Politechnika Poznańska
Instytut Informatyki

Adam Meissner
Adam.Meissner@put.poznan.pl
http://www.man.poznan.pl/~ameis

SZTUCZNA INTELIGENCJA

Gry dwuosobowe

Literatura

[1] Pawlewicz J., *Techniki sztucznej inteligencji w programach grających*, (dostęp: 15.05.2020) https://www.mimuw.edu.pl/~pan/gry.pdf

[2] Sterling L., Shapiro E., *The Art of Prolog*, The MIT Press, 1999.

[3] Gdr@en.wikipedia, *Tic-tac-toe-game-tree.svg*, https://en.wikipedia.org/wiki/File:Tic-tac-toe-game-tree.svg, (dostęp: 15.05.2020)

[4] Jezz9999@en.wikipedia, *AB pruning.svg*, http://pl.wikipedia.org/wiki/Plik:AB_pruning.svg (dostęp: 15.05.2020)

Gry dwuosobowe (1)

Założenia wstępne

Rozpatruje się gry dwuosobowe, deterministyczne, z doskonałą informacją o tzw. sumie zerowej. Obowiązują przy tym następujące zasady ogólne:

- w grze bierze udział 2 przeciwników (uczestników lub inaczej – graczy)
- rozgrywka polega na naprzemiennym wykonywaniu przez uczestników posunięć (ruchów) zmieniających stan gry, aż do uzyskania stanu końcowego, który jest remisowy lub zwycięski dla jednego z graczy
- w zasadach wykonywania ruchów nie występuje żaden element „niezależny” od gracza, np. element losowy (jak rzut kostką)
- każdy ruch oraz każdy stan gry są jawne dla obu graczy
- zysk jednego z graczy jest równy stracie gracza drugiego

Strategie rozgrywki

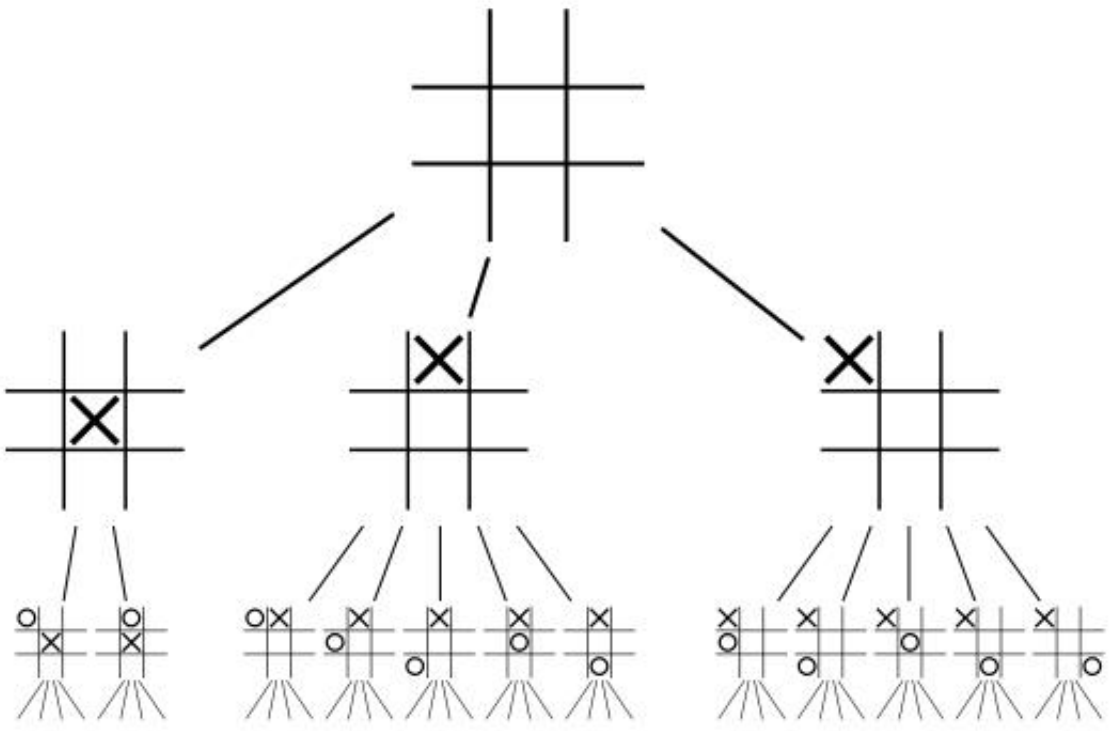
- dla każdej gry rozpatrywanego typu istnieje metoda prowadzenia rozgrywki, czyli strategia gwarantująca jednemu z uczestników wygraną lub remis
- wśród metod prowadzenia rozgrywki wyróżnia się *strategie ogólne*, które można zastosować do dowolnej gry dwuosobowej oraz *strategie specyficzne* (charakterystyczne) dla konkretnej gry
- najczęściej przyjmuje się, że modelem rozgrywki jest tzw. drzewo gry, a prowadzenie rozgrywki polega na wyszukiwaniu odpowiednich ścieżek w tym drzewie

Gry dwuosobowe (2)

Drzewo jako model rozgrywki

1. Modelem rozgrywki jest drzewo, którego wierzchołki reprezentują stany gry; dla dowolnych stanów *s* oraz *s*', stan *s* jest połączony krawędzią ze stanem *s*' wtw gdy istnieje posunięcie (ruch) prowadzące od stanu *s* do stanu *s*'.
2. W toku rozgrywki, każdy gracz stara się znaleźć ścieżkę prowadzącą od stanu początkowego do stanu zwycięskiego lub, o ile to niemożliwe, do stanu remisowego.
3. Każdemu wierzchołkowi drzewa można przyporządkowaną wartość tzw. funkcji oceny f: $S \rightarrow \mathbb{R}$. Niech *s* będzie dowolnym wierzchołkiem wewnętrznym i niech *S'* będzie zbiorem wszystkich wierzchołków potomnych wierzchołka *s*. Jeżeli dla każdego wierzchołka w zbiorze *S*' istnieje dokładnie 1 wierzchołek o ekstremalnej wartości funkcji oceny, to funkcja oceny jest *doskonała*.

Przykładowy fragment drzewa gry ([3])



Gry dwuosobowe (3)

Schemat programu realizującego rozgrywkę [2]

Przyjmuje się, że w grze uczestniczy człowiek (*human*) oraz maszyna (*computer*). Program, w ramach prowadzenia rozgrywki, realizuje strategię gracza *computer*, opisaną przez predykat *evaluate_and_move*.

```
play :–
    initialize(State,Player),
    display(State),
    play(State,Player).

play(State,Player) :–
    final_state(State,Player,Result), !,
    announce(Result).
play(State,Player) :–
    choose_move(State,Player,Move),
    move(Move,State,State1),
    display(State1),
    next_player(Player,Player1), !,
    play(State1,Player1).

choose_move(State,computer,Move) :–
    findall(M,move(State,M),Moves),
    evaluate_and_choose(Moves,State,(nil,min),Move).
choose_move(State,human,Move) :– ask(State,Move).
```

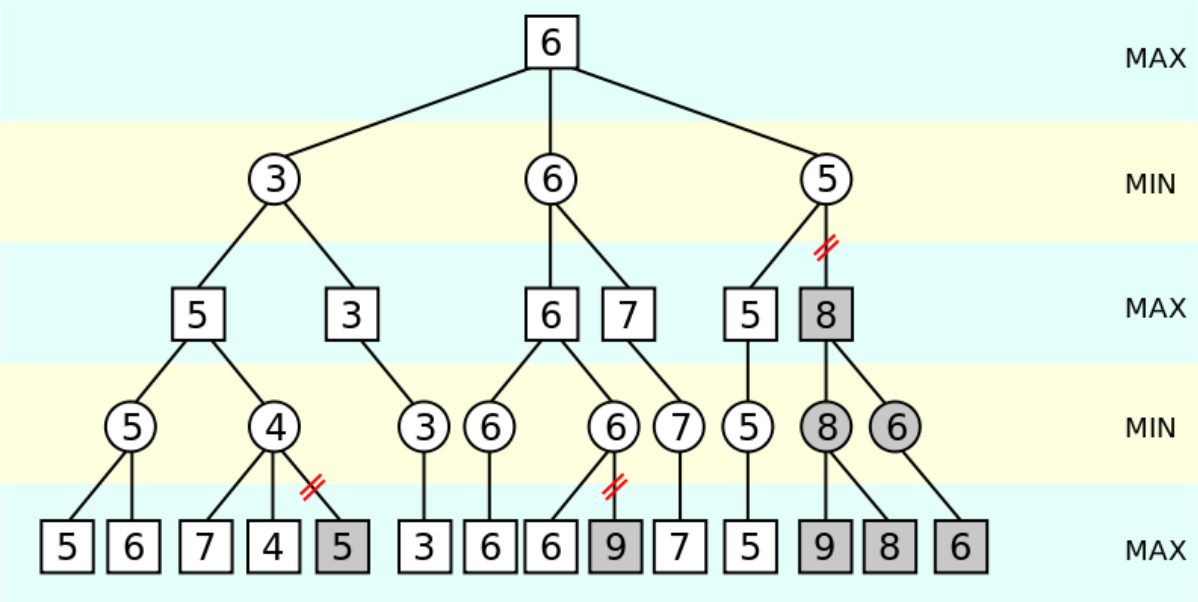
Gry dwuosobowe (4)

Wybór 1-go ruchu w przód z doskonałą funkcją oceny [2]

```
evaluate_and_choose([M|Ms],State,Record,BestMove) :–
    move(M,State,State1),
    value(State1,Value),
    update(M,Value,Record,Record1),
    evaluate_and_choose(Ms,State,Record1,BestMove).
evaluate_and_choose([],_,(Move,_),Move).

update(Move,Val,(BestMove,BestVal),(BestMove,BestVal)) :–
    BestVal >= Val, !.
update(Move,Val,(BestMove,BestVal),(Move,Val)) :–
    BestVal < Val.
```

Przykładowe drzewo gry ([4])



Gry dwuosobowe (5)

Strategia minimaksowa [2]

```
evaluate_and_choose([M|Ms],State,D,MaxMin,Record,Best) :–
   move(M,State,State1),
   minmax(D,State1,MaxMin,_,Value),
   update(M,Value,Record,Record1),
   evaluate_and_choose(Ms,State,D,MaxMin,Record1,Best).
evaluate_and_choose([],_,_,_,Record,Record).

minmax(0,State,MaxMin,_,Value) :–
   value(State,V),
   Value is V * MaxMin.
minmax(D,State,MaxMin,Move,Value) :–
   D > 0,
   findall(M,move(State,M),Ms),
   D1 is D - 1,
   MinMax is -MaxMin,
   evaluate_and_choose(Ms,State,D1,MinMax,(nil,min),(Move,Value)).
```

Gry dwuosobowe (6)

Strategia cięć alfa-beta [2]

```
evaluate_and_choose([M|Ms],State,D,A,B,Move,Best) :–
   move(M,State,State1),
   alpha_beta(D,State1,A,B,_,Value),
   Value1 is -Value,
   cutoff(M,Value1,D,A,B,Ms,State,Move,Best).
evaluate_and_choose([],_,_,A,_,Move,(Move,A)).

alpha_beta(0,State,_,_,_,Value) :–
   value(State,Value).
alpha_beta(D,State,A,B,Move,Value) :–
   findall(M,move(State,M),Ms),
   A1 is -B, B1 is -A, D1 is D - 1,
   evaluate_and_choose(Ms,State,D1,A1,B1,nil,(Move,Value)).

cutoff(M,Value,_,_,B,_,_,_,(M,Value)) :– Value >= B.
cutoff(M,Value,D,A,B,Ms,State,Move,Best) :–
   A < Value, Value < B,
   evaluate_and_choose(Ms,State,D,Value,B,M,Best).
cutoff(M,Value,D,A,B,Ms,State,Move,Best) :–
   Value =< A,
   evaluate_and_choose(Ms,State,D,A,B,Move,Best).
```

Gry dwuosobowe (7)

Nim - historia i zasady gry

- gra prawdopodobnie pochodzi z Chin, znana w Europie od XVI wieku, nazwę *nim* zaproponował w 1901 roku Charles L. Bouton - matematyk z Uniwersytetu Harwarda
- plansza, w stanie początkowym, zawiera bierki ustawione w rzędy – zarówno liczba rzędów jaki i liczba bierek w każdym rzędzie jest dowolna
- w każdym ruchu zawodnik usuwa dowolną (niezerową) liczbę bierek z jednego rzędu, w szczególnym przypadku cały rząd
- wygrywa gracz usuwający z planszy ostatnią bierkę

Nim - strategia specyficzna

- wszystkie stany „niekońcowe” można podzielić na 2 rozłączne klasy: stany bezpieczne (ang. *safe*) oraz stany niebezpieczne (ang. *unsafe*)
- dla każdego stanu niebezpiecznego istnieje przejście (ruch) do stanu zwycięskiego lub do stanu bezpiecznego
- w każdym stanie bezpiecznym, każdy ruch prowadzi do stanu niebezpiecznego
- strategia zwycięska istnieje dla gracza wykonującego ruch w stanie niebezpiecznym – polega ona każdorazowo na wykonaniu ruchu powodującego przejście do stanu zwycięskiego lub do stanu bezpiecznego

Gry dwuosobowe (8)

Nim - implementacja [2]

- w celu stwierdzenia, czy dany stan jest bezpieczny, należy przedstawić liczbę pionków w każdym rzędzie w systemie dwójkowym a następnie wyznaczyć sumę modulo 2 wszystkich tych liczb bez uwzględniania przeniesień (tzw. sumę nim); stan jest bezpieczny wtw gdy suma nim równa się 0
- w poniższym programie plansza jest reprezentowana jako lista, której każdy element to liczba pionków w kolejnym rzędzie; reprezentacją ruchu jest term (N, M) oznaczający zabranie M pionków z N-tego rzędu

```
play :–
        initialize(State,Player),
        display_state(State),
        play(State,Player).
play(State,Player) :–
        final_state(State,Player,Result), !,
        announce(Result).
play(State,Player) :–
        choose_move(State,Player,Move),
        display_move(Player,Move),
        move(Move,State,State1),
        display_state(State1),
        next_player(Player,Player1), !,
        play(State1,Player1).
choose_move(State,computer,Move) :–
        evaluate(State,Safety,Sum),
        decide_move(Safety,State,Sum,Move).
choose_move(State,human,Move) :– ask(State,Move).
```

Gry dwuosobowe (9)

Nim - implementacja [2] (cd.)

```
initialize([1,3,5,7],human).

display_state(State) :–
   write('Game state: '),
   write(State),
   writeln('.'), nl.

display_move(human,_) :– !.
display_move(computer,Move) :–
   write('My move is: '),
   write(Move),
   writeln('.').

final_state([],Player,Player).

announce(computer) :–
   writeln('You won! Congratulations!').
announce(human) :– writeln('I won...').

ask(State,Move) :–
   write('Your move > '),
   read(Move),
   legal(Move,State).

legal((K,N),State) :–
   nth1(K,State,M),
   N =< M.

evaluate(State,Safety,Sum) :–
   nim_sum(State,[],Sum),
   safety(Sum,Safety).
```

Gry dwuosobowe (10)

Nim - implementacja [2] (cd.)

```
safety(Sum,safe) :– zero(Sum), !.
safety(Sum,unsafe) :– not(zero(Sum)),!.

decide_move(safe,_,_,(1,1)).
decide_move(unsafe,State,Sum,Move) :–
   safe_move(State,Sum,Move).

move((K,M),[N|Ns],[N|Ns1]) :–
   K > 1,
   K1 is K–1,
   move((K1,M),Ns,Ns1).
move((1,N),[N|Ns],Ns).
move((1,M),[N|Ns],[N1|Ns]) :–
   N > M,
   N1 is N – M.

next_player(computer,human).
next_player(human,computer).

nim_sum([N|Ns],Bs,Sum) :–
   binary(N,Ds),
   nim_add(Ds,Bs,Bs1),
   nim_sum(Ns,Bs1,Sum).
nim_sum([],Sum,Sum).

nim_add(Bs,[],Bs).
nim_add([],Bs,Bs).
nim_add([B|Bs],[C|Cs],[D|Ds]) :–
   D is (B+C) mod 2,
   nim_add(Bs,Cs,Ds).
```

Gry dwuosobowe (11)

Nim - implementacja [2] (cd.)

```
binary(1,[1]).
binary(N,[D|Ds]) :–
   N > 1,
   D is N mod 2,
   N1 is N // 2,
   binary(N1,Ds).

decimal(Ds,N) :– decimal(Ds,0,1,N).

decimal([],N,_,N).
decimal([D|Ds],A,T,N) :–
   A1 is A+D*T,
   T1 is T*2,
   decimal(Ds,A1,T1,N).

zero([]).
zero([0|Zs]) :– zero(Zs).

safe_move(Piles,NimSum,Move) :–
   safe_move(Piles,NimSum,1,Move).

safe_move([Pile|_],NimSum,K,(K,M)) :–
   binary(Pile,Bs),
   can_zero(Bs,NimSum,Ds,0),
   decimal(Ds,M).
safe_move([_|Piles],NimSum,K,Move) :–
   K1 is K+1,
   safe_move(Piles,NimSum,K1,Move).
```

Gry dwuosobowe (12)

Nim - implementacja [2] (cd.)

```
can_zero([],NimSum,[],0) :– zero(NimSum).
can_zero([_|Bs],[0|NimSum],[C|Ds],C) :–
  can_zero(Bs,NimSum,Ds,C).
can_zero([B|Bs],[1|NimSum],[D|Ds],C) :–
  D is 1–B*C,
  C1 is 1–B,
  can_zero(Bs,NimSum,Ds,C1).
```