Wskazówki do ćwiczenia 3.

WSI - ćwiczenia 2022Z

Jakub Łyskawa Stanisław Pawlak

10.11.2022

Do czego zastosować algorytm minimax?

- Gry dwuosobowe,

Do czego zastosować algorytm minimax?

- Gry dwuosobowe,
- Deterministyczne

Do czego zastosować algorytm minimax?

- Gry dwuosobowe,
- Deterministyczne
- Znany jest pełen stan gry

Potrzebne informacje o grze

Dla stanu potrzebna:

- Wiedza, kto robi aktualny ruch

Potrzebne informacje o grze

Dla stanu potrzebna:

- Wiedza, kto robi aktualny ruch
- Wiedza, jakie ruchy są możliwe

Potrzebne informacje o grze

Dla stanu potrzebna:

- Wiedza, kto robi aktualny ruch
- Wiedza, jakie ruchy są możliwe
- Wiedza, do jakiego stanu doprowadzi każdy ruch

Potrzebne informacje o grze

Dla stanu potrzebna:

- Wiedza, kto robi aktualny ruch
- Wiedza, jakie ruchy są możliwe
- Wiedza, do jakiego stanu doprowadzi każdy ruch
- Wiedza, czy jest terminalny

Potrzebne informacje o grze

Dla stanu potrzebna:

- Wiedza, kto robi aktualny ruch
- Wiedza, jakie ruchy są możliwe
- Wiedza, do jakiego stanu doprowadzi każdy ruch
- Wiedza, czy jest terminalny

Trzeba móc określić jakość stanu z perspektywy gracza - terminalnego oraz, dla ograniczonej głębokości, dowolnego innego.

Potrzebne informacje o grze

Dla stanu potrzebna:

- Wiedza, kto robi aktualny ruch
- Wiedza, jakie ruchy są możliwe
- Wiedza, do jakiego stanu doprowadzi każdy ruch
- Wiedza, czy jest terminalny

Trzeba móc określić jakość stanu z perspektywy gracza - terminalnego oraz, dla ograniczonej głębokości, dowolnego innego. Im lepszy dla tego gracza, tym większa wartość heurystyki. Największa - dla zwycięstwa (np. $+\infty$). Najmniejsza - dla przegranej (np. $-\infty$).

Algorytm minimax

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if player(node) is maximizingPlayer then
        value := −inf
        for each child of node do
            value := max(value,
                minimax(child, depth − 1, maximizingPlayer))
        return value
    else (* minimizing player *)
        value := +inf
        for each child of node do
            value := min(value,
                minimax(child, depth − 1, maximizingPlayer))
        return value
```

Czyli...

Jeżeli nie można sprawdzać dalej (koniec gry, maksymalna głębokość) - oceniamy obecny stan

- Np. szachy - różnica sum wartości wszystkich figur gracza maksymalizującego i minimalizującego

Czyli...

Jeżeli nie można sprawdzać dalej (koniec gry, maksymalna głębokość) - oceniamy obecny stan

- Np. szachy - różnica sum wartości wszystkich figur gracza maksymalizującego i minimalizującego

Jeżeli ruch ma gracz maksymalizujący - wybiera najlepszy ruch dla siebie, czyli o największej wartości

- Np. szachy - mogąc wybrać:
  - (-3) utratę gońca,
  - (+1) utratę piona i zbicie dwóch pionów, albo
  - (+4) utratę wieży i zbicie królowej

  algorytm wybierze trzeci ruch

Czyli...

Jeżeli nie można sprawdzać dalej (koniec gry, maksymalna głębokość) - oceniamy obecny stan

- Np. szachy - różnica sum wartości wszystkich figur gracza maksymalizującego i minimalizującego

Jeżeli ruch ma gracz maksymalizujący - wybiera najlepszy ruch dla siebie, czyli o największej wartości

- Np. szachy - mogąc wybrać:
  - (-3) utratę gońca,
  - (+1) utratę piona i zbicie dwóch pionów, albo
  - (+4) utratę wieży i zbicie królowej

  algorytm wybierze trzeci ruch

Jeżeli ruch ma gracz minimalizujący - analogicznie, tylko o najmniejszej wartości

Obcinanie $\alpha - \beta$

```
function alphabeta(
    node, depth, a, b, maximizingPlayer) is
  if depth = 0 or node is a terminal node then
    return the heuristic value of node
  if player(node) is maximizingPlayer then
    value := -inf
      for each child of node do
        value := max(value,
          alphabeta(child, depth - 1, a, b,
            maximizingPlayer))
        a := max(a, value)
        if value >= b then
          break (* b cutoff *)
      return value
  ...
```

Obcinanie $\alpha - \beta$ cd.

```
function alphabeta(
    node, depth, a, b, maximizingPlayer) is
  ...
  else
    value := +inf
    for each child of node do
      value := min(value,
        alphabeta(child, depth - 1, a, b,
          maximizingPlayer))
      b := min(b, value)
      if value <= a then
        break (* a cutoff *)
    return value
```

Czyli...

Zapamiętywana jest najlepsza wartość osiągana przez każdego z graczy: α dla maksymalizującego, β dla minimalizującego.

Czyli...

Zapamiętywana jest najlepsza wartość osiągana przez każdego z graczy: α dla maksymalizującego, β dla minimalizującego.

Jeżeli gracz miałby móc dostać większy wynik, niż przeciwny może otrzymać, to przeciwnik mu nawet nie umożliwi tego ruchu.

- np. szachy - jeżeli obecnie analizowany ruch pozwala nam zbić królową kosztem piona ($\alpha = +8$), a przeciwnik w poprzednim ruchu może do tego nie dopuścić i stracić tylko swojego piona ($\beta = +1$), to nie wybierze ruchu który doprowadza do obecnie analizowanej sytuacji - nie ma sensu jej dalej analizować

Użycie

Każdy możliwy ruch ocenić algorytmem minimax/alphabeta, aby znaleźć najlepsze możliwe.

Użycie

Każdy możliwy ruch ocenić algorytmem minimax/alphabeta, aby znaleźć najlepsze możliwe.

Początkowe wartości: $\alpha = -\infty$, $\beta = +\infty$

Użycie

Każdy możliwy ruch ocenić algorytmem minimax/alphabeta, aby znaleźć najlepsze możliwe.

Początkowe wartości: $\alpha = -\infty$, $\beta = +\infty$

Uwaga - nie we wszystkich grach ruchy wykonuje się naprzemiennie.