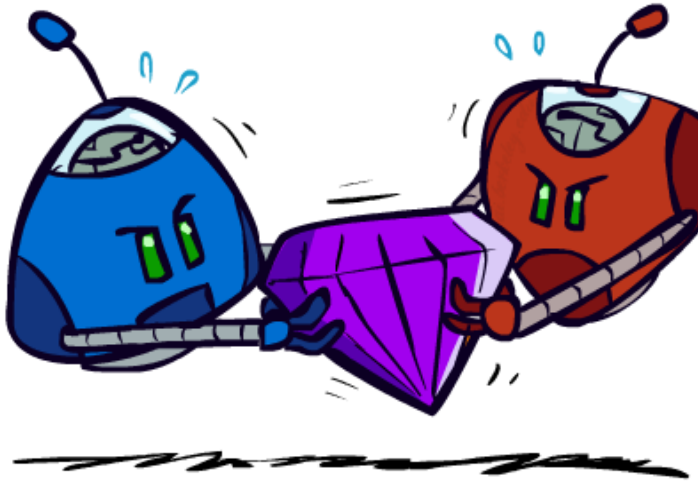
Deterministic Games

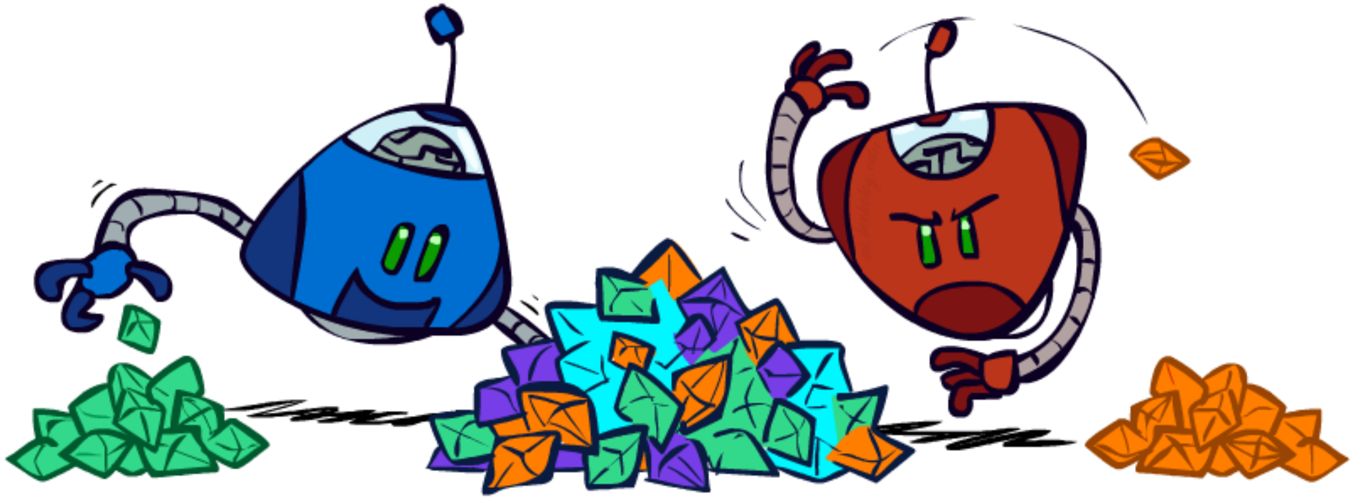
- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: $S \times A \rightarrow S$
  - Terminal Test: $S \rightarrow \{t,f\}$
  - Terminal Utilities: $S \times P \rightarrow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



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

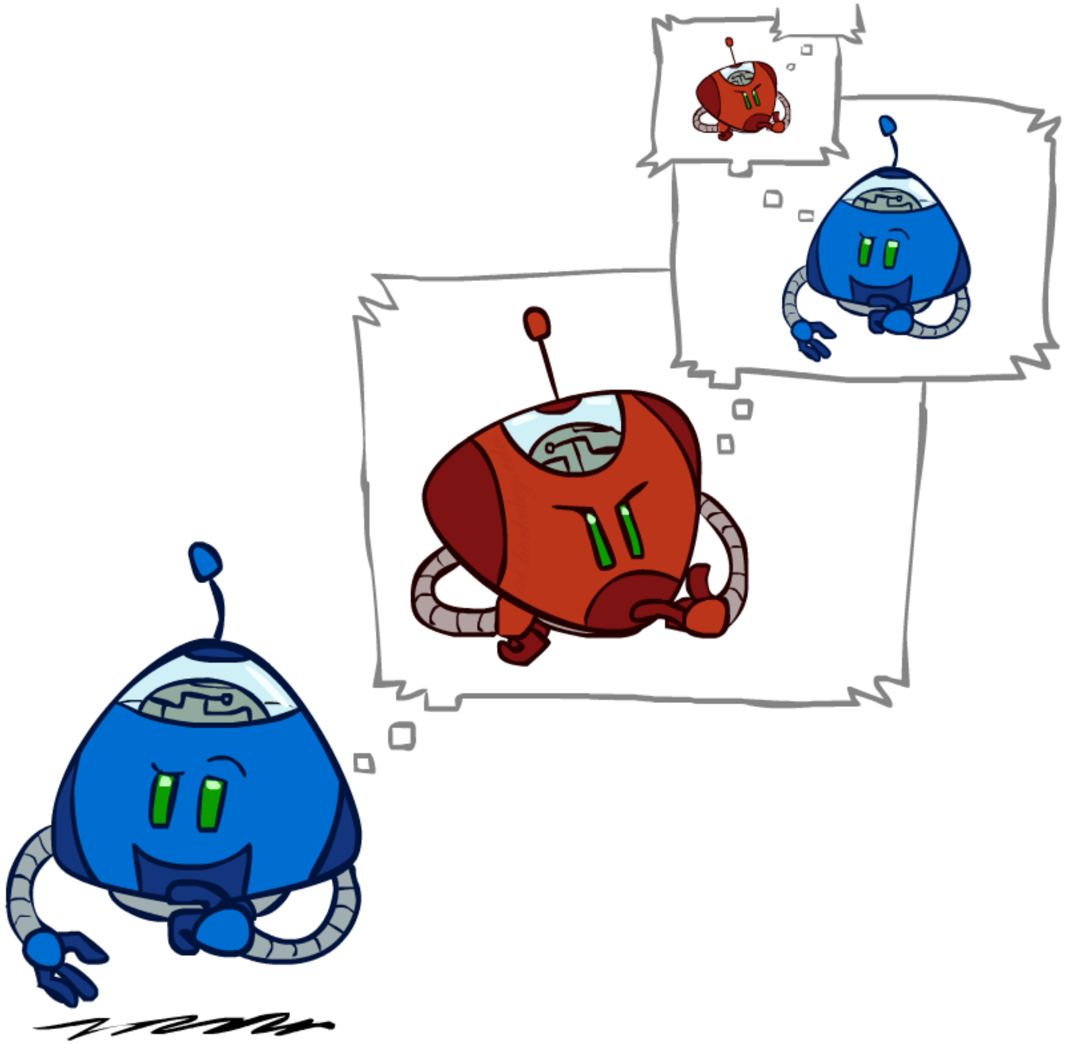
Zero-Sum Games



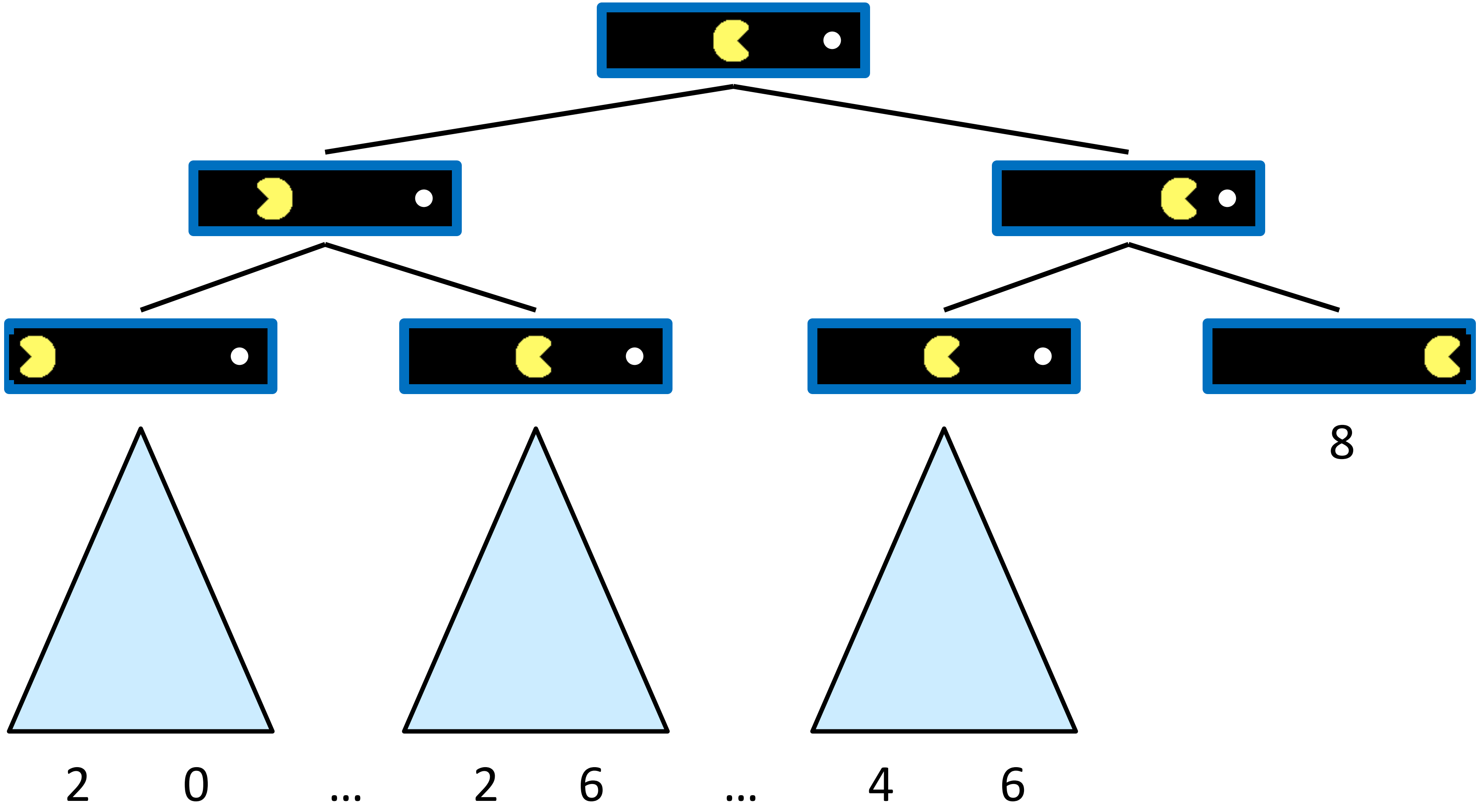


- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
  - More later on non-zero-sum games

Adversarial Search

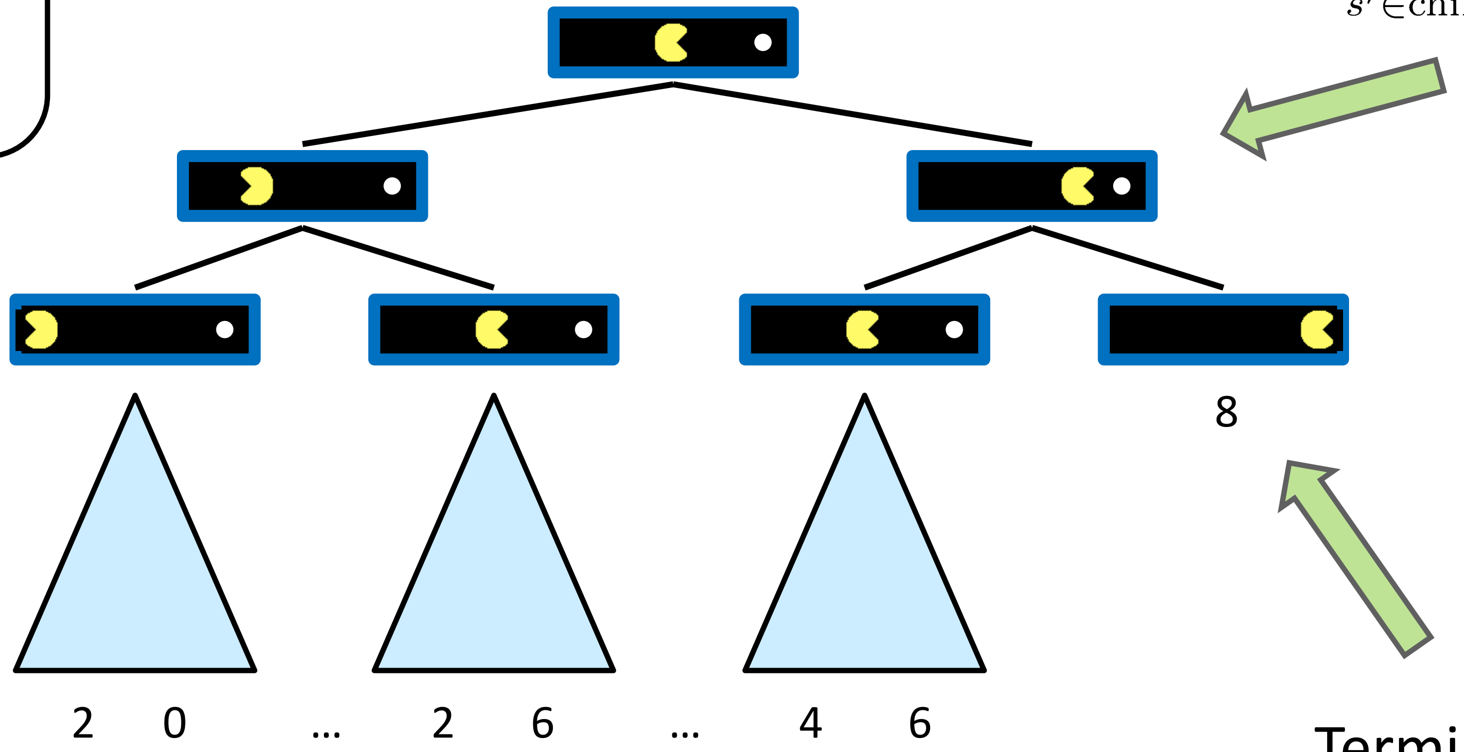


Single-Agent Trees



Value of a State

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



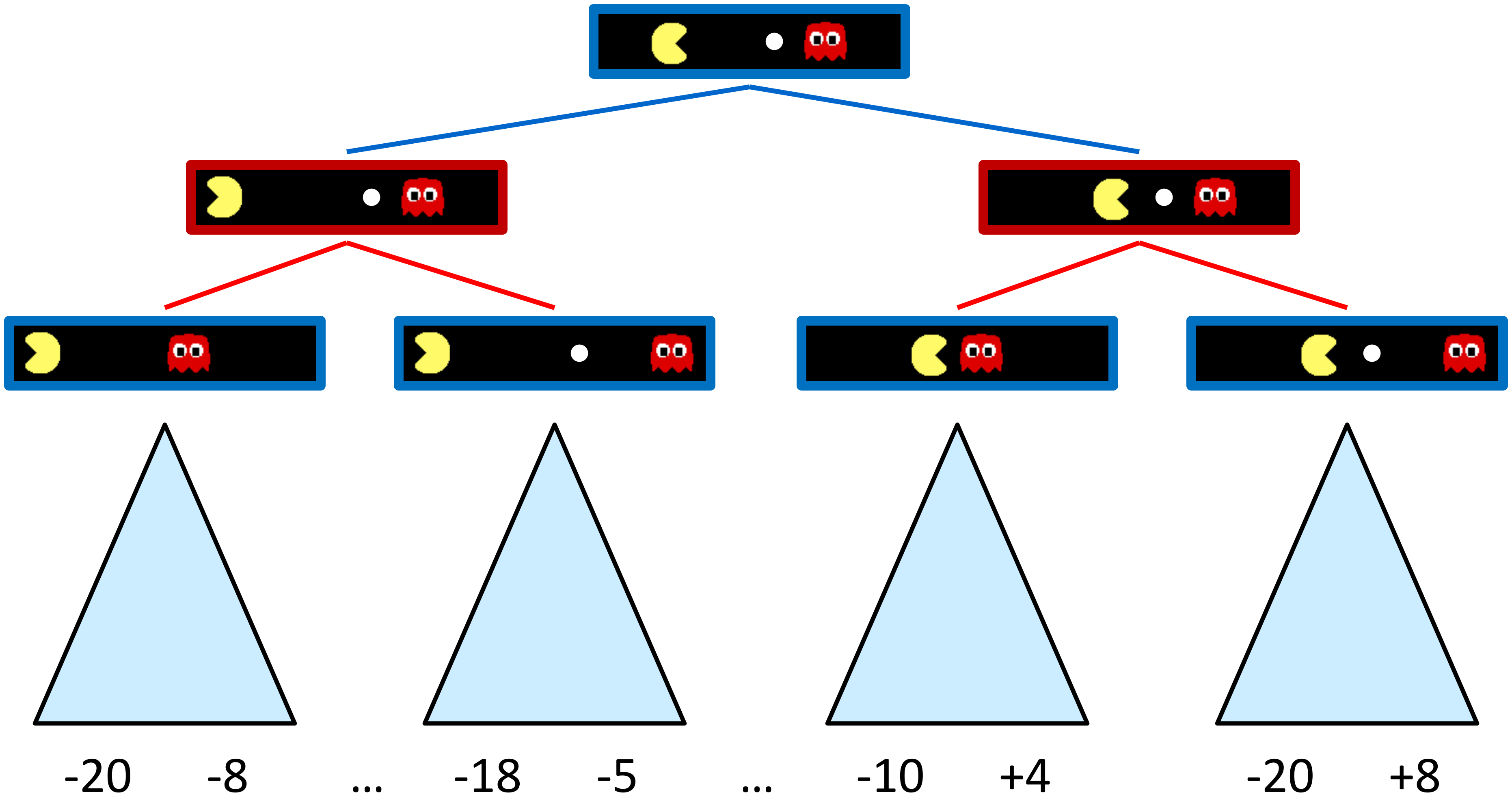
Non-Terminal States:

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

Terminal States:

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

Adversarial Game Trees



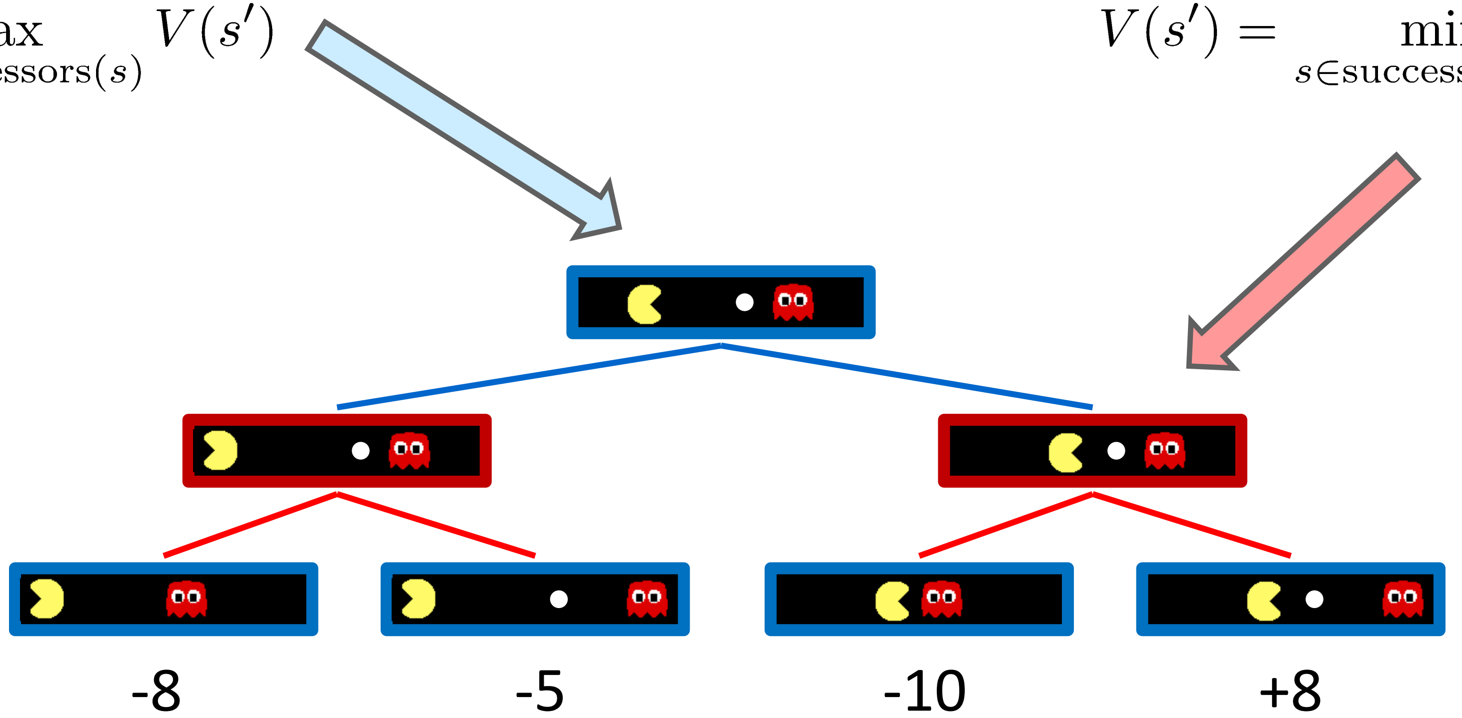
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

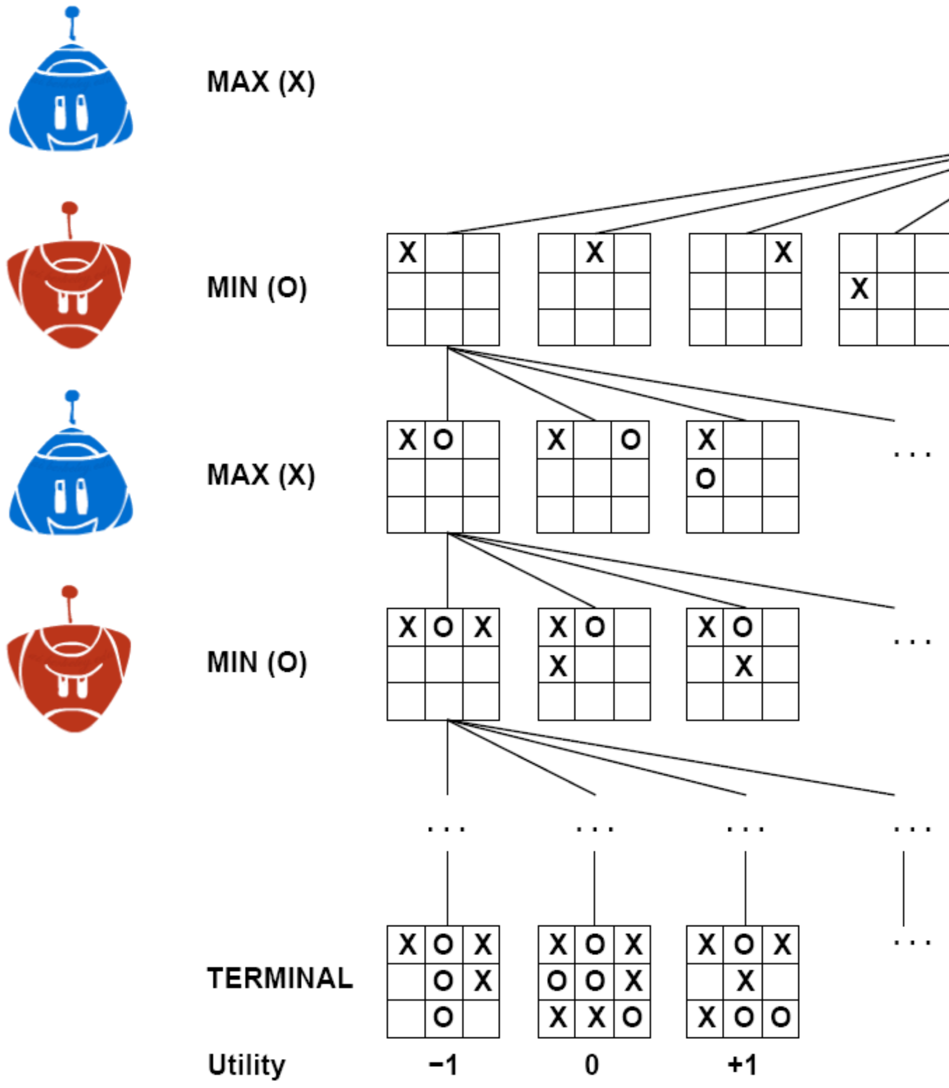
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

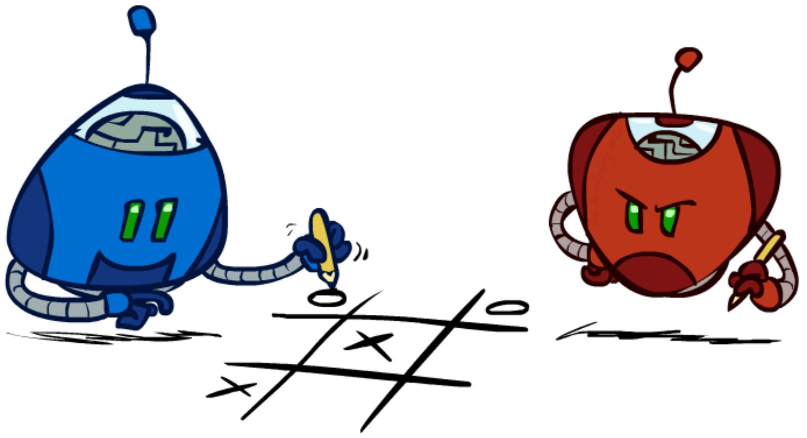


Terminal States:

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

Tic-Tac-Toe Game Tree



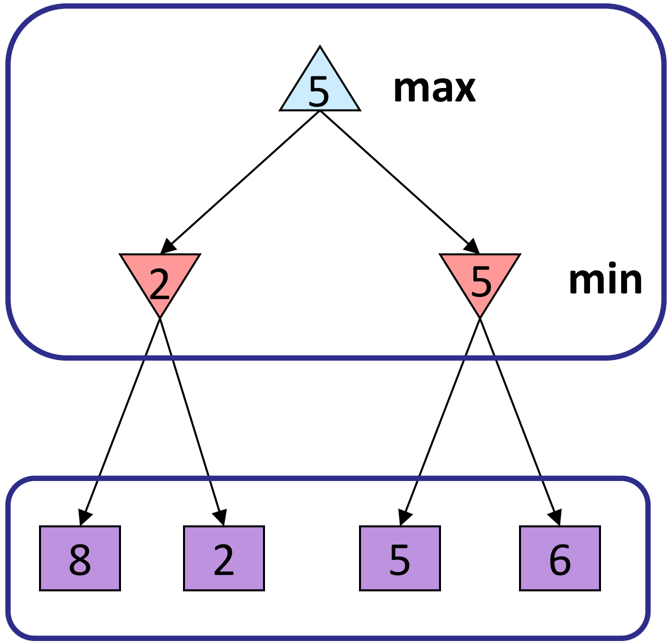


Adversarial Search (Minimax)

- 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
```

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

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

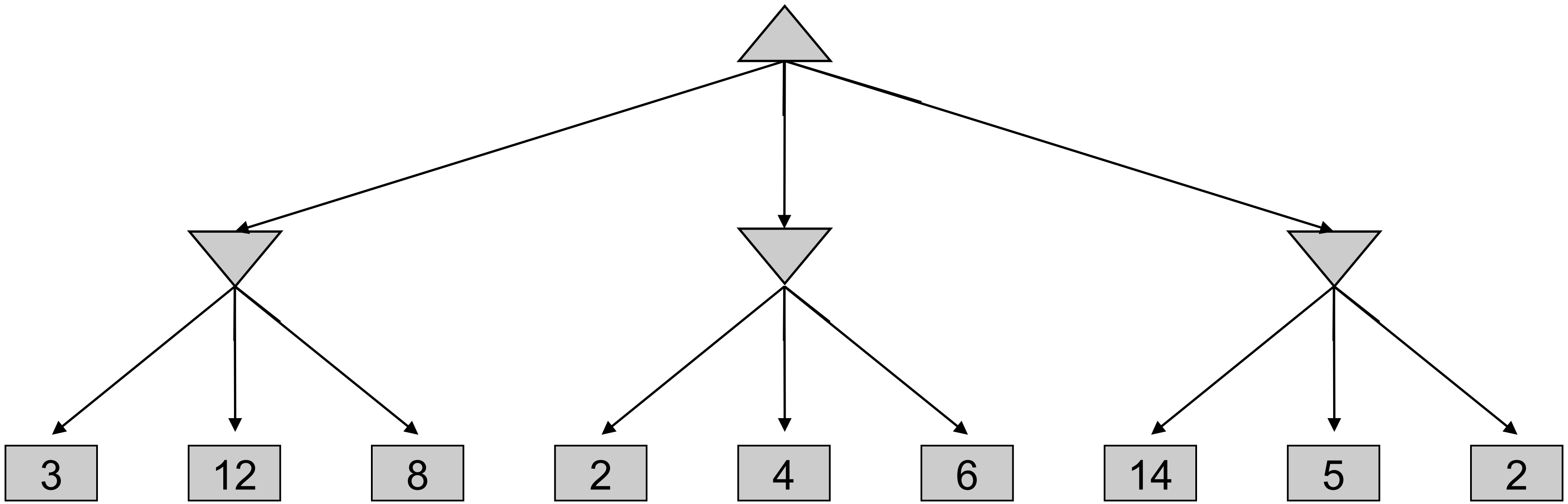
Minimax Implementation (Dispatch)

```
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)
```

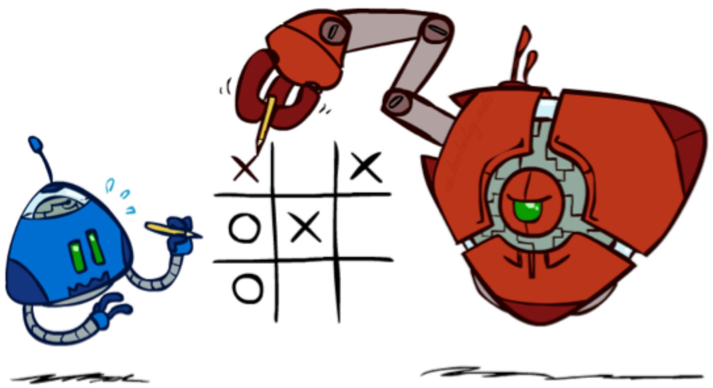
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

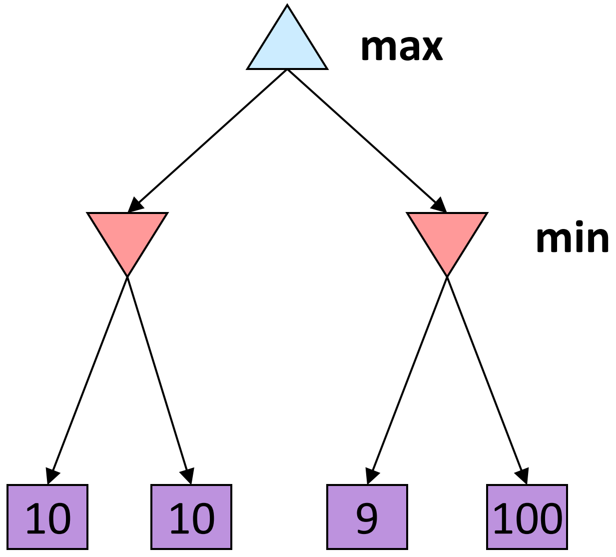
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

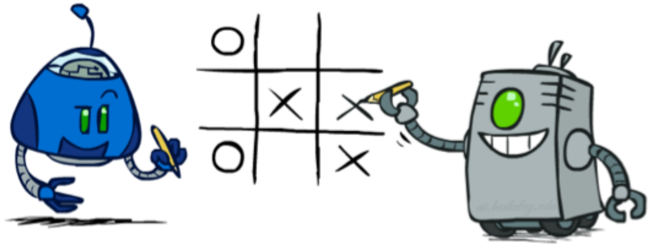
Minimax Example



Minimax Properties







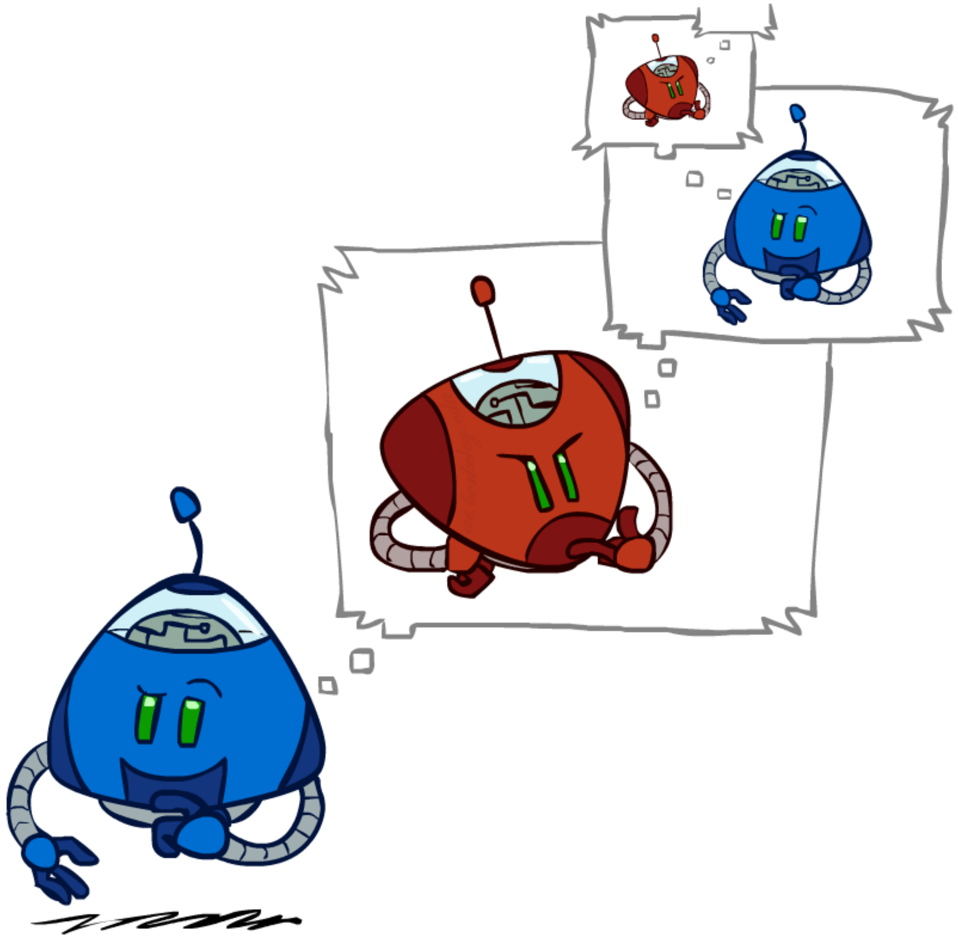
Optimal against a perfect player. Otherwise?

[Demo: min vs exp (L6D2, L6D3)]

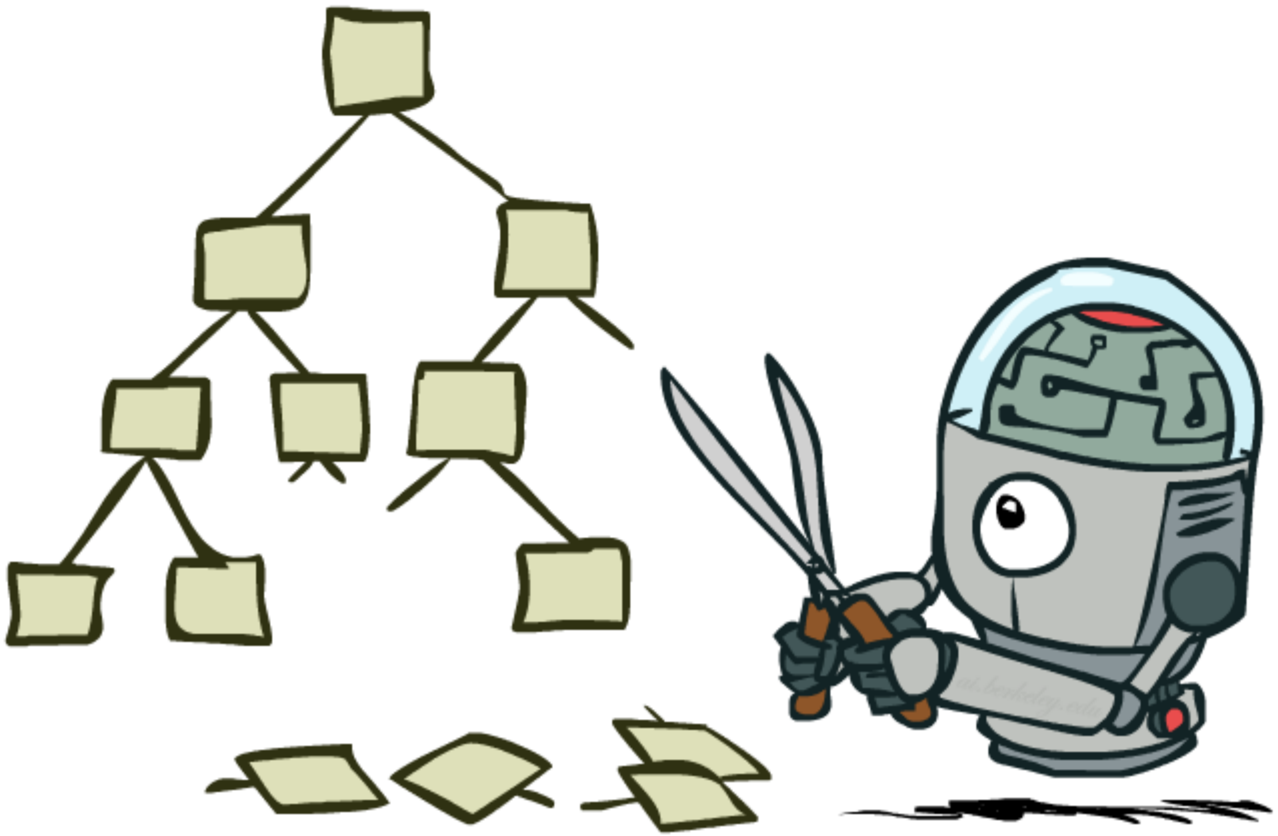
Minimax Efficiency

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

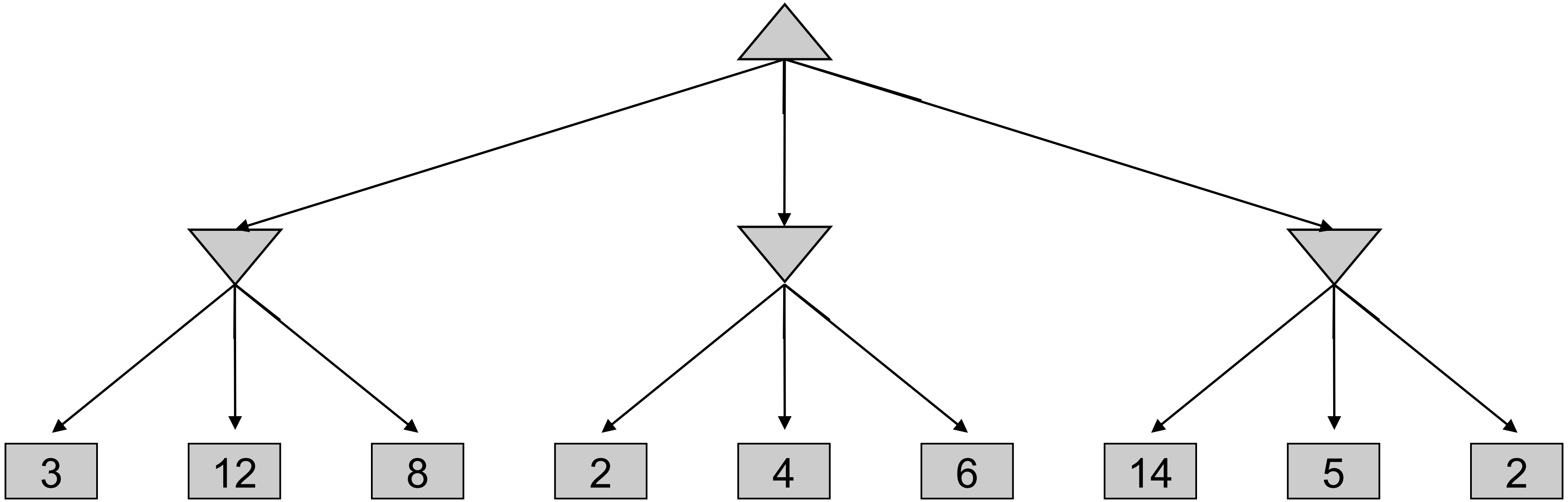
- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



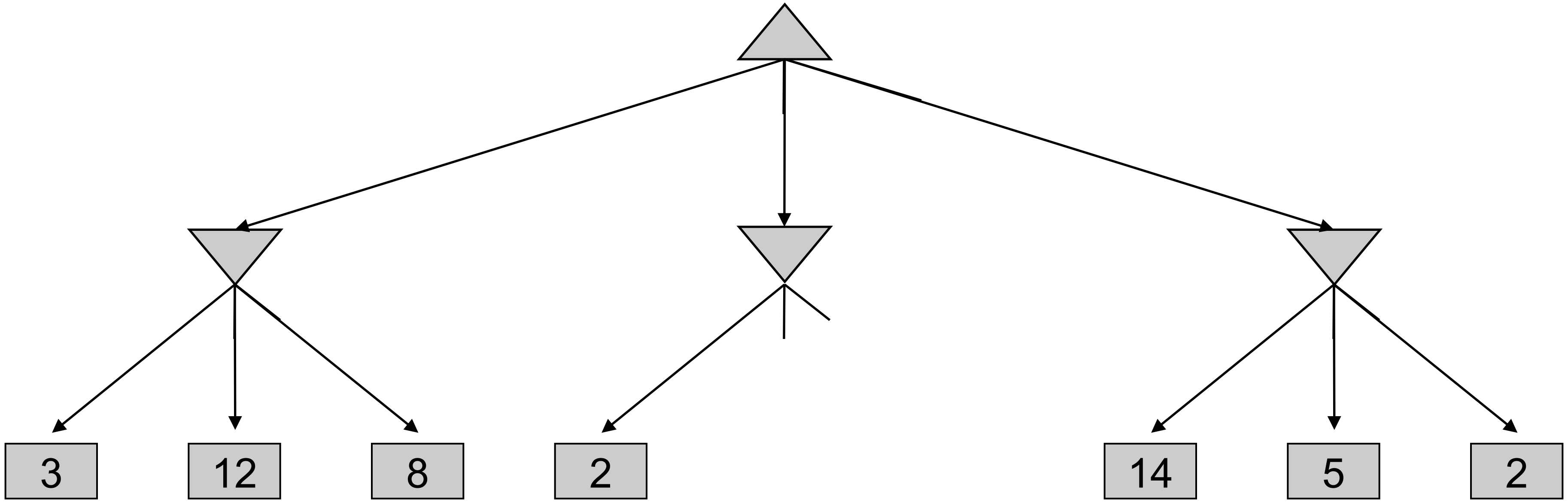
Game Tree Pruning



Minimax Example



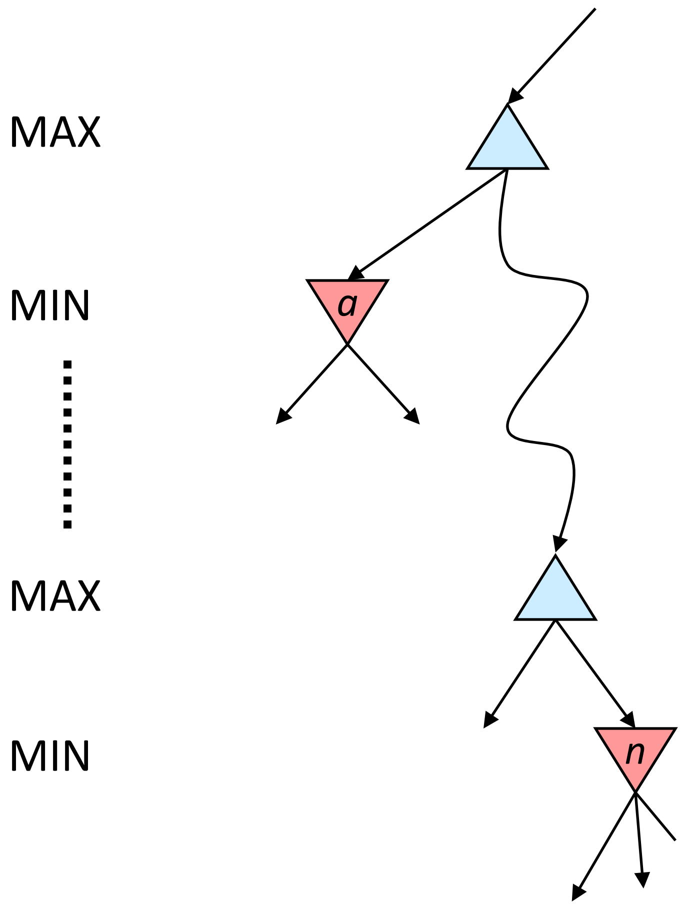
Minimax Pruning



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



Alpha-Beta Implementation

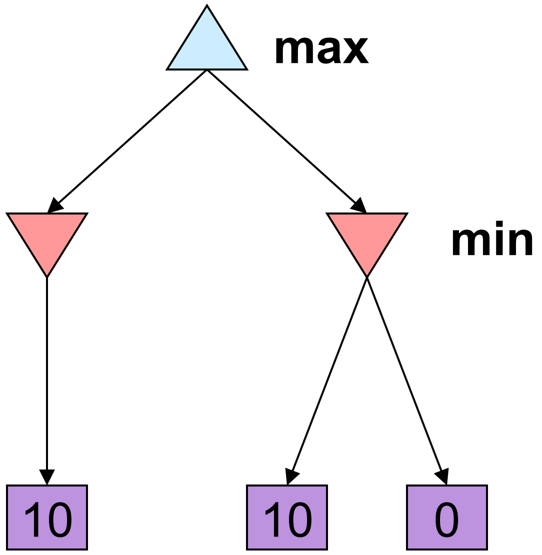
α: MAX's best option on path to root
β: MIN's best option on path to root

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

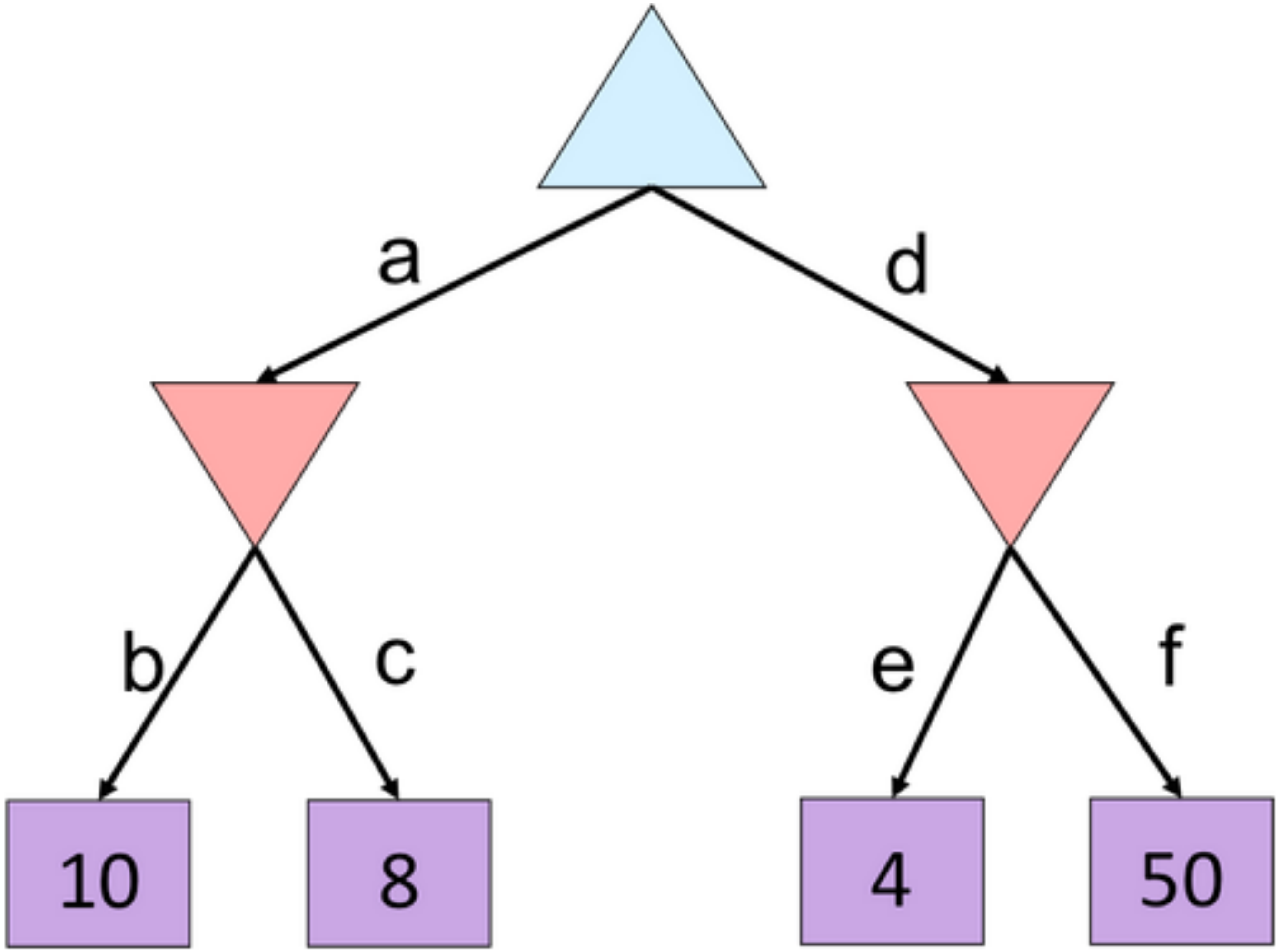
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

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…



Alpha-Beta Quiz



Alpha-Beta Quiz 2