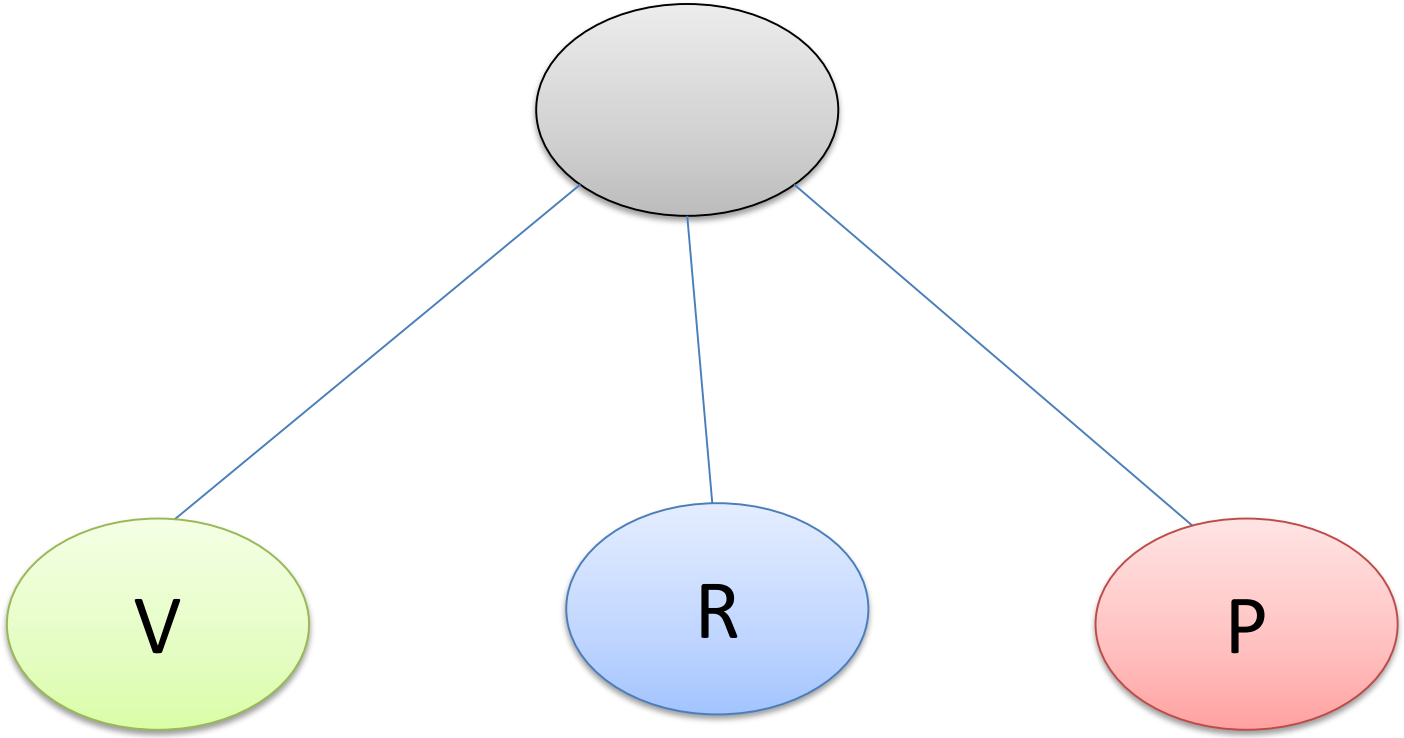
Optimalizácia výhernej stratégie na základe pravdepodobnosti ťahu protihráča

Pracovisko: ústav informatiky

Vedúci: Mgr. Alexander Szabari PhD

Autor: Slavomír Slovenkai

Vyriešená hra



V – výhra
R – remíza
P - prehra

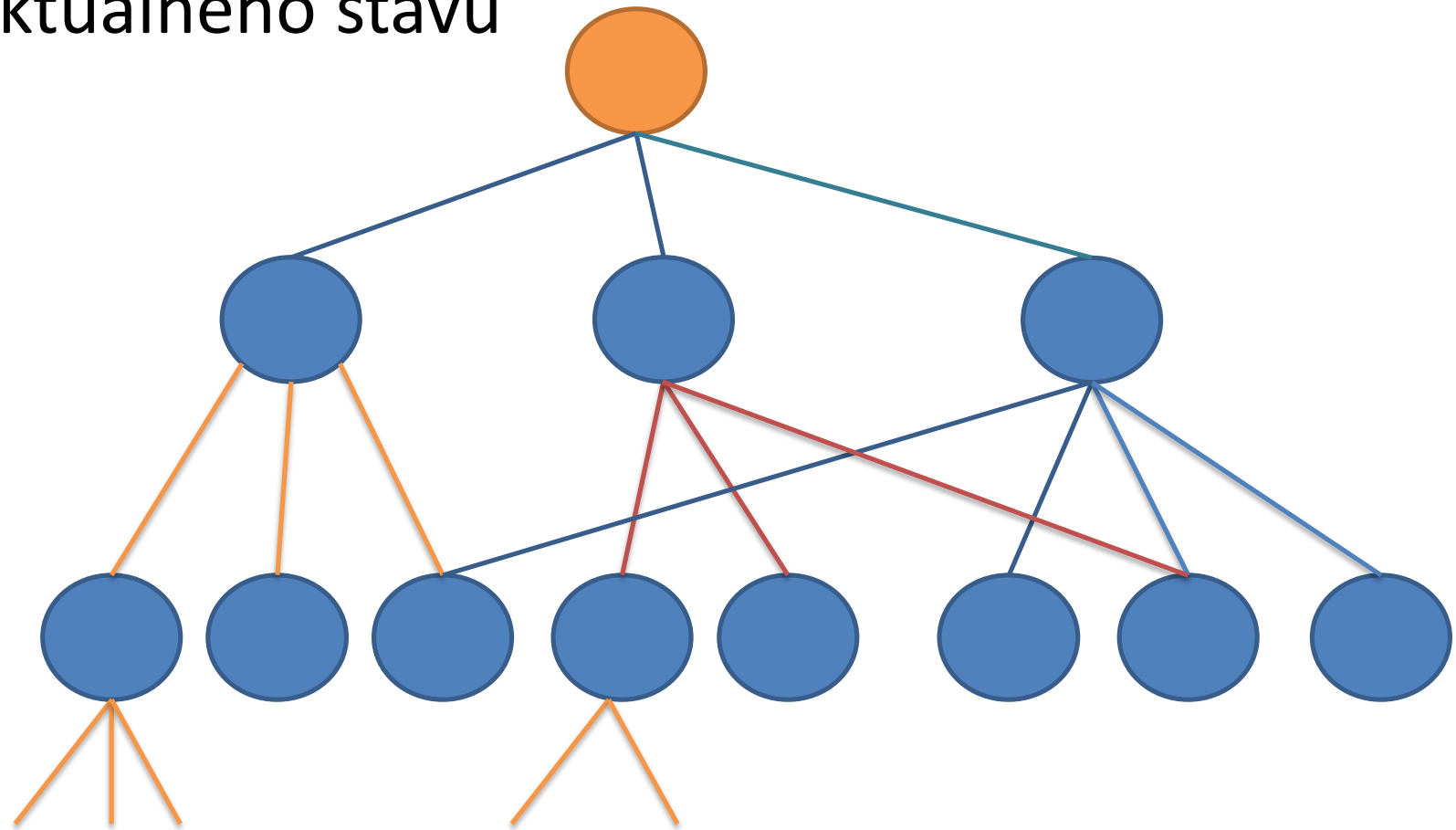
Hra dvoch hráčov

- hráč1: program (perfektný hráč)
- hráč2: človek

Chceme algoritmus, ktorý povie programu, aký ťah spraviť, aby vyhral na najmenej ťahov (alebo aspoň remizoval). V najhoršom prípade, aby prehral na čo najviac ťahov.

Strom hry

- Strom pozícií, do ktorých sa môže hra dostať z aktuálneho stavu



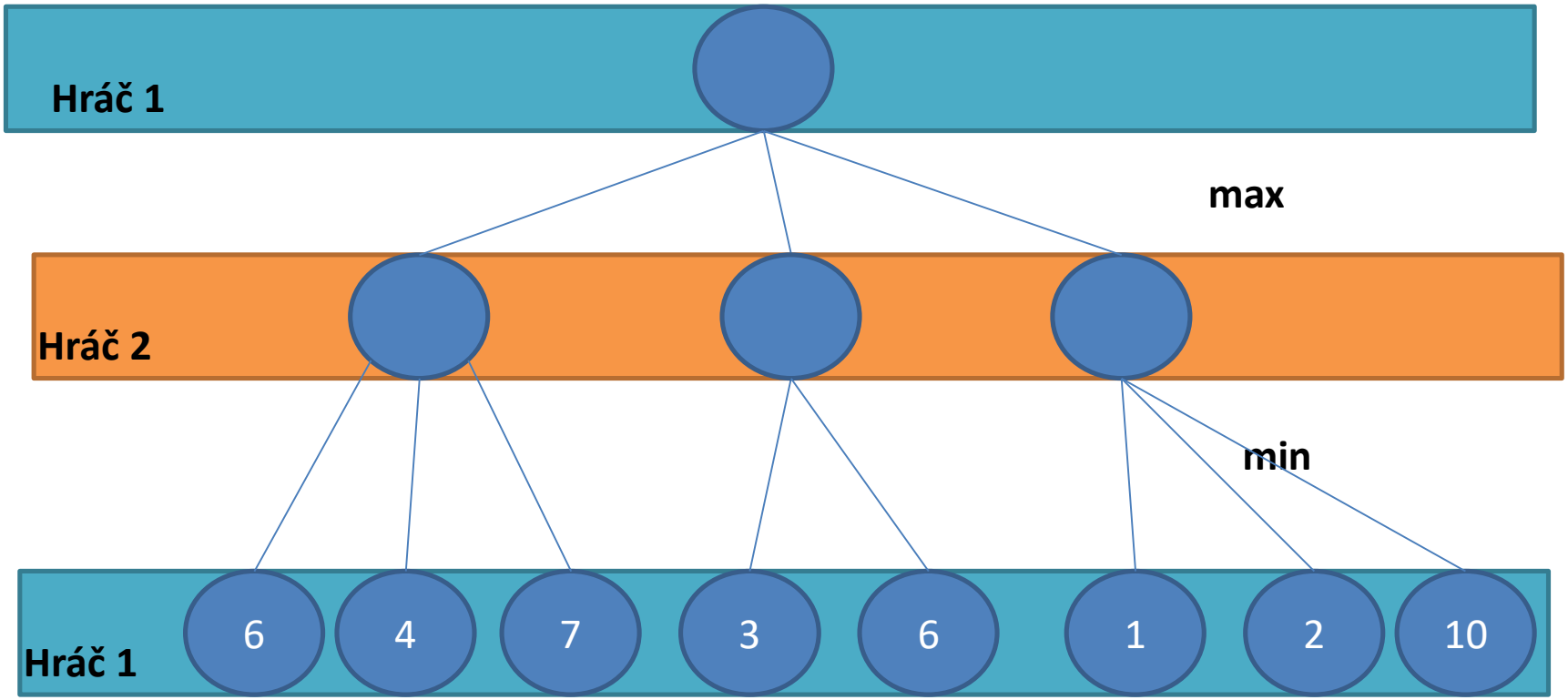
Alpha-Beta Search

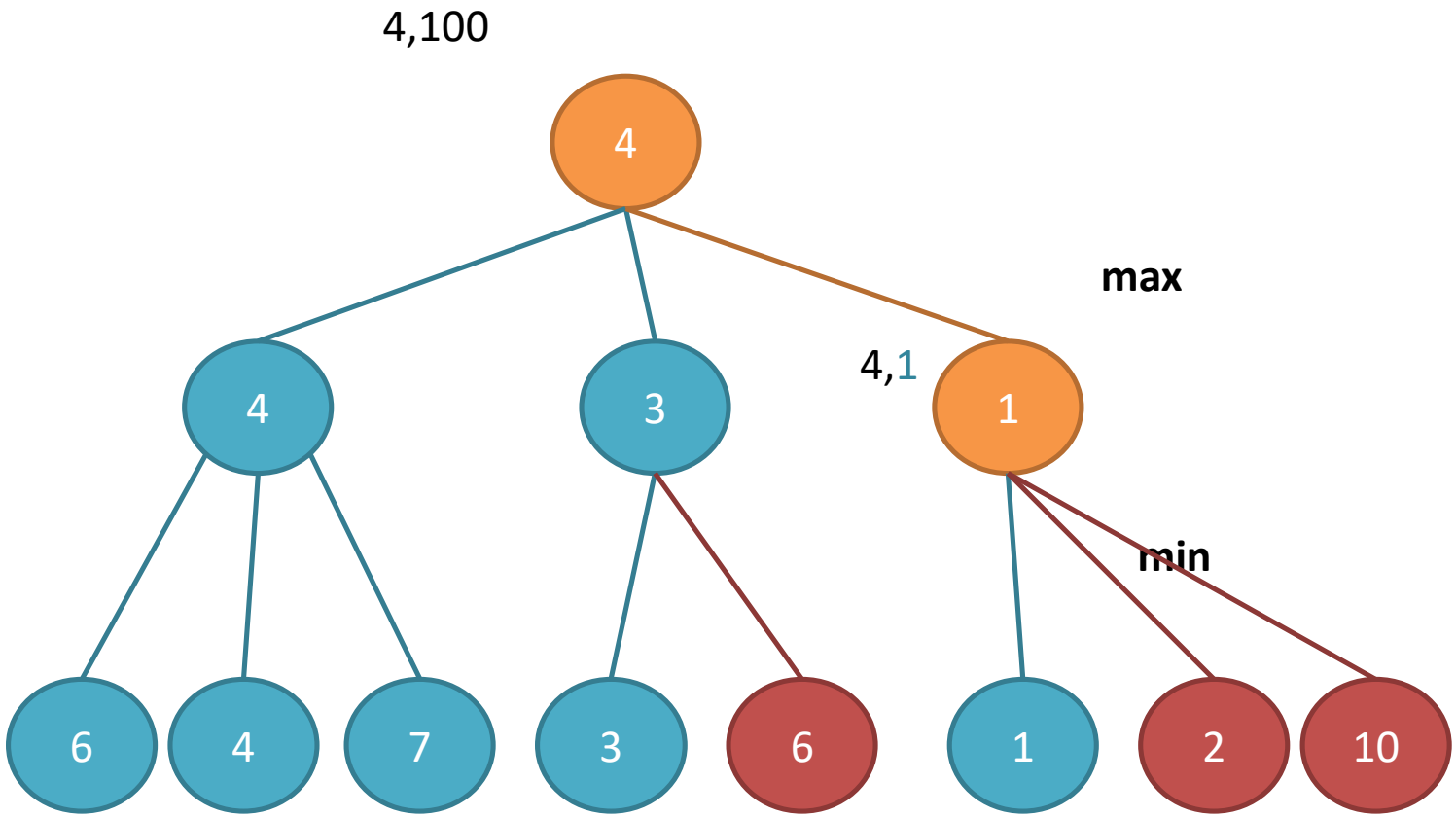
- Používaný pre hranie strategických hier medzi dvoma a viac hráčmi
- Úlohou je nájsť najlepší ťah v danej pozícii

- Predpoklad: existencia hodnotiacej funkcie
- Princípom je prechádzať strom hry a minimalizovať, maximalizovať možné straty

- Alpha-Beta search je vylepšený minmax algoritmus

- Min Max algoritmus prejde všetky možné ťahy, oboduje vzniknuté pozície a vyberie najvýhodnejší ťah (exponenciálny čas)
- Alpha-Beta: ak neviem dosiahnuť lepší výsledok, nehľadám v danej vetve


Hráč 1
max
Hráč 2
min
Hráč 1
6
4
7
3
6
1
2
10


4,100
4
max
4,1
4
3
1
min
6
4
7
3
6
1
2
10

Okrem hodnotiacej funkcie môžeme používať aj databázu stavov (alebo iba databázu)

Typ pohybu

- Min-max algoritmus

- Hodnotiaca funkcia brala do úvahy:
  - Uzavretie mlyna
  - Zabránenie mlynu
  - Uzavretie mlyna na ďalší ťah

Poloha figúrok

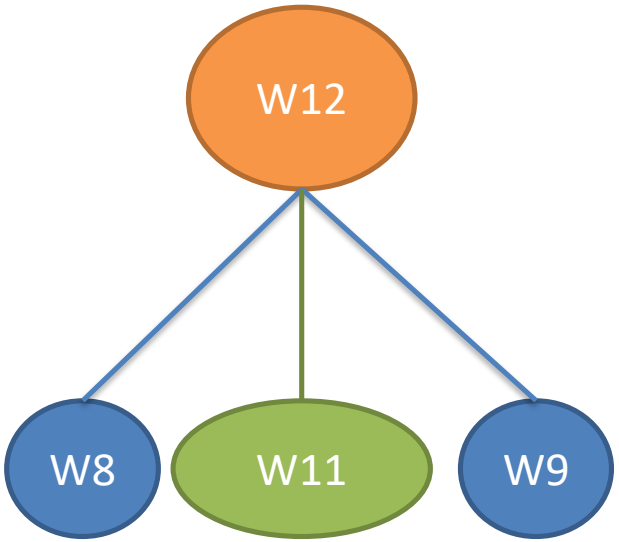
- Alpha-Beta algoritmus

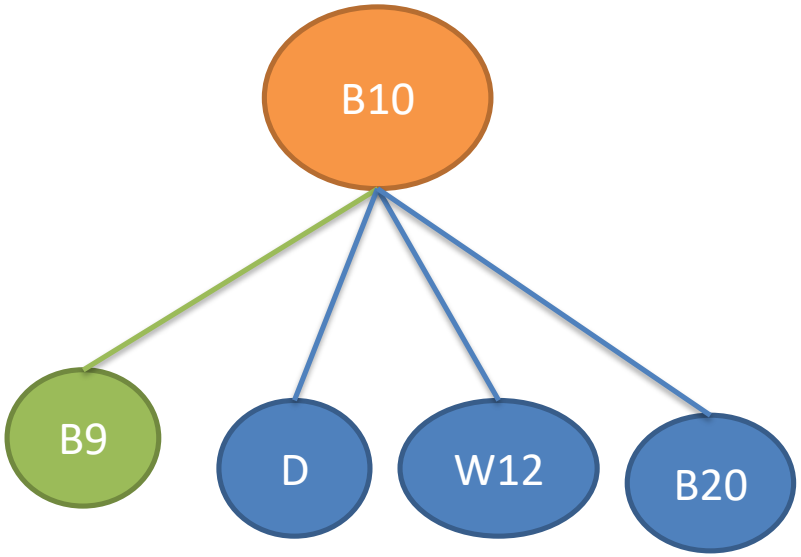
- Hodnotiaca funkcia berie do úvahy:
  - Počet figúrok hráčov
  - Koľko je voľných ťahov

Ultra-strong solution

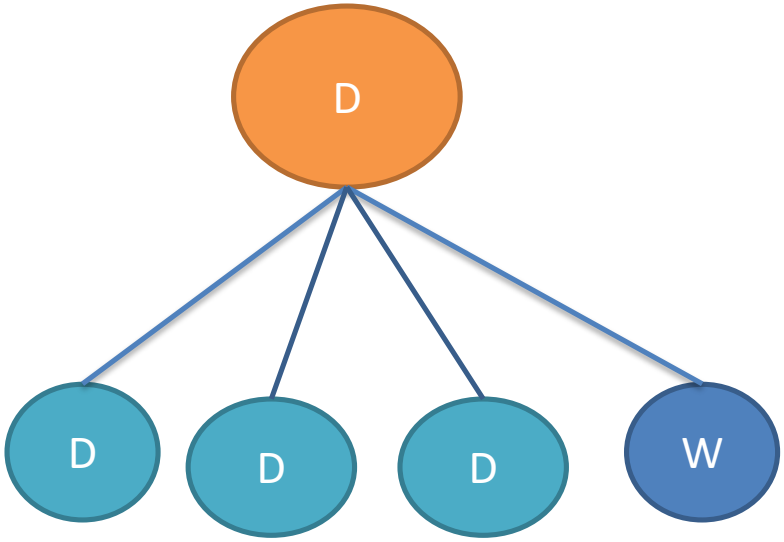
- Štandardná metóda
  - retrográdna analýza pre výpočet databázy z koncových stavov
  - Program používa túto databázu v každom ťahu
- Problém
  - Zacyklenie
- DTW (depth to win)
  - Rieši zacyklenie
  - Nezahŕňa remízu

Výber ťahu



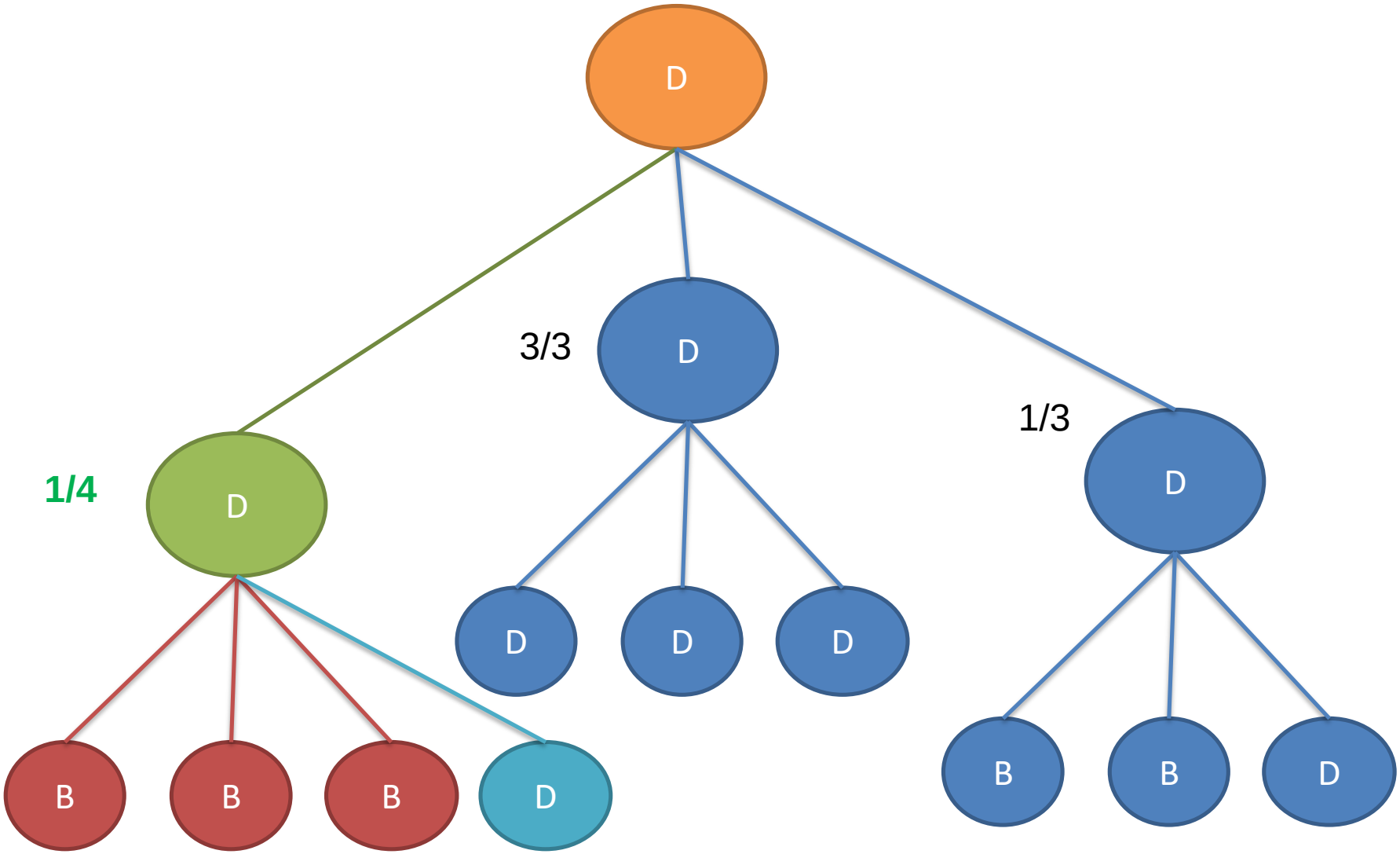


Výber ťahu



- **Vyber ťah, ktorý je pre protihráča najťažší**
  - pomer počtu prehier a remiz
  - distance dependent move selection
- **Klasifikácia remíz**

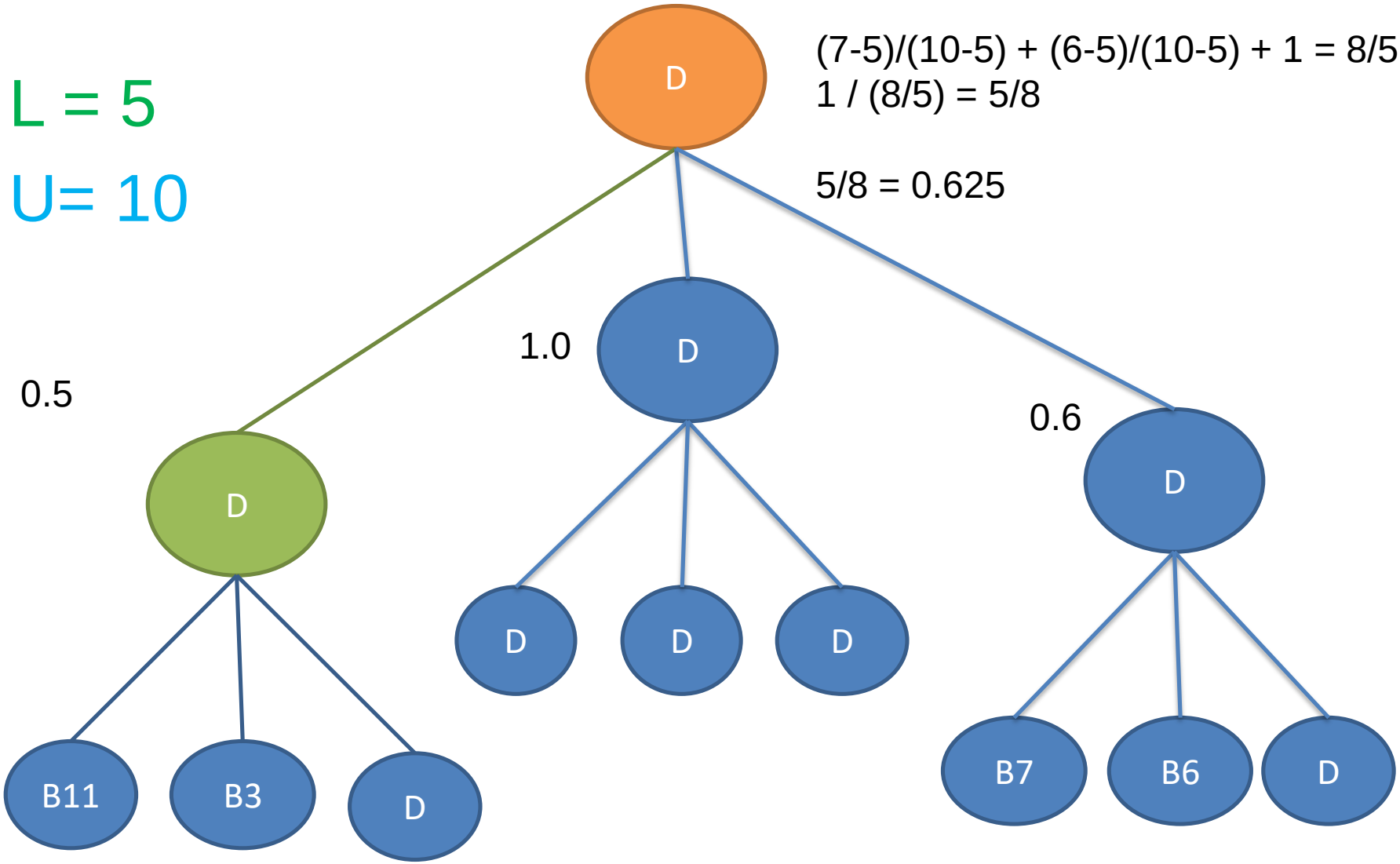
Počet prehier a remíz



Distance dependent move selection

L = 5

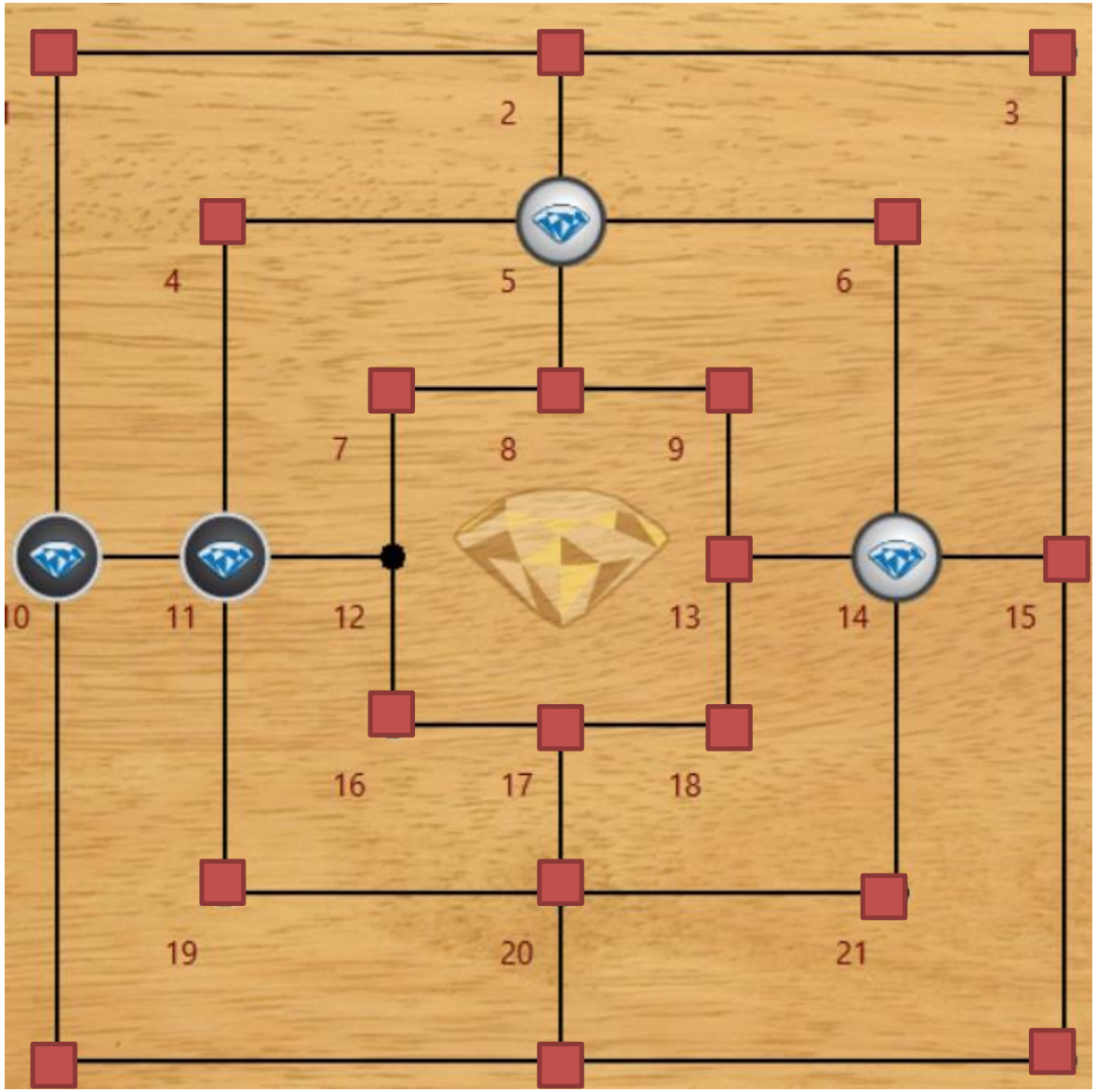
U= 10



Klasifikácia remíz

- Podpriestor: (f1, f2, p1, p2)
  - fi = počet figúrok na položenie pre hráča i
  - pi = počet položených figúrok pre hráča i
- Pracovná jednotka: s = (f1, f2, p1, p2) a -s= (f2, f1, p2, p1)
- Ohodnotenie podpriestoru:
- $(W_s + L_{-s} + D_s/2 + D_{-s}/2)/(T_s + T_{-s})$
- **Idea:** Ak nie je na výber výhra, vyber remízu s ktorou stráviš čo najviac ťahov v dobrom podpriestore.

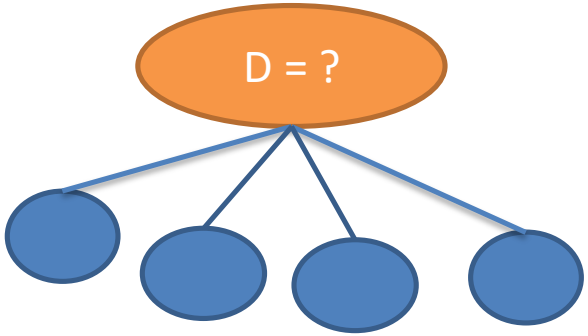
Na ťahu biely



- bola daná: databáza DTW
- C++
- počítač vs človek
- alpha-beta (+cache prehľadávania)
- Ohodnotenie listov:
  - f1: #W/#T
  - f2: ohodnotenie pohybu (voľné miesta, mlyn, blokovanie)
  - f3: ohodnotenie na základe # ťahov do konca

Budujeme databázu f3

- po leveloch: 3x3, 3x4, ... , 9x9, 8x9 (u),... ,1x1 (u)
- generujeme pozície pre XxY
- ohodnocujeme P: pozície
  - P == D
    - Nájdi možné ťahy a ohodnoť P



Veľmi drahý krok

    - Ulož ohodnotenie do MP

Pravdepodobnostná matica

- Jeden element DTW je 1 byte
- Jeden element PM sú 2 byty

- # element(DTW) = # element(PM)

Nájdi možné ťahy a ohodnoť P

- DTW hodnota P
- Možné ťahy M
- DTW pre m : M

Matica XY

Prob XY

Matica YX

Matica (Y-1)X

Prob YX

Prob (Y-1)X

Matica Y(X+1)

Matica (Y-1)(X+1)

Prob Y(X+1)

Prob (Y-1)(X+1)

ukladanie

vs

posúvanie

Vlákna (Worker – Consumer)

- Potrebovali sme upraviť kód aby bol thread safe
- Worker: generovanie pozícií
- Consumer: ohodnoť pozíciu

Budujeme databázu f3

- r: {3x3, 3x4, …, 9x9} U {8x9(u), … , 1x1(u)}
  - načítaj potrebné matice
  - generuj (r)
    - P: pozície -> vlákno
      - P == D -> výpočet -> updatePM
  - PM -> disk

Chceme prepočet s ohodnotenými remízami

Tento proces je potrebné spustiť viac krát

Prechod ukladanie posun

- počas hrania na základe ťahov
- # mlynov + # počet figúrok = # počet ťahov

- sú zmiešané s ostatnými pozíciami

Extra matice

vs

Ďalšia iterácia

Ďakujem za pozornosť