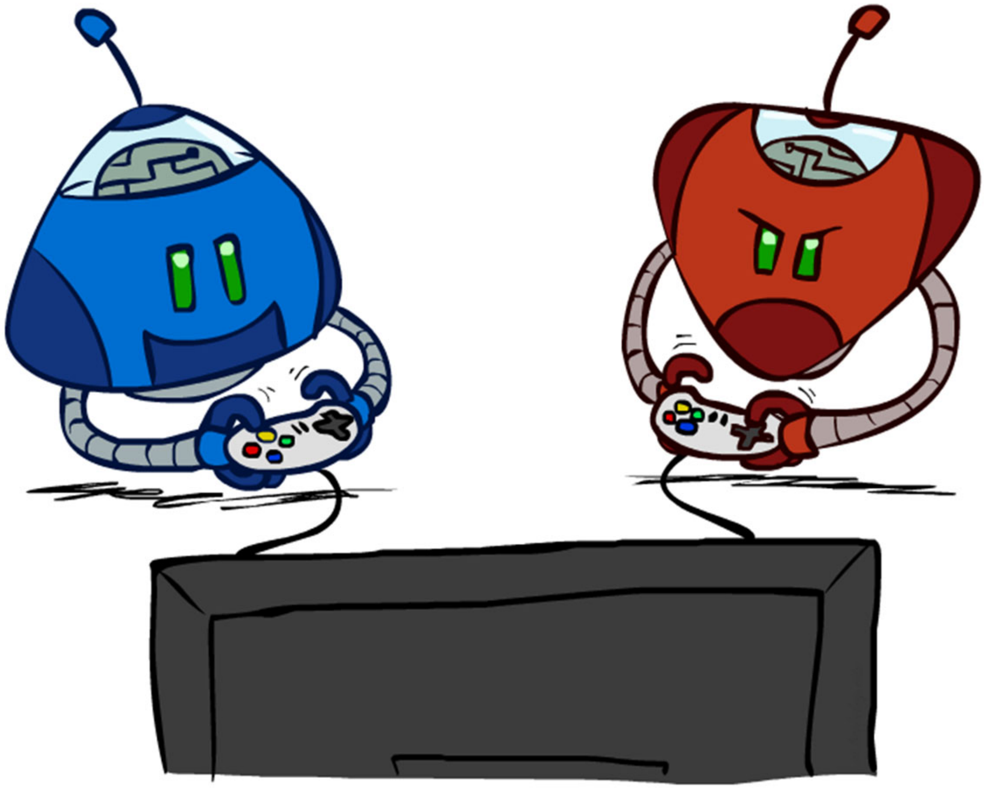
京都大学　情報学科 3回生向け講義「人工知能」

敵対探索（二人ゲーム）



情報学研究科　教授　神田崇行
kanda@i.kyoto-u.ac.jp

[Original slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

例題

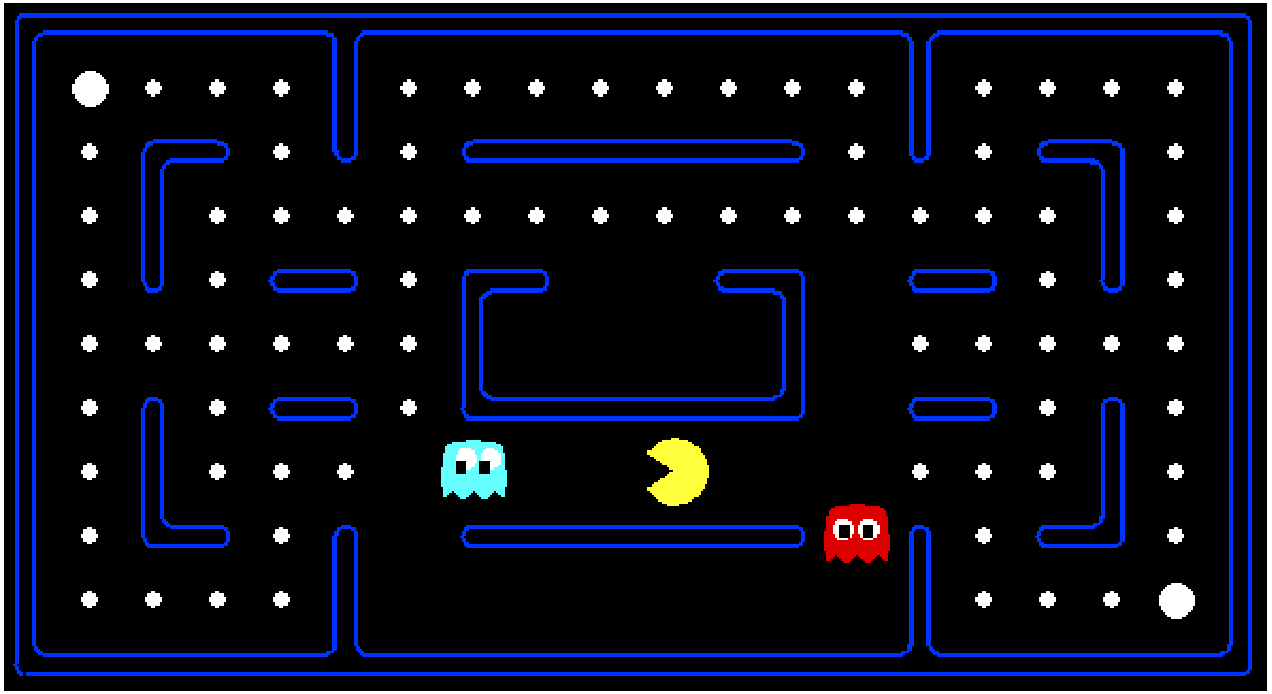
- ○×ゲーム (三目並べ、tic-tac-toe)：
  縦、横、斜め、いずれかに先に３つ並べたら勝ち
- １手先、２手先を読んだら、どこに置くのが良さそうでしょうか？
- みなさんの「知能」のように、ゲームで良い手を指すエージェントを作るには？

×	○	

次はあなた（×）の番

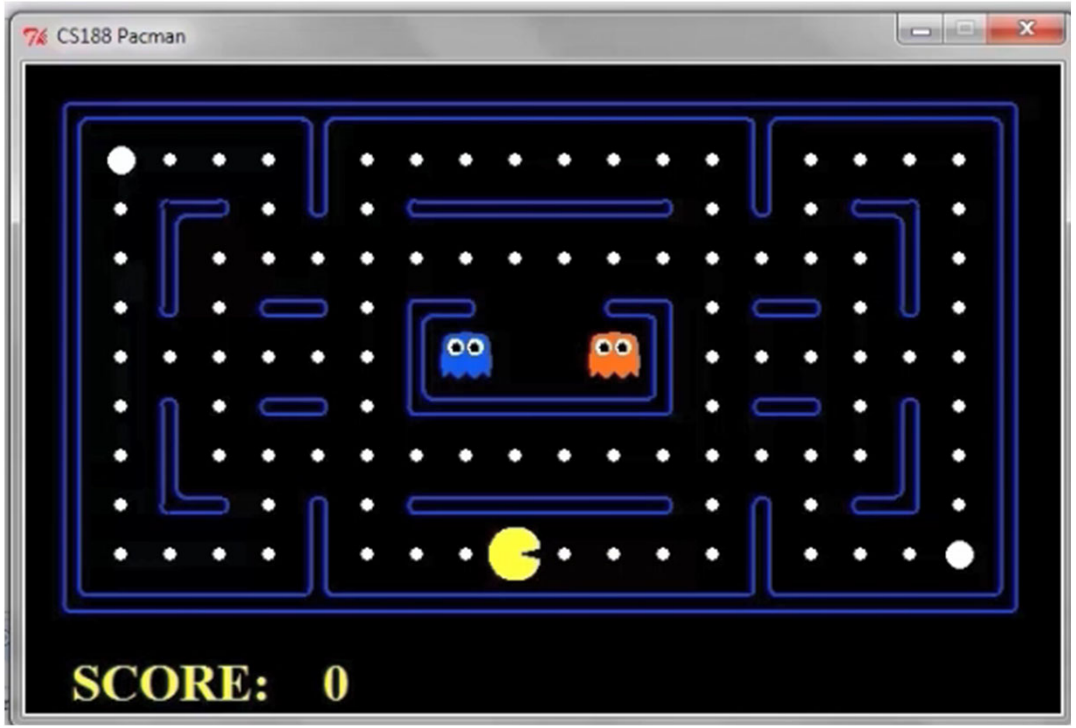
計算に基づく知的行動

この講義の目指すものは、知的なふるまいを作り出す計算方法を明らかにすること



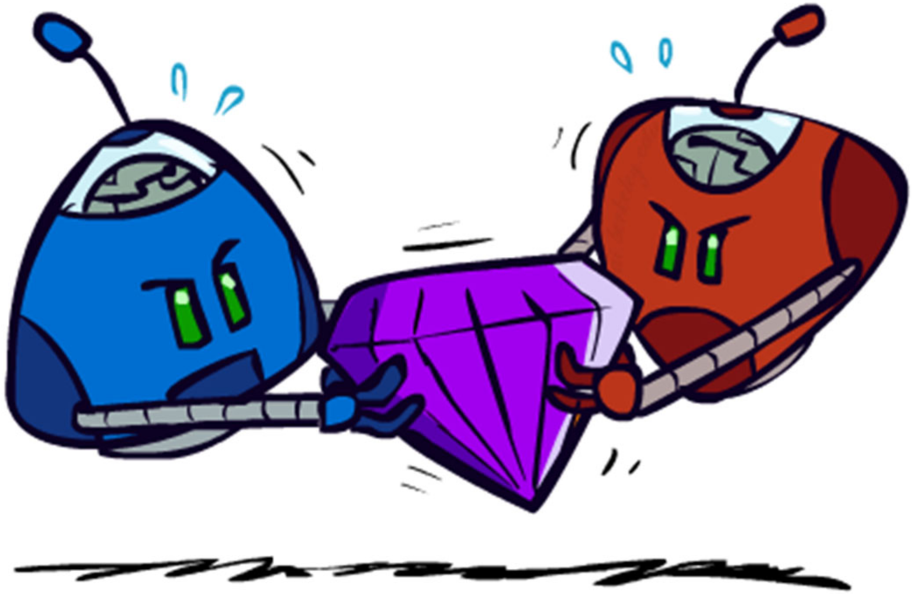
パックマンの例

たとえば、こういったパックマンの知的な行動はどうやったら可能になるか？



みなさんよりも上手くプレイしているかも？　これ、コンピュータのプログラムによるものです。
いったい、どうやったら、こんなことができるようになるのでしょうか？

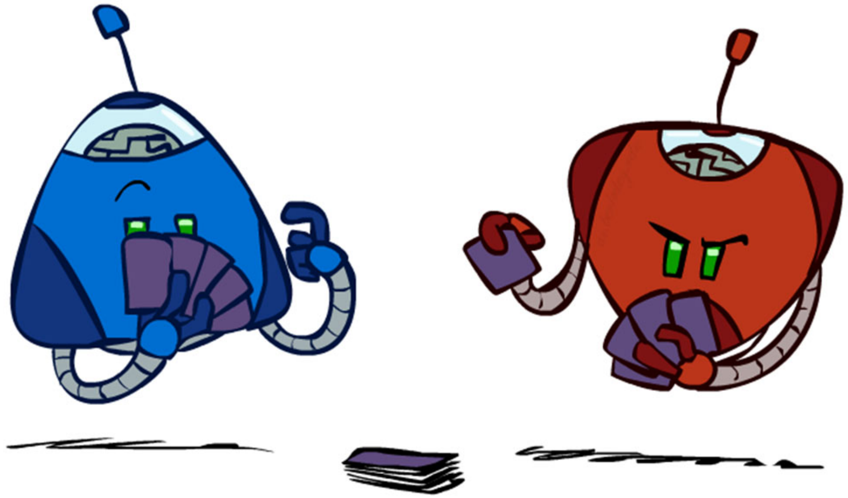
敵対的なゲーム



ゲームの種類

- 様々なタイプのゲーム!

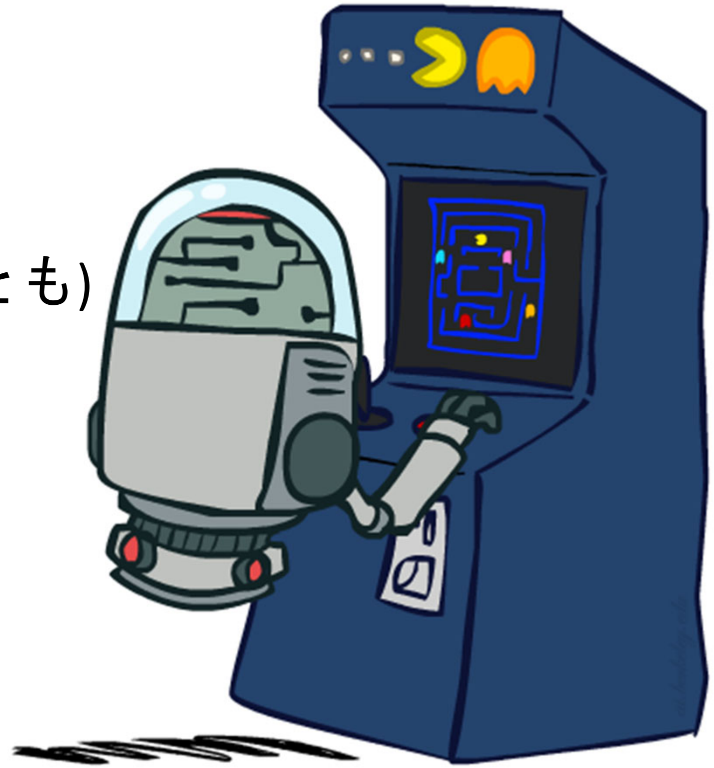
- 種類の分類:
  - 決定的か統計的か?
  - プレイヤーは一人、二人、それ以上？
  - ゼロサム問題か?
  - 完全情報 (状態を観測できるか)?



- 各状態からの次の一手を見出す戦略 (ポリシー) を計算するアルゴリズムとは？

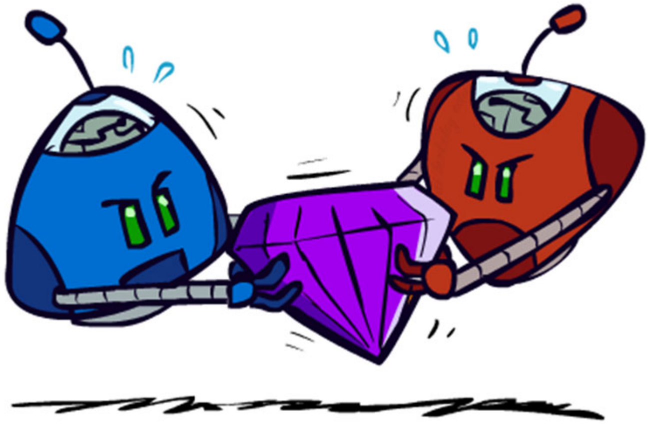
決定的なゲーム

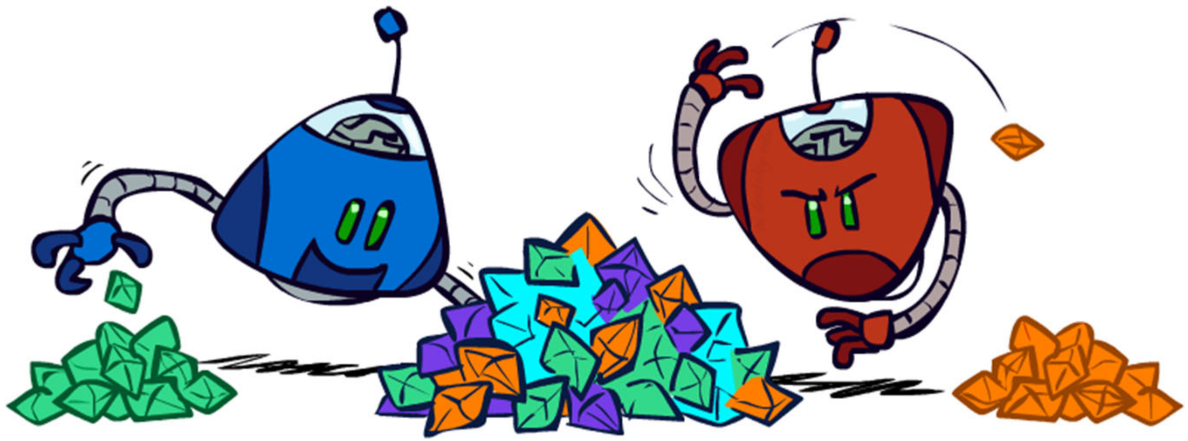
- 様々な定式化が可能だが、例えば:
  - 状態: S (初期状態s_0)
  - プレイヤー: P={1...N} (通常は交互に)
  - 行動: A (プレイヤーや状態によって変わることも)
  - 遷移関数: SxA $\rightarrow$ S
  - 終端テスト: S $\rightarrow$ {t,f}
  - 終端効用: SxP $\rightarrow$ R



- あるプレイヤーへの解はポリシー: S $\rightarrow$ A

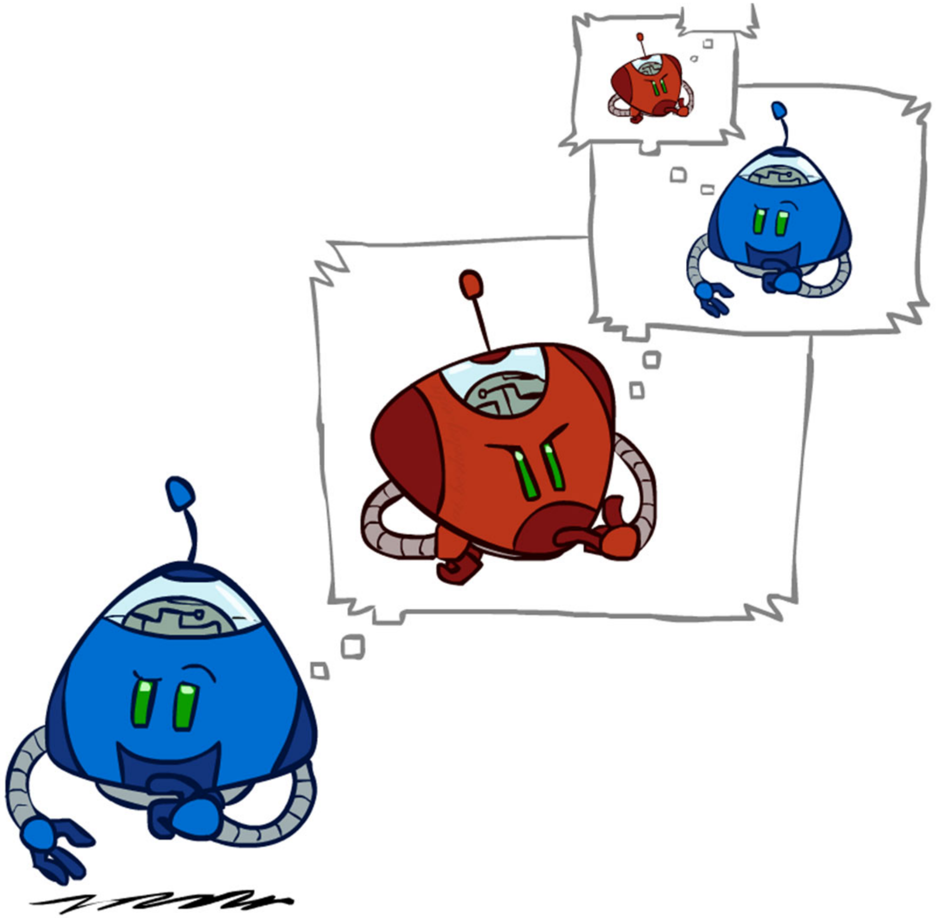
ゼロサムゲーム (Zero-Sum Games)



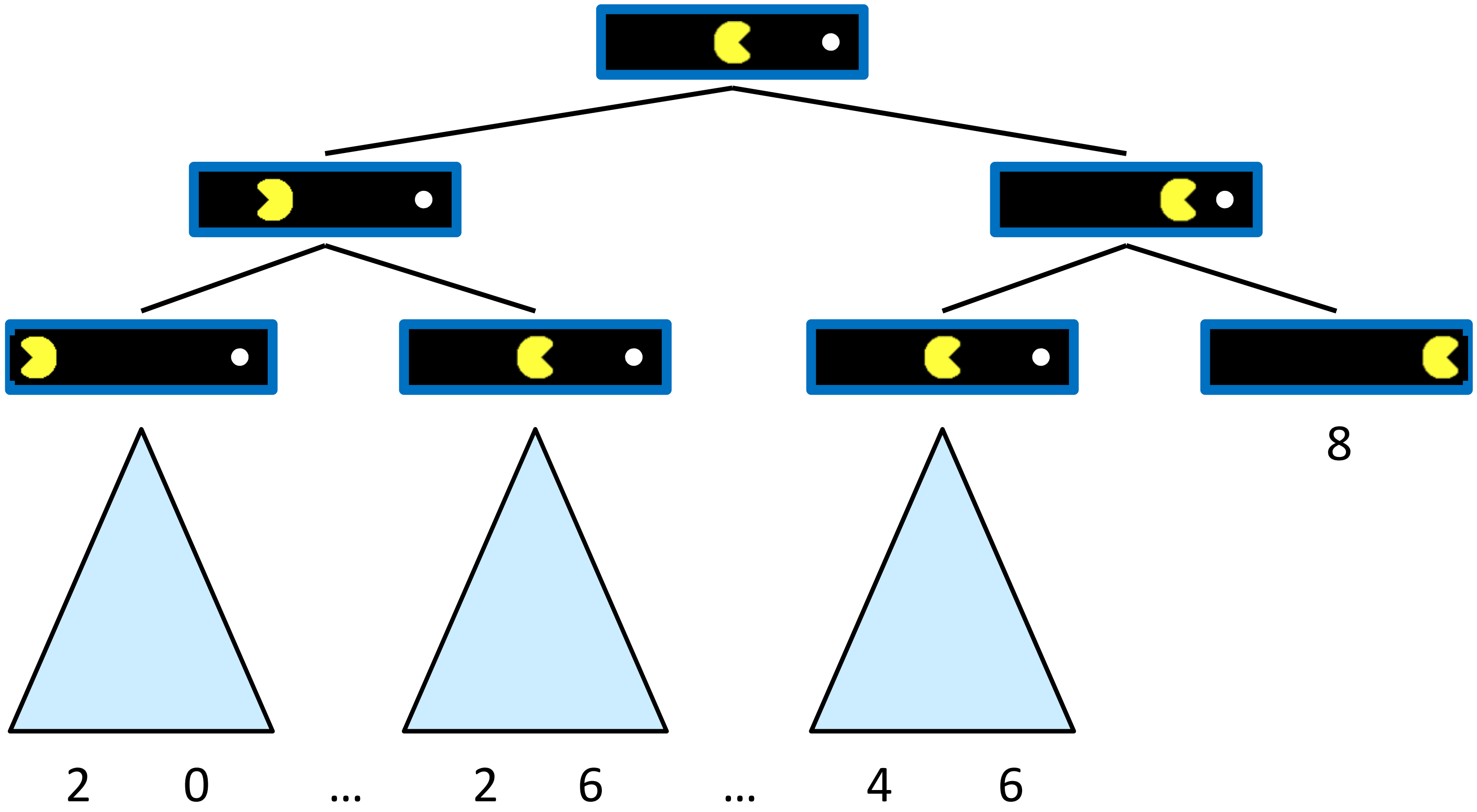


- ゼロサムゲーム
  - エージェント同士は互いに相反する効用 (結果の価値) を持つ
  - あるエージェントがその価値を最大化したいとすれば、他のエージェントは最小化したい
  - 敵対的、純粋な競争
- 一般的なゲーム
  - それぞれのエージェントは互いに独立した効用 (結果の価値) を持つ
  - 時に協調的、関わらない、競合、などさまざまなかかわり方
  - ゼロサムゲームでない問題は、後の回の講義で扱う

敵対探索 (Adversarial Search)

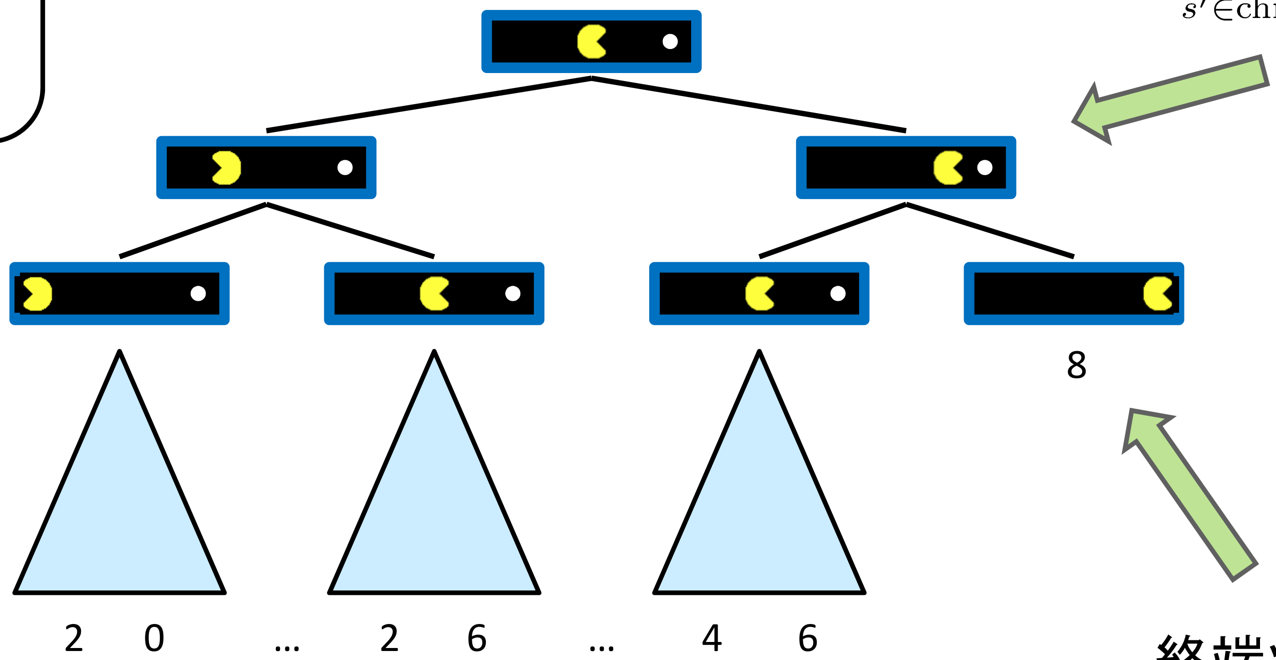


単一のエージェントに関する探索木



状態の価値

状態の価値:
その状態から達成
しうる最大の結果
（効用）



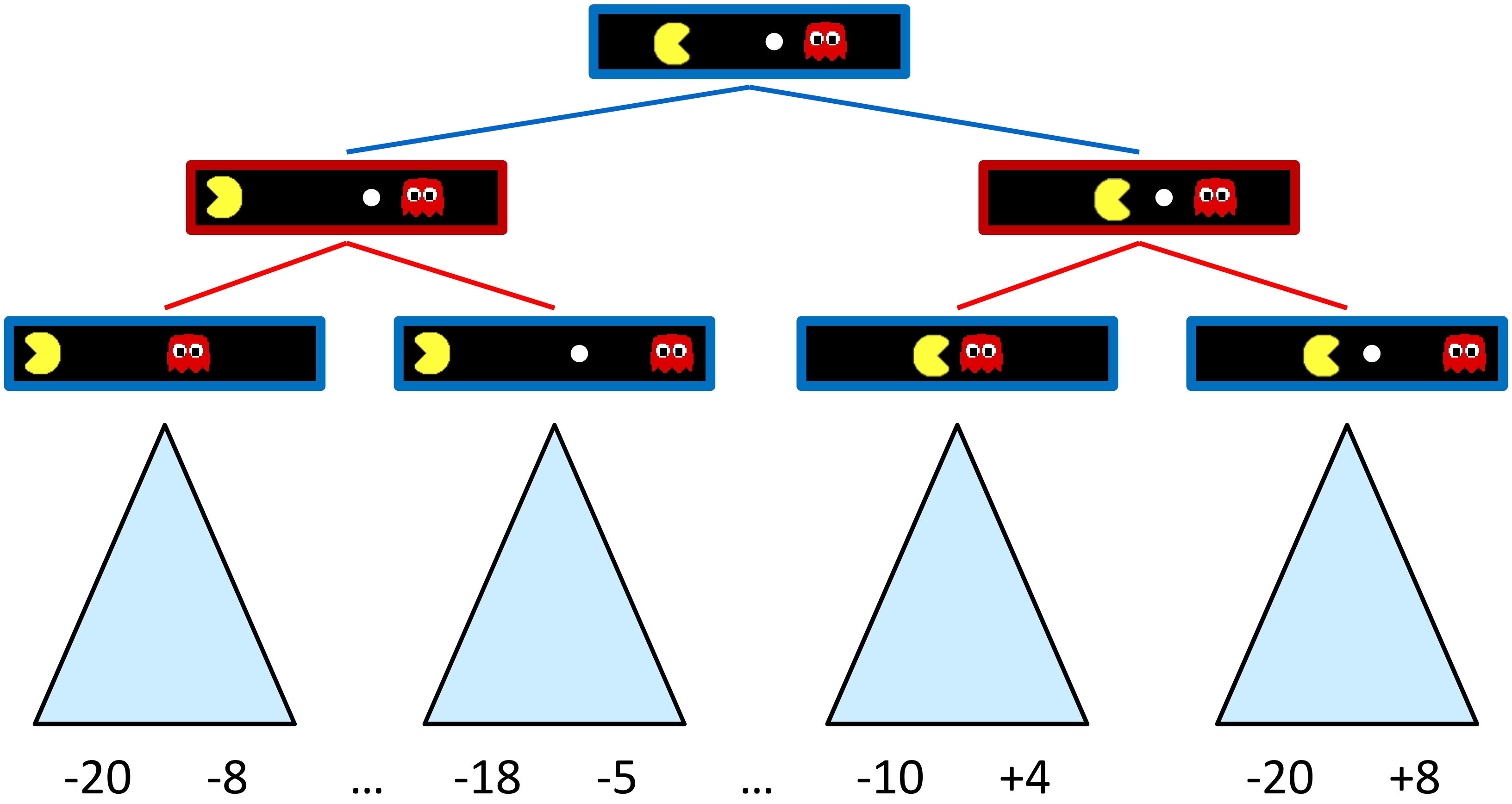
非終端状態:

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

終端状態:

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

敵対的ゲーム木



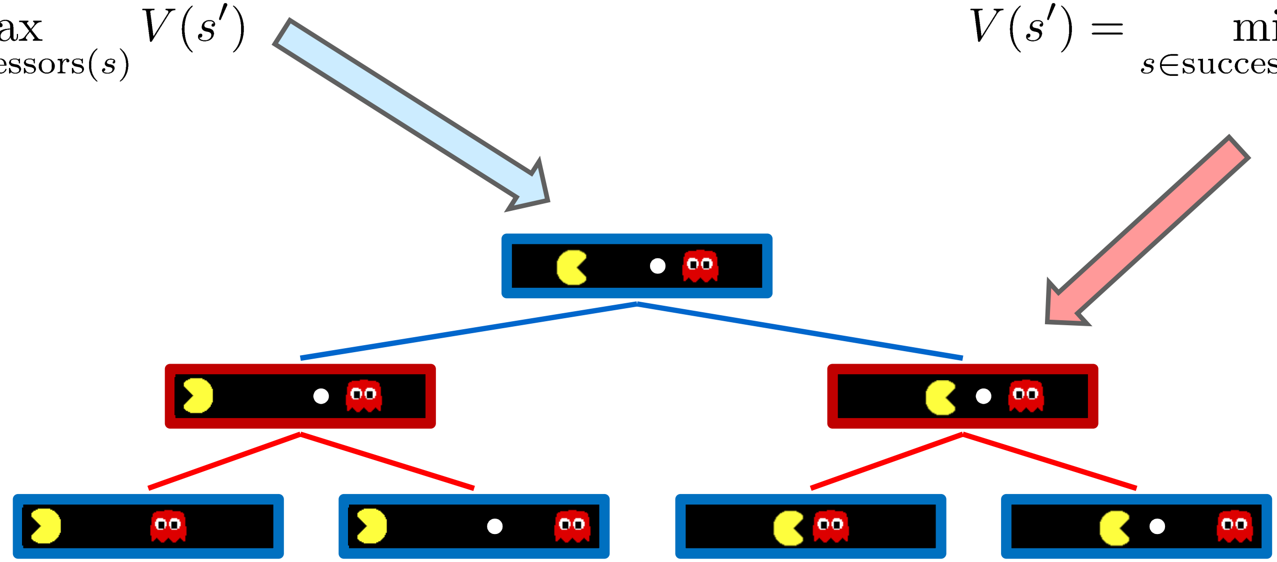
ミニマックス価値

このエージェントが操作する状態：

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

相手が操作する状態：

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

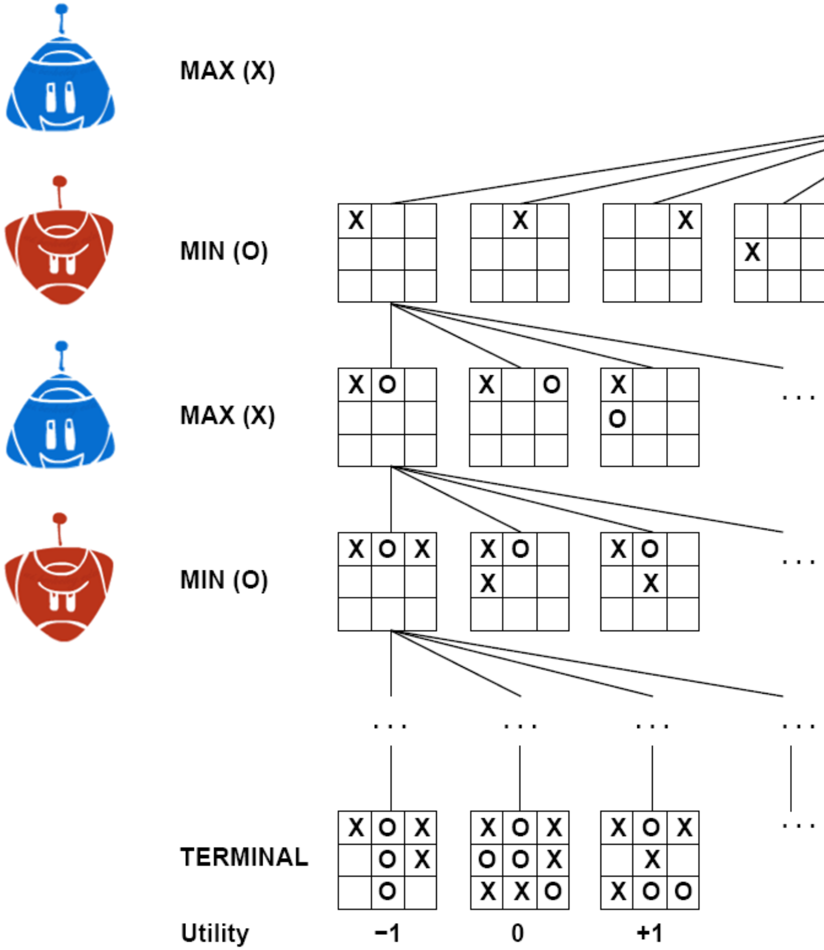


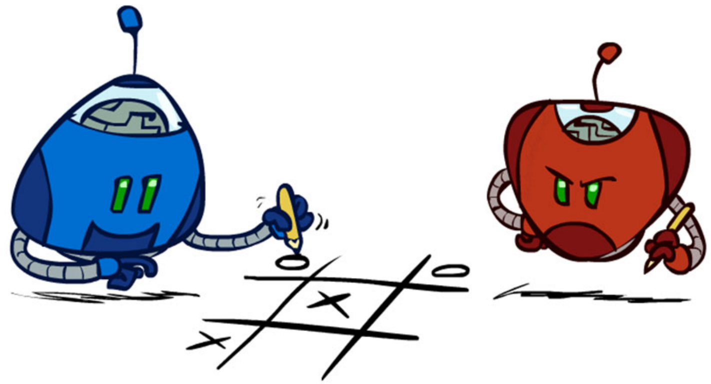
-8 -5 -10 +8

終端状態:

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

３目並べ (Tic-Tac-Toe) のゲーム木



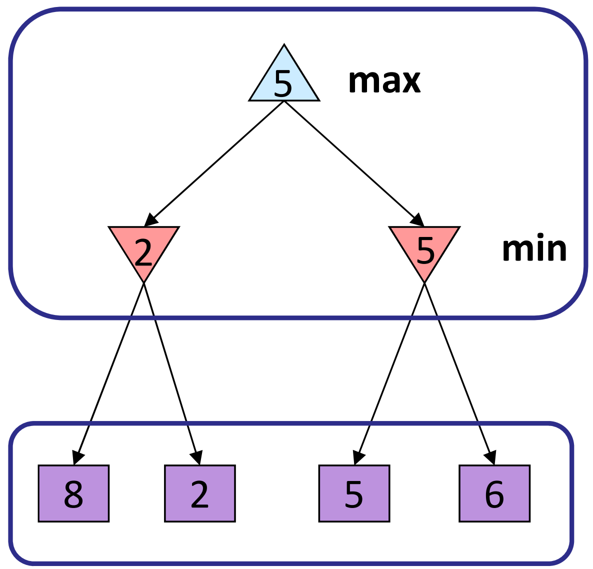


敵対探索 (ミニマックス法)

- 決定的, ゼロサムゲーム:
  - ３目並べ、チェス、チェッカー
  - あるプレイヤーが結果の最大化を目指す
  - 他者は最小化を目指す

- ミニマックス探索:
  - 状態空間の探索木
  - プレイヤーが交互の順番で行動
  - 各節点のミニマックス値を計算：
    合理的で最適な行動をする敵に対してとりうる最良の結果

ミニマックス価値:
再帰的に計算



終端価値:
ゲームによって定まる

ミニマックス法の実装

```
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
```

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

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

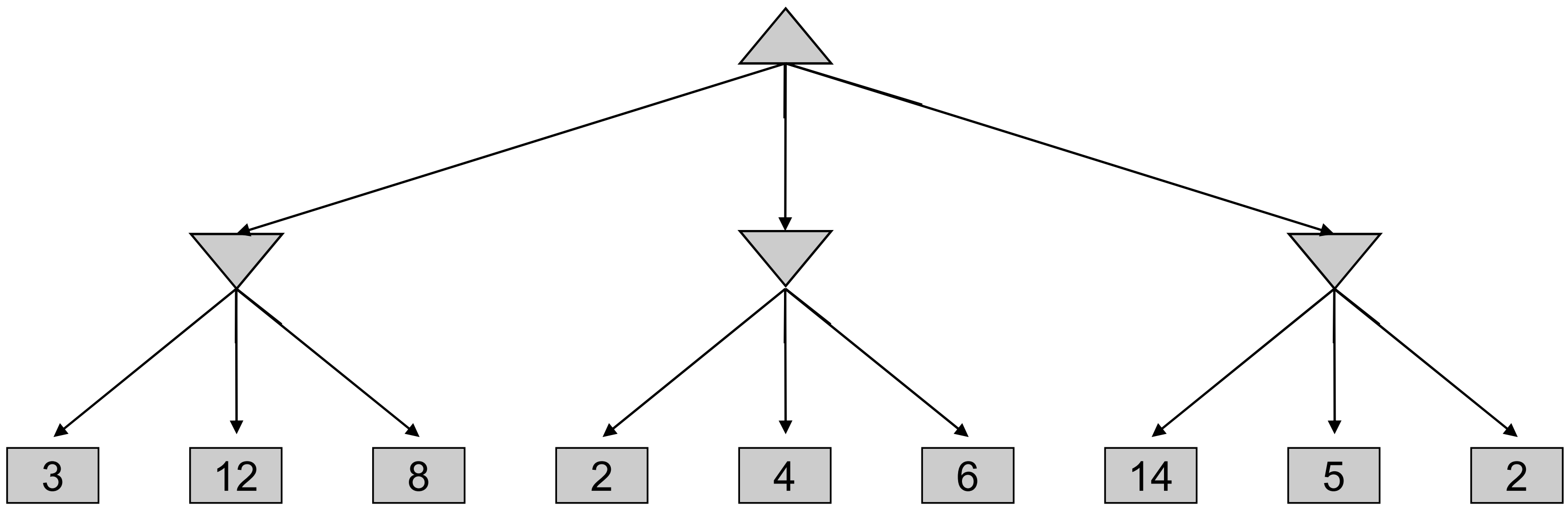
ミニマックス法の実装 (より一般的に)

```
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)
```

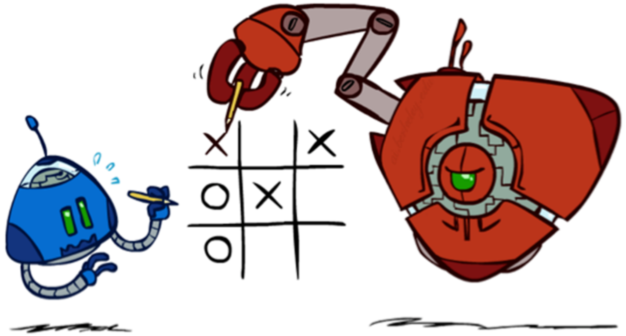
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

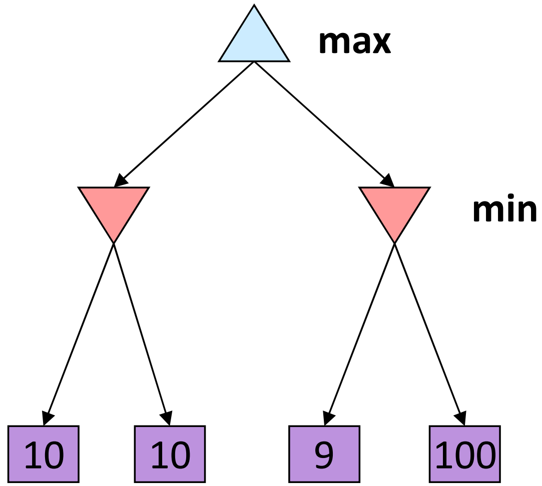
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

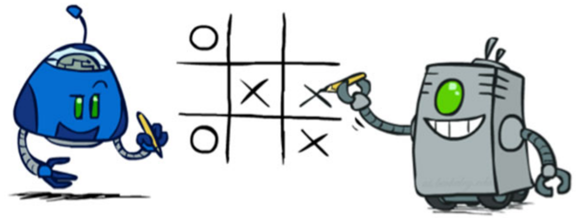
ミニマックス法の例



ミニマックス法の特徴



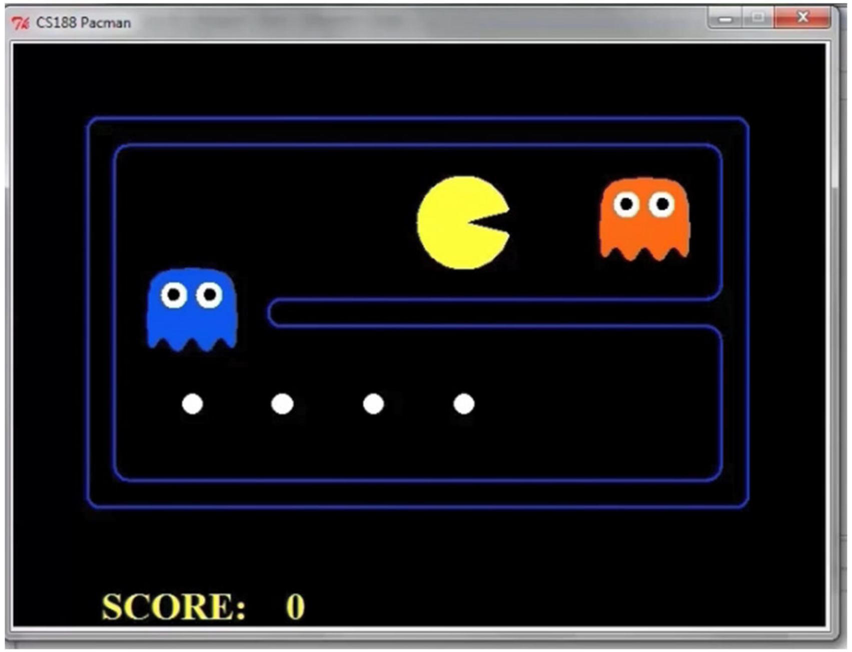




完全なプレイヤーに対する最適行動
では、相手が最適行動をしなければ？

ミニマックス法による行動例

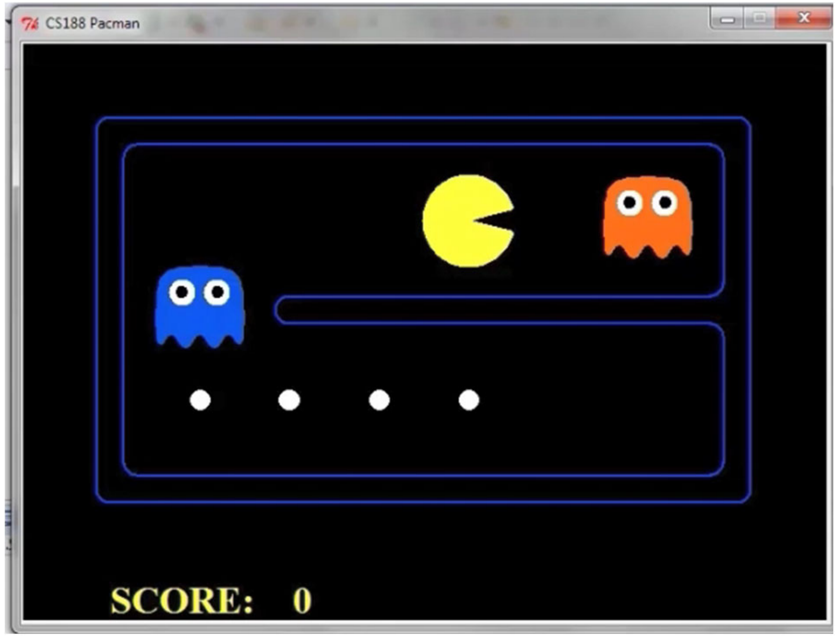
「相手が最適行動する」と考えるならば、どのような行動が最適か？



フードを食べるとスコアが増え、すべて食べたら勝ち。ゴーストに食べられたら負け、時間がたつごとにスコアは減る

期待マックス法による行動例（次週）

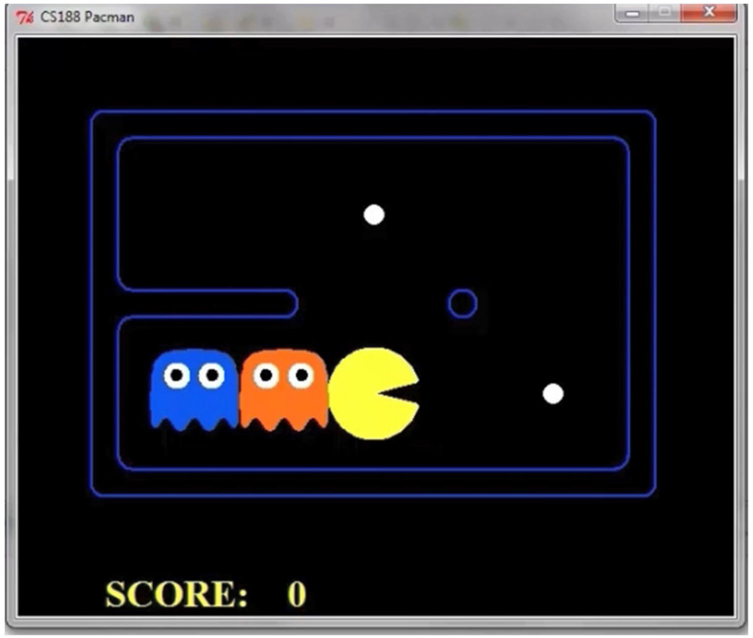
「相手がランダム行動する」と考えるならば、どのような行動が最適か？



フードを食べるとスコアが増え、すべて食べたら勝ち。ゴーストに食べられたら負け、時間がたつごとにスコアは減る

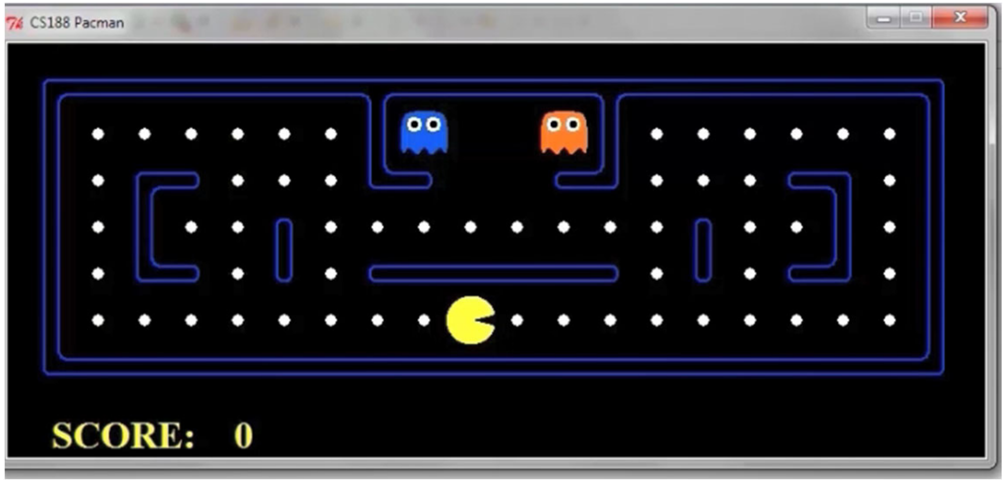
ミニマックス法による行動例

ゴースト側がミニマックス法によりプランニングしている場合。それぞれのゴーストは独立にプランニングしているが、結果として協調が起きている。



ミニマックス法による行動例

ゴースト側がミニマックス法によりプランニングしている場合。それぞれのゴーストは独立にプランニングしているが、結果として協調が起きている。



ミニマックス法の特徴

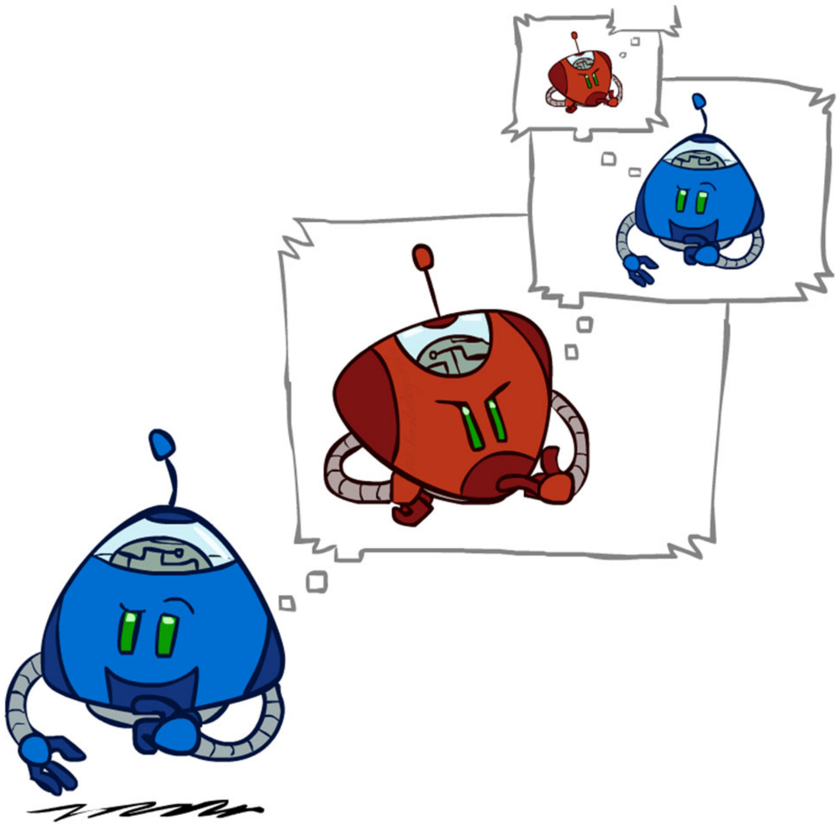
- ミニマックス法はどの程度の効率か？
  - (徹底的な) 深さ優先探索と同じ

- 完全性：解があれば停止.
  （ゲーム木が有限の場合）

- 最適性: 最適な解が見つかる.
  （相手も最適行動する場合）

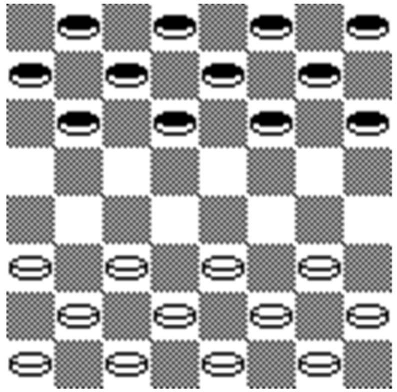
- 計算量: $O(b^m)$
  但し, bは分岐数, mは探索木の最も深い深さ.

- 記憶量: $O(bm)$（深さ優先探索と同じ）



二人ゲームの状態空間

- Tic-tac-toe（〇×ゲーム）の状態空間は約10^5 節点程度
- チェッカの状態空間は約 5×10^{20} 個の節点を含む.
  - 何ダースものコンピュータを使って20年弱の間にわたって探索を続け、2007年に完全に解かれた (双方が最善を尽くすと引き分けになる)
  - この時点で、解かれた最大のゲーム https://webdocs.cs.ualberta.ca/~chinook/project/

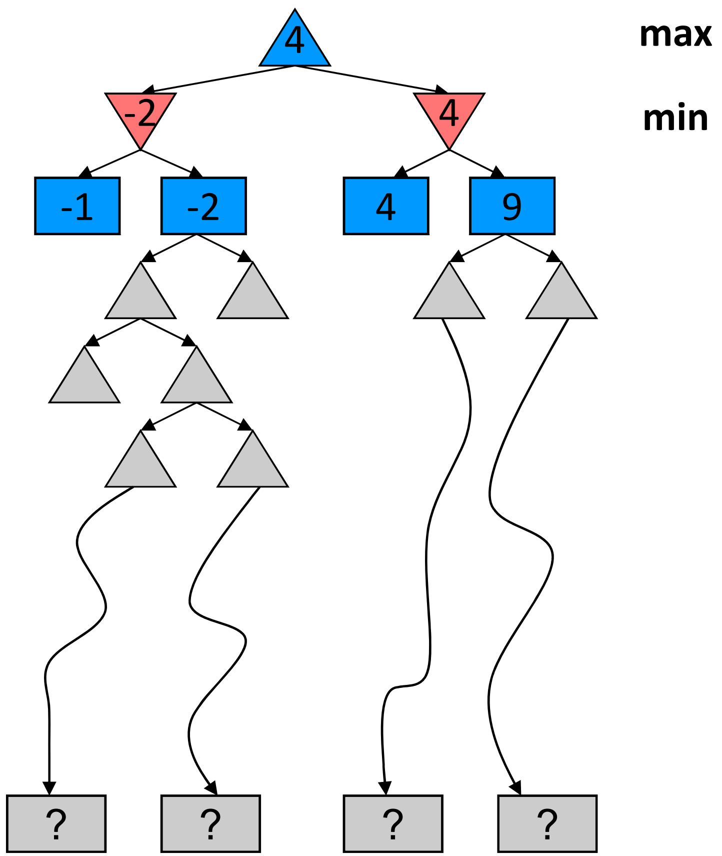


- チェスは、全体では 10^{120} 程度の状態空間だと言われている（$b \approx 35$, $m \approx 100$）
  - 完全解を求めるのはほぼ不可能
  - しかし、木を完全に探索することが必要なのだろうか？

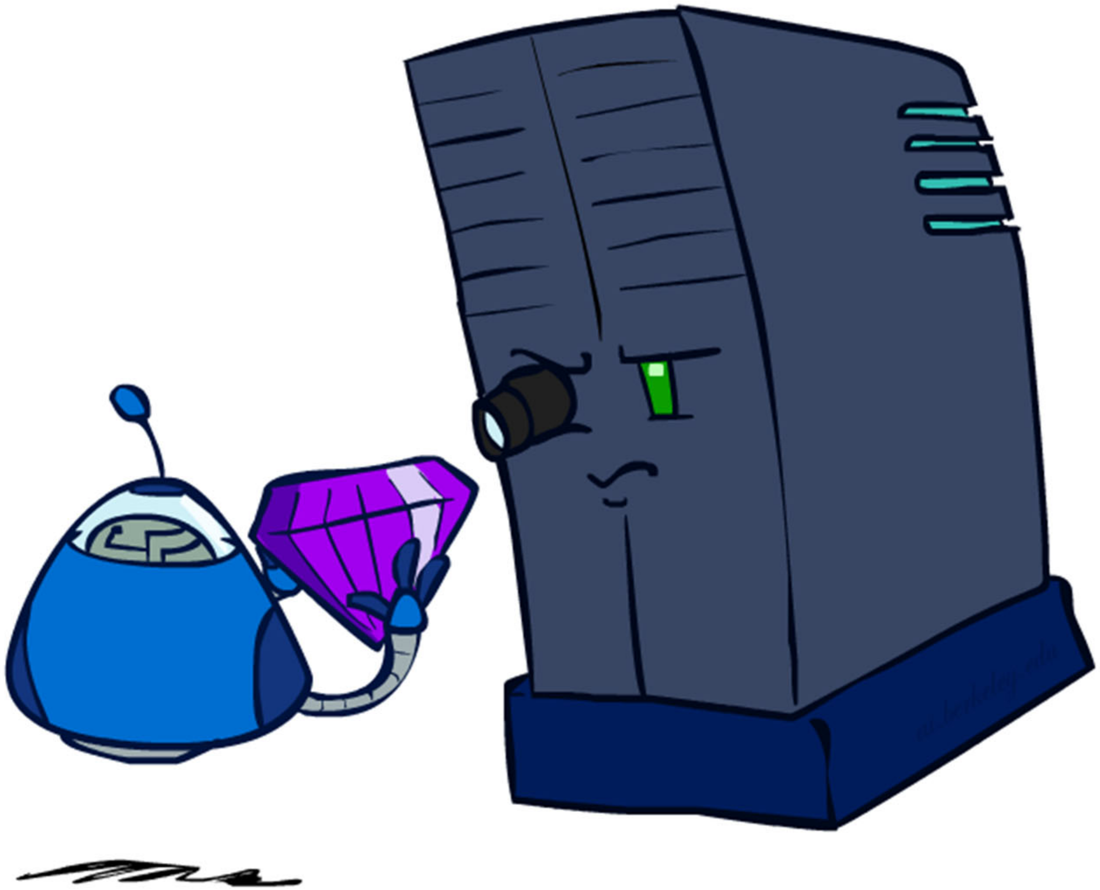
リソースの限界

リソースの限界

- 問題: 現実的なゲームでは, 葉節点まで探索できない!
- 解法: 深さを限った探索
  - 木の深さを制限して、探索を行う
  - 終端効用を、非終端節点での評価関数に置き換え
- 例:
  - たとえば 100 秒あれば, 毎秒1万節点を探索できるとする
  - １手ごとに100万節点が探索できる
  - チェスで8手先読み – そこそこの腕のプレイヤー
- 最適なプレイヤーという保証は成り立たない
- より多くの先読みができるかは「大きな」違い
- 即時性を持たせるために、反復深化法を利用

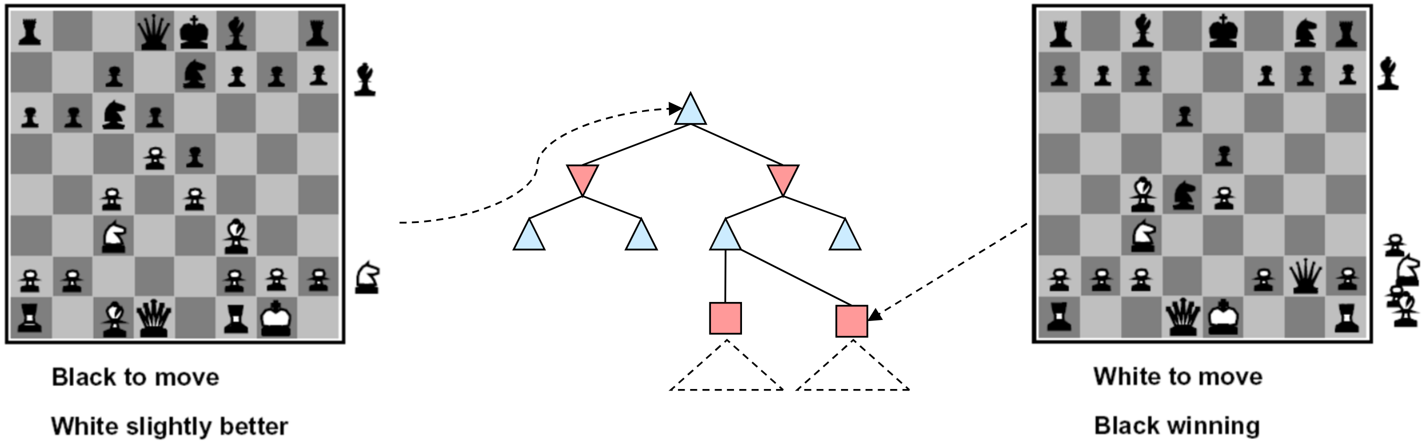


評価関数



評価関数

- 評価関数は深さを限った探索において、非終端節点をスコア付けする



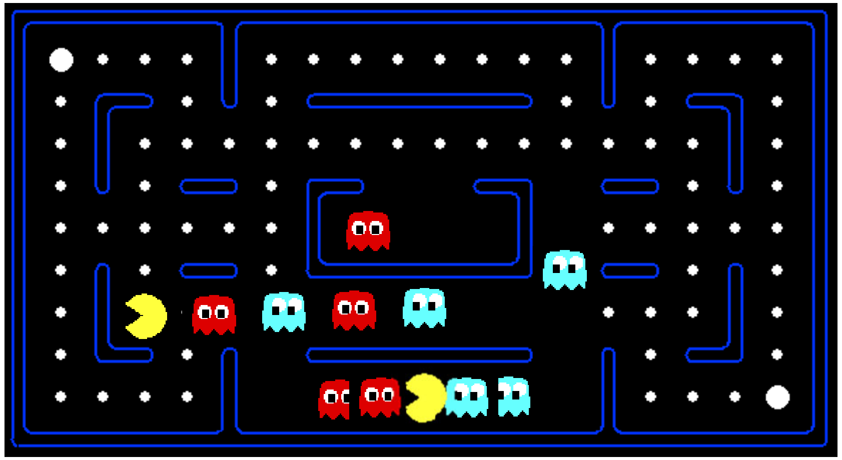
- 理想的には: その節点の本当のミニマックス値を返す関数
- 実際的には: 複数の特徴の重みづけ線形結合：

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

- 例： $f_1(s)$ = (白のクイーンの数– 黒のクイーンの数), etc.

どういう評価関数を作ればよいか？

以下の状態、それぞれ比較すると？　今の状態と先ほどの状態、どっちが良い？

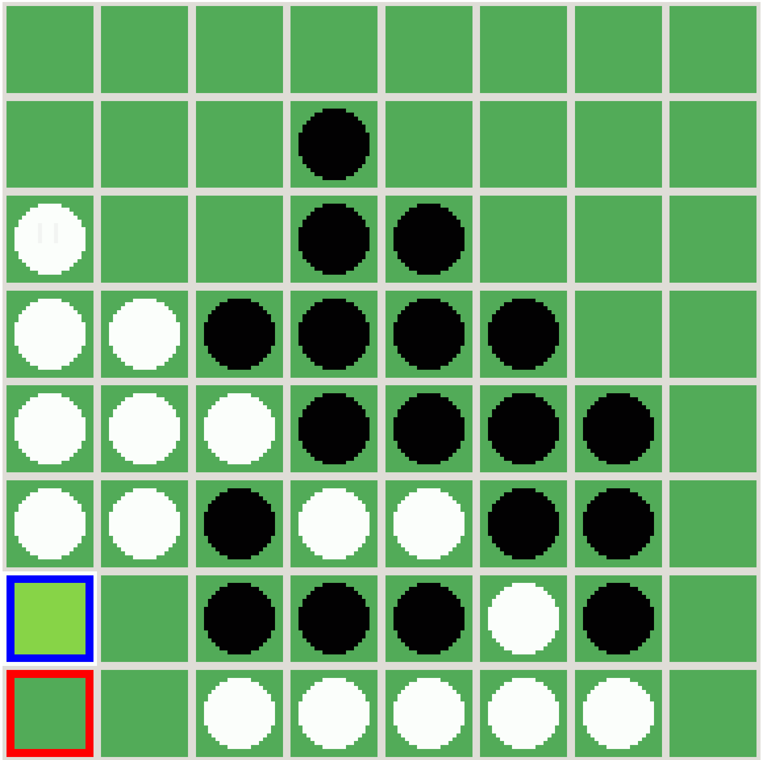


参考：将棋の駒の価値の評価関数

https://ja.wikipedia.org/wiki/%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB:Shogi_board_pieces_and_komadai.jpg

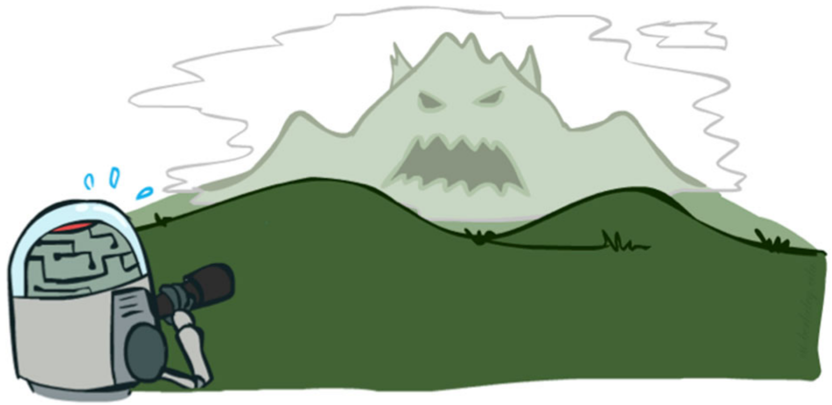
飛	15点	竜	17点
角	13点	馬	15点
金	9点		
銀	8点	成銀	9点
桂	6点	成桂	10点
香	5点	成香	10点
歩	1点	と	12点

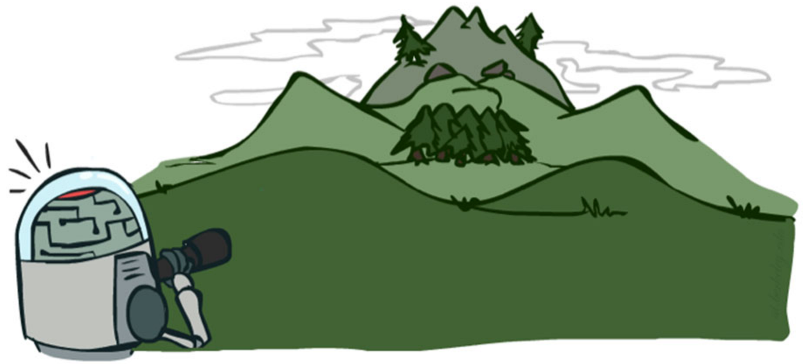
参考：評価関数



深さの重要性

- 評価関数は常に不完全（e.g. チェスや将棋、何が正しい評価関数か？）
- 木が深いと評価関数の影響は埋もれるので、評価関数の質はあまり重要でなくなる
- 特徴の複雑さと、計算量の複雑さにはトレードオフがある





深さを限った探索　（深さ＝２）

考えてみよう：（何らかの評価関数を利用した上で）２手だけ先を読む
→どういう行動をとることになるか？

深さを限った探索　（深さ＝10）

考えてみよう： ずっと先の手を読めるなら？どういう行動をとることになるか？

水平線効果

- 先読み手数でカットオフする探索では, 葉節点(水平線)の先は分からない. 高い評価値の向うに低い評価値の節点が存在することを, 水平線効果(horizontal effect)と呼ぶ.
- 特に、探索が浅いときに起きやすい。

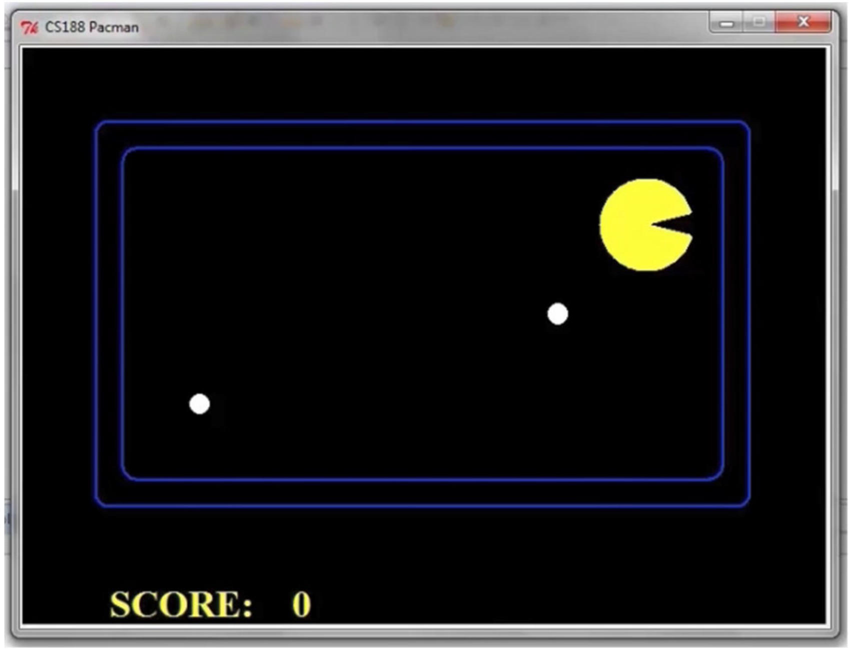
- 水平線効果の例
  - 「将棋で、不利になると、次々に自分の駒を捨てだす」といった現象

    “探索範囲が２手であるとしよう。いま自分の角がとられそうになっているとき、相手の飛車の頭に歩を打つとする、そうすると、「歩を打つ」「とり返す」で 2 手消費されるため、自分の角がとられる状況が探索範囲の外にでてしまうのである。”

鶴岡慶雅. (2003). ゲーム情報学: 2. ゲーム情報学研究の事例 2.1 将棋. 情報処理, 44(9), 900-904.

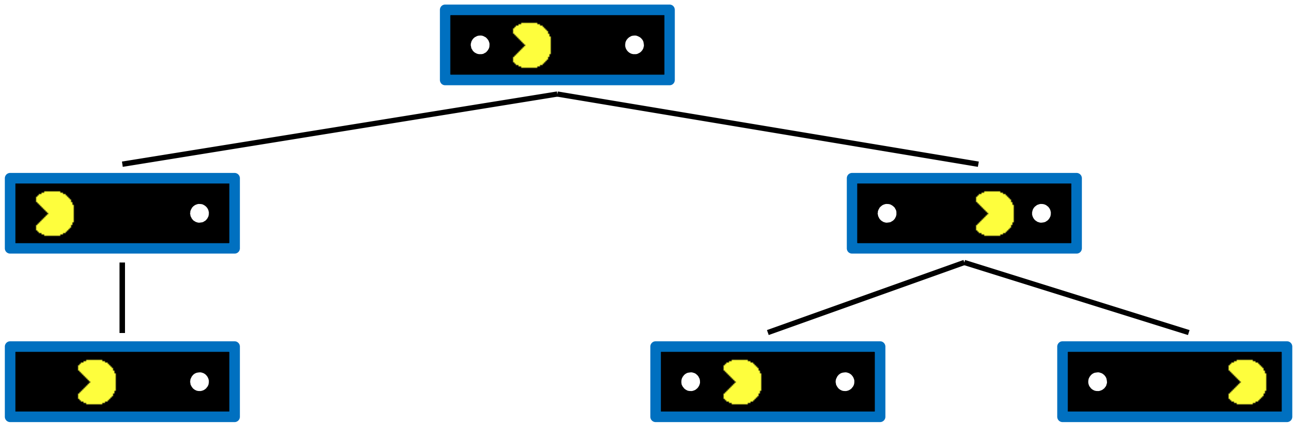
スラッシング(Thrashing)問題 (d=2)

ゴースト無し、フードが２つあるだけ、の場合。評価関数無し。



一般に、評価関数が適切でないと、このような問題が起きる。

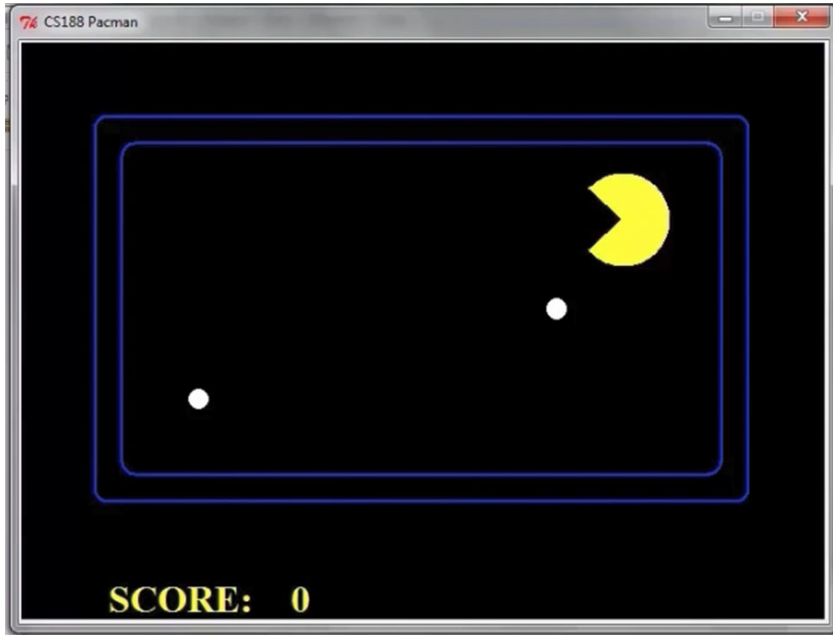
なぜパックマンはずっと食べれないか？



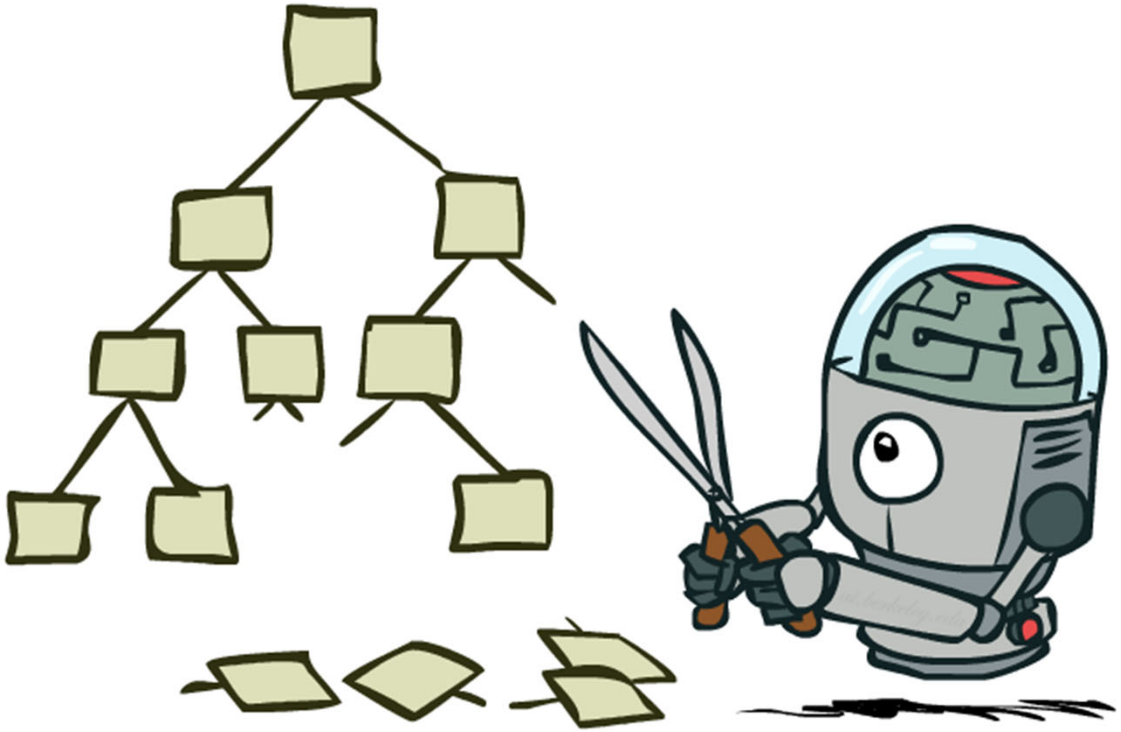
- 再プランニングするエージェントのリスク!
  - 今、フードを食べればスコアが上がることはわかっている（左、右）
  - 後でフードを食べても、同様にスコアが上がることが分かっている（左、右）
  - ドットを食べた後には、得点を得る機会はない（この「水平線」の範囲、深さ２で）
  - そうすると、待っていても、今フードを食べても、どちらも同程度に良い。再プランニングするたびに、右に左に動くことになってしまった。

スラッシング(Thrashing)問題の解決 (d=2)

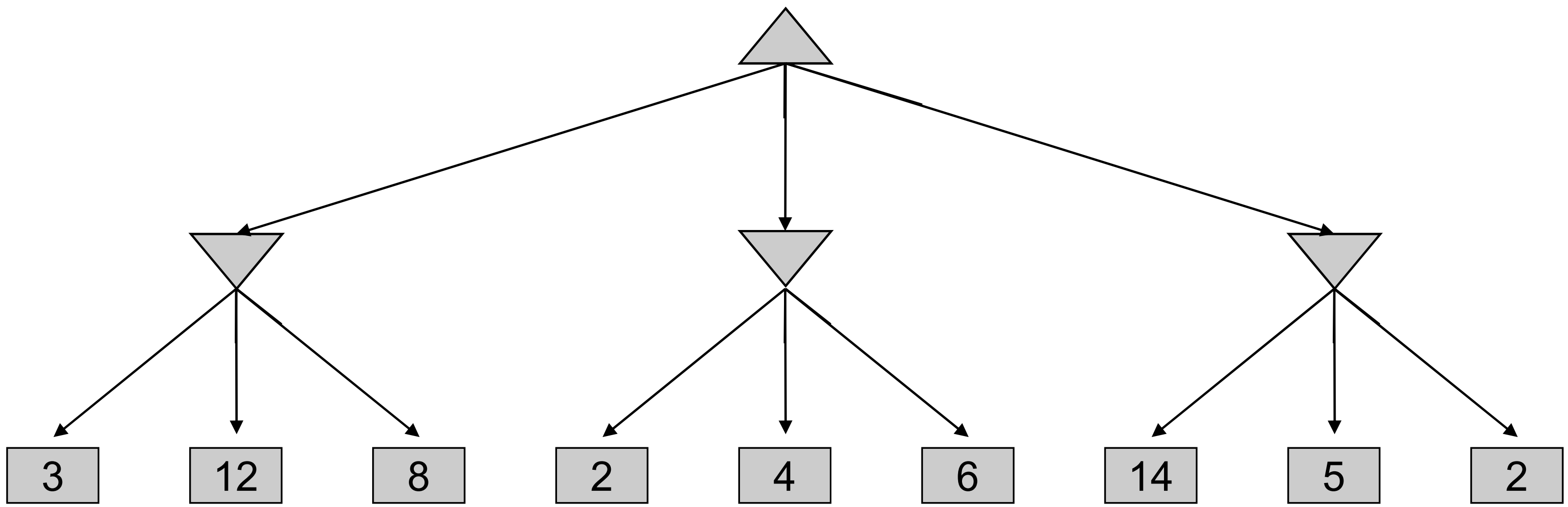
評価関数は、フードに近いほどスコアが高くなるもの。



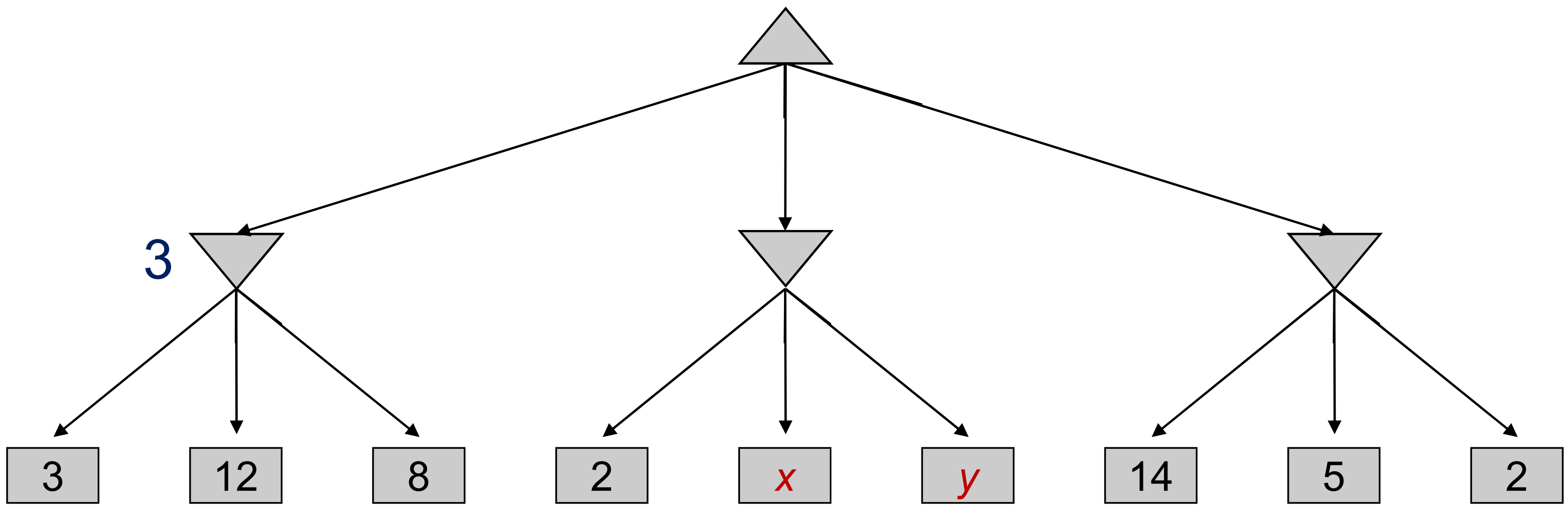
ゲーム木の枝刈り



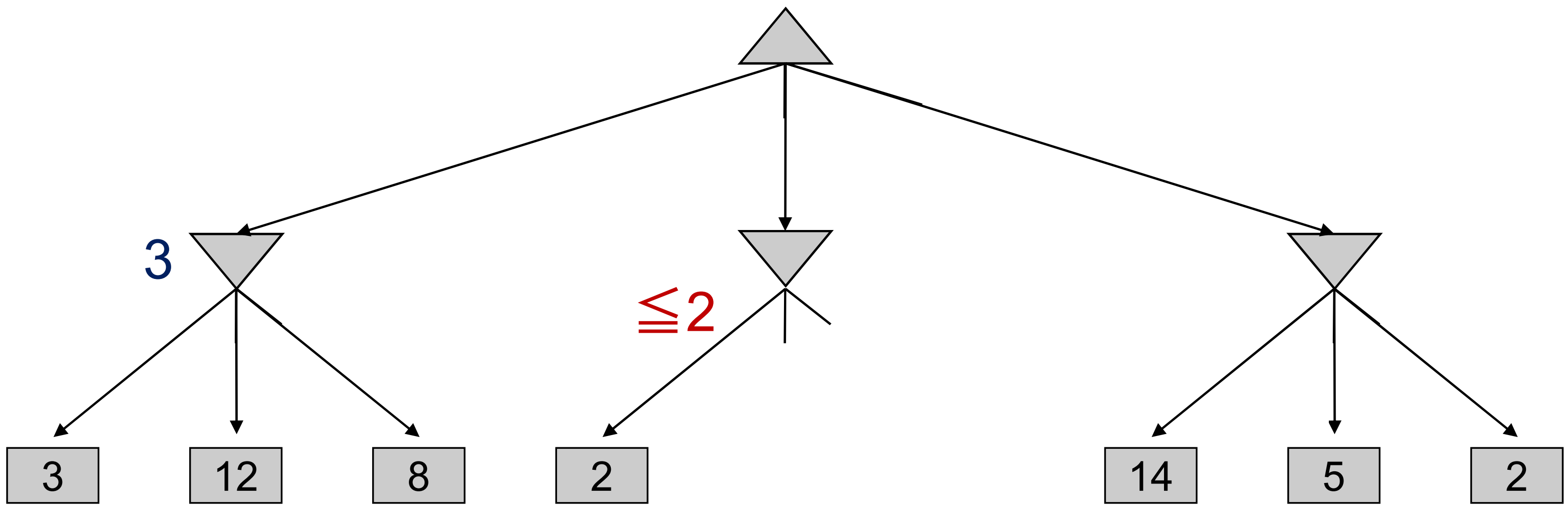
ミニマックス法の例



ミニマックス法の例

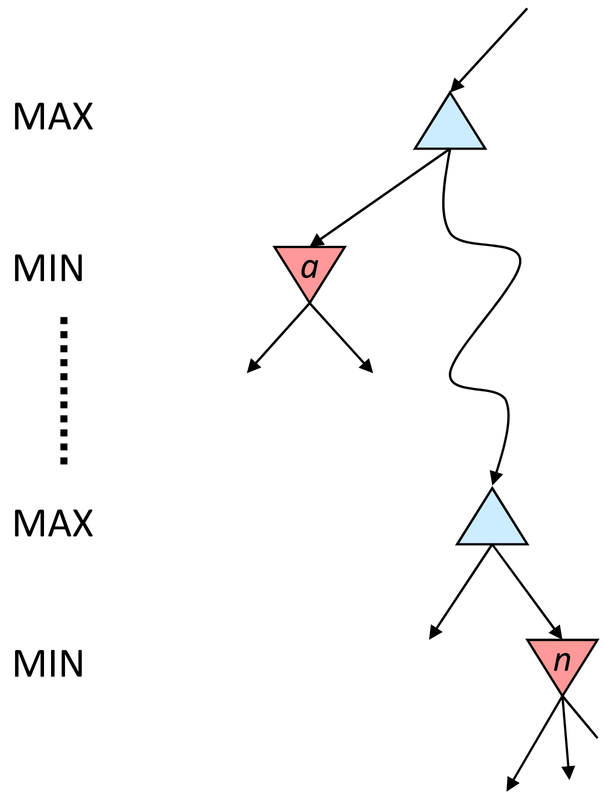


ミニマックス法の枝刈り



アルファベータ枝刈り

- 基本的な考え方 (MIN手番の場合)
  - ある節点 *n* での MIN-VALUE を求めたい
  - *n* の子節点を探索していく
  - *n* の子節点の最小値は減少していく
  - この *n* の値はどの節点に影響するか？　MAX
  - 仮に *α* を MAX が今のパス上で得ることができる最良の値だとする
  - もし *n* の値が *α* より低くなることがわかったら, MAX はこの経路を選ばない。つまり、これ以上、*n* の他の子節点を調べる必要はない



- MAX手番の時も、これと同様

アルファベータ枝刈り　実装

α: MAXにとって今のところ最良の選択肢
β: MINにとって今のところ最良の選択肢

```
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
```

アルファベータ枝刈りの特徴

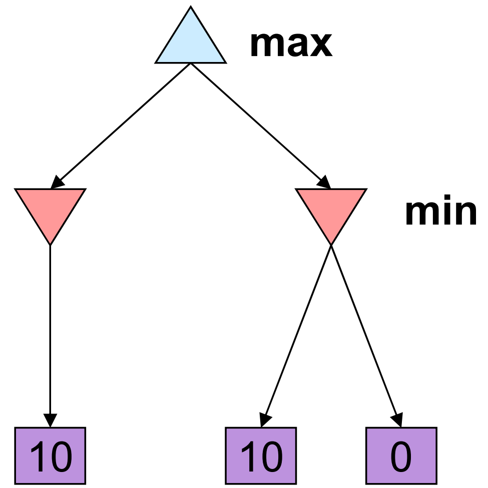
- この枝刈りは根節点でのミニマックス値に影響しない！

- 中間的な節点の値は正しくない場合がある
  - 重要: 根節点の子節点は誤った値となる可能性がある
  - もっともナイーブな処理では行動計画に利用できない（根節点の子節点において処理を始める必要がある）

- 順序：子節点の適切な並べ替えにより性能を改善できる

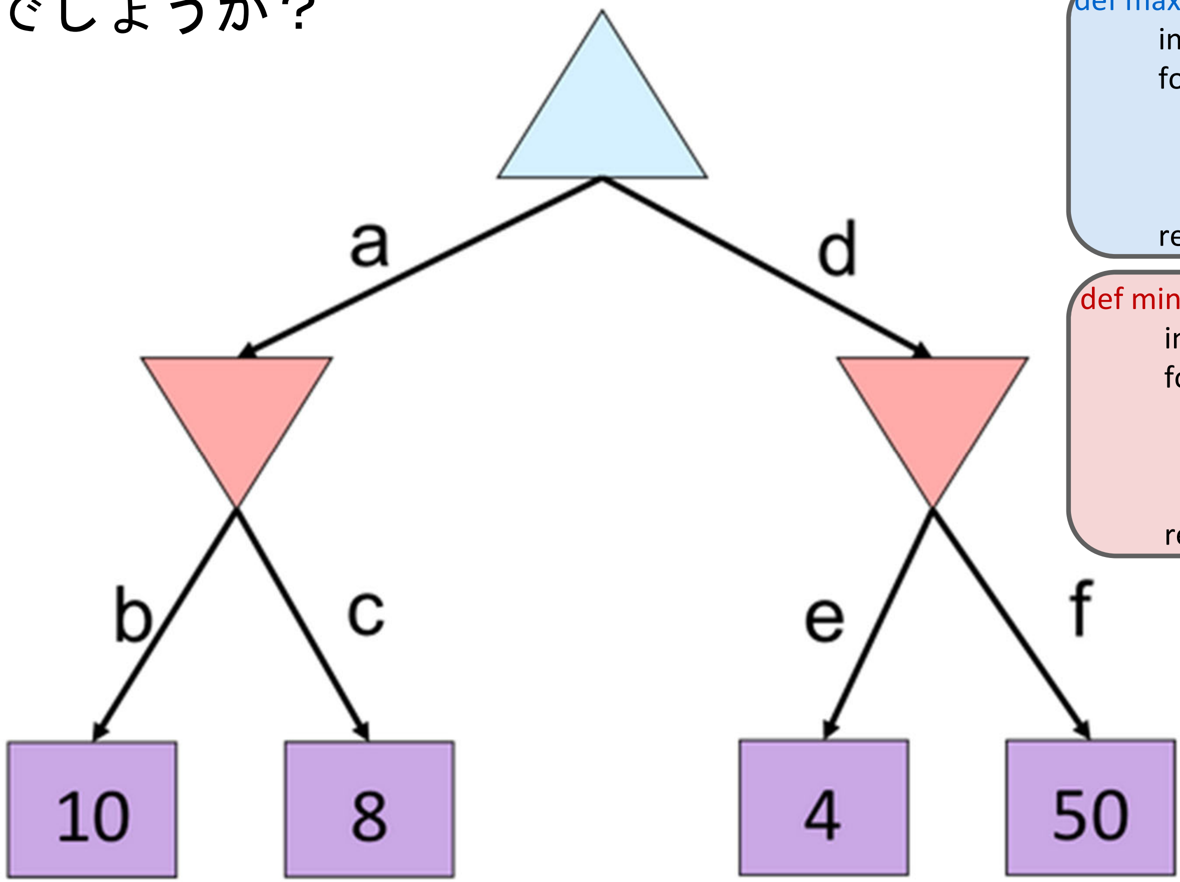
- 「完全な並べ替え」のもとでは:
  - 計算量は $O(b^{m/2})$
  - つまり、２倍の深さまで探索できる!
  - とはいえ、チェスのような問題の完全探索は無理…

- これは、メタ推論 (何を計算すべきかの計算)の例



演習：アルファベータ枝刈り

どの枝が刈られるでしょうか？

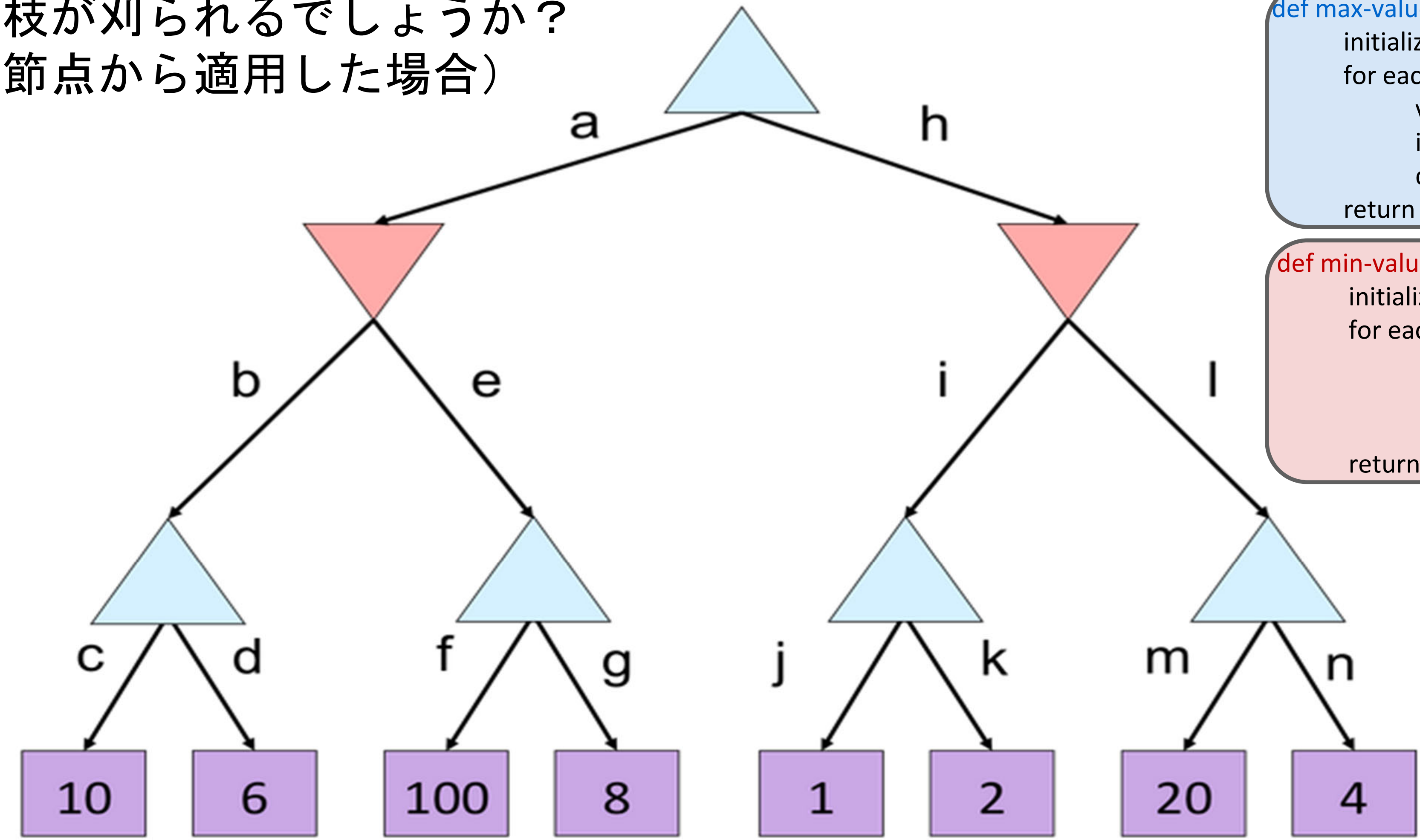



```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
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
```

演習：アルファベータ枝刈り２

どの枝が刈られるでしょうか？
（根節点から適用した場合）




```
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
```

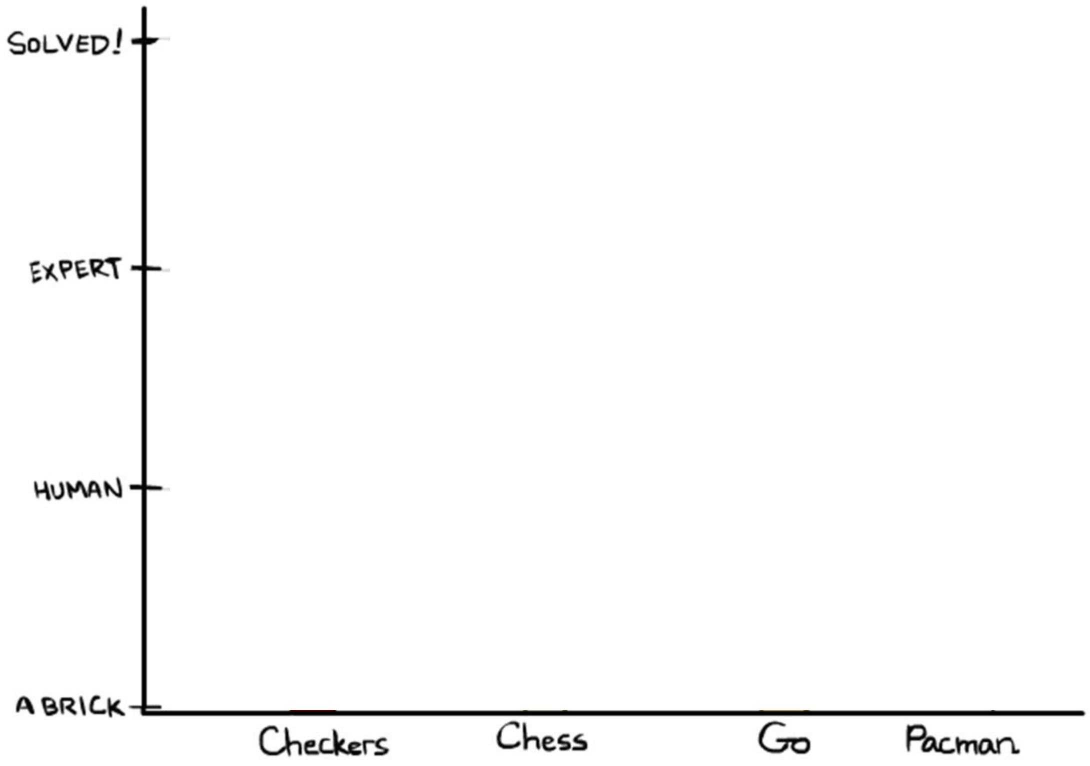
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

発展的話題

これまでの歴史と発展的なテクニック

コンピュータによる二人ゲームの最先端

- **チェッカー(Checkers):** 1950年に最初のコンピュータのプレイヤー。1994年にChinookが40年間チャンピオンを続けたMarion Tinsleyに勝ってチャンピオンに。2007年に完全解が求まった。
- **チェス(Chess):** 1997年にDeep Blueが人間のチャンピオン Gary Kasparovに勝った。Deep Blueは２億局面/秒の評価を行って、最大では40手の先読みを実現した。
- **Go:** 2016年に人間のチャンピオンに勝った。深層学習とモンテカルロ木（ランダム）探索を組み合わせる方法
- **Pacman**



コンピュータと人間の対戦の歴史

1950年　Shannonによるゲーム木の探索, 静的評価関数の導入.

1960年代 始めて対戦可能なプログラムの出現.

知識主導型(静的評価関数で前向き枝刈り).

人間の初級者レベル.

1974年　コンピュータチェス世界選手権の開始.

反復深化,終盤データベース,キラーヒューリスティックスなどを採用.

人間の上級者レベル.

1980年代

専用ハードウェア. 70万局面/秒で評価. 人間の超一流 (Top 100)

1997年

Deep Blueが人間のチャンピオン Kasparov と対戦して勝利.

その後、より難しい、将棋、囲碁、に研究対象が移っていく.

（最終的に、将棋では2013年に、囲碁では2016年に、コンピュータが人間のチャンピオンに勝った）

コンピュータと人間の対戦の歴史

- チェス
  - 分岐数b ≈ 35, 深さm ≈100、探索空間10^{120}
  - 初級者で4手先、上手い人は8手先、チャンピオンは12手以上先読み
  - 盤面の評価がしやすい　（残りの盤面上の駒の価値）
  - 1997年にコンピュータが人間のチャンピオンに勝った
  - IBMの専用ハードウェア Deep Blue (512 CPU)、2億局面/秒、12手先読み
- 将棋
  - 分岐数b ≈ 200, 深さm ≈120、探索空間10^{220}
  - プロは20手～30手先読み　（一部を深読み）
  - 盤面の評価が難しい　（駒の再利用、多様な駒、守りの堅さなど）
  - 2013年に人間のチャンピオンに勝った（クラウド計算1092 CPU＋128 GPU）
- 囲碁
  - 分岐数b ≈ 250、探索空間10^{360}
  - 2016年に人間のチャンピオンに勝った　（クラウド計算1202 CPU+176 GPU）
- 勝因： 計算機の高速化、評価関数の強化学習・独自の定石(自己対戦)、過去の盤面の学習（DNN）

ルックアップテーブル (look-up table) の利用

- 定石データベース
  - 序盤では探索を用いるより知識を用いた方が得策.
    （敵が定石から変化した場合には探索に切替える必要がある.）
- ハッシュ表
  - すでに探索した局面の情報(静的評価値など)を表に記憶. チェスでは400から4000000程度の局面が記憶される.
- キラーヒューリスティックス(killer heuristics)
  - アルファベータ手続きでは見込みのある節点を先に展開した方が効率が良い。そこで、評価が高かった手 (killer move) (敵の指し手を咎める手, refutation move とも呼ぶ) を記録しておき、同様の状況で, 優先的に探索する.
- 終盤データベース
  - 終盤の典型的な局面を探索し尽くし, データベースに格納.
  - 一方が, キング, ビショップ, ビショップで, 他方がキング, ナイトである局面は引き分けだとされていた. しかし, 実は66手で勝負が つくことを発見.

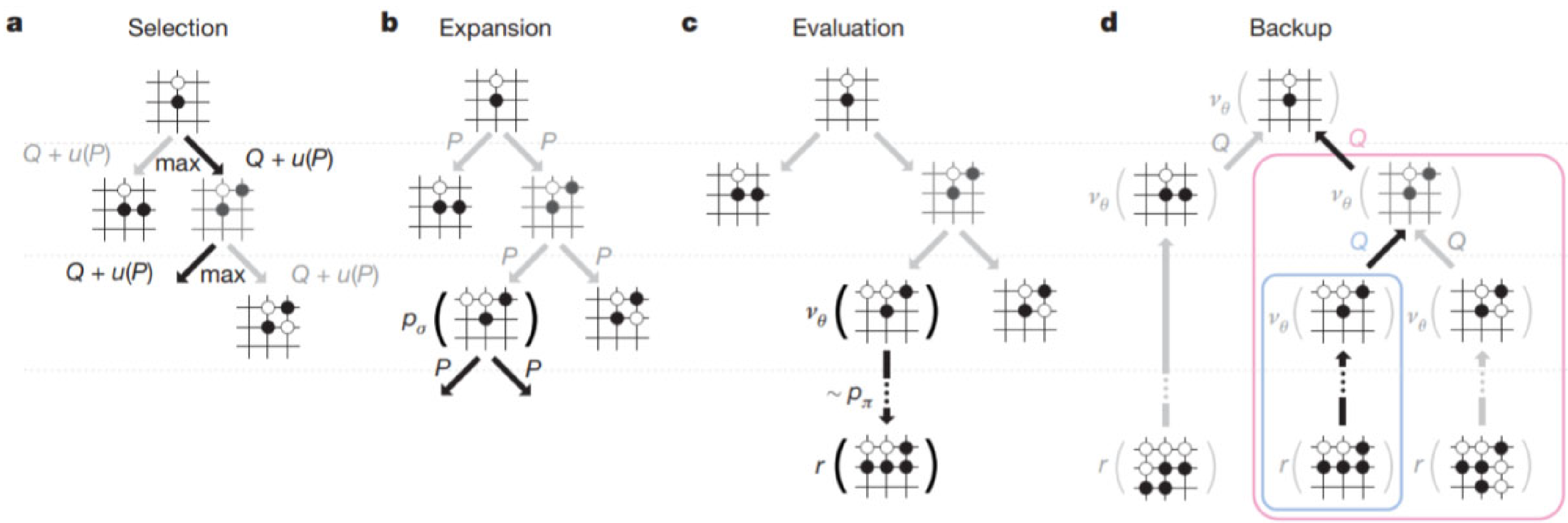
リアルタイム性を考慮した探索

- 静止探索(quiescence search)
  - 駒の取り合いなどの急場は1手深く読む毎に評価値が大きく変化する. 評価値が大きく変化しなくなるまで読む方法を静止探索(quiescence search)と呼ぶ.水平線効果の対策として有効.
- 非凡拡張
  - ある節点(局面)が他の兄弟節点より著しく評価値が高い場合,その節点を非凡(singular)であると言う. 非凡な節点の先を深く読む方法を非凡拡張 (singular expansion)と呼ぶ.
- 時間配分
  - 残り時間を *T* とし, その時間内に指さなければならない手数を *N* とする. $B=T/N$ が 1 手あたりに使える平均時間. まず, 1 手に使う時間の限界を決める (例えば 4*B*). *B*/2 で一旦探索を中止し, 簡単な局面なら終了, そうでなければ *B* まで探索. 水平線効果に陥った場合には2*B*, 4*B* まで探索.

参考

モンテカルロ木探索

- ロールアウト：ランダムに残りの手を打ってみる（有限サイズの木）
- Alpha Go では、より望ましい手を強化学習の考え方を利用し、迅速に有望な手を評価してその周辺を念入りに探索する方法を利用（下図）



Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. *nature*, *529*(7587), 484-489.

まとめ

- 敵対探索: Adversary Search
  - ミニマックス手続き
    ・・・相手の最適行動を仮定　→ AND/ORグラフ探索
  - アルファベータ手続き
    ・・・いかにして探索結果に影響せずに展開節点数を減らすか
  - 評価関数
    ・・・終端節点以外では、局面の良さを数値化．複数の効用を線形結合

展開節点数、何手先まで読めるか、が勝負

議論

- 現状
  - 二人ゲームに勝つ人工知能エージェントができるようになってきた。（二人ゲームを解くアルゴリズムは完成した）
  - 当初の人工知能研究の狙いは、人のような知能を人工的に作る、ということにあった。これは実現できたか？

  ⇔人よりも四則演算が速いコンピュータはずっと前からできていた。これを「知能」だとは考えてなかった。

- 人工知能は実現できたか？　どう思いますか？

引用文献

- S. Russell and P. Norvig, Artificial Intelligence A Modern Approach, 3rd edition, Pearson Education Limited, 2016.
- UC Berkeley CS188 Intro to AI -- Course Materials http://ai.berkeley.edu/