[Connect 4]

MinMax algoritam

|  |  |  |
| --- | --- | --- |
| Marija Magdalena Pamuković  Dora Parmać |  |  |

# O igrici

Connect4 je društvena igra za dva igrača u kojoj igrači prvo izabiru svoju boju, a zatim naizmjenično ubacuju žetone u mrežu 7x6 s ciljem da prvi spoje 4 žetona horizontalno, vertikalno ili dijagonalno.

Prvi put je prodana u veljači 1974. godine pod zaštitnim znakom *"Connect Four"*.



Connect4 spada u takozvane riješene (eng. solved) igre. Igrač koji prvi igra uvijek može pobijediti ako izabere prave poteze.

Jedna od mjera složenosti Connect4 igre je broj mogućih pozicija igre na ploči. Za klasični Connect4 sa 6 visokom i 7 širokom mrežom, postoje 4,531,985,219,092 pozicije za sve ploče naseljene s 0 do 42 komada žetona.

### Rješenje igre

Igru je prvi riješio James Dow Allen (1. listopada 1988.), a zatim nezavisno Victor Allis (16. listopada 1988.). Obojica opisuju dobitne strategije u svojoj analizi igre. Tada se *"brute force"* način nije smatrao izvedivim zbog računalne tehnologije dostupne u to vrijeme.

Rješenja *"brute force"* medodom počinju 1995. radom Johna Trompa koji sastavlja 8-slojnu bazu podataka. Algoritmi umjetne inteligencije koji dobro rješavaju ovu igru su *"minimax"* i *"negamax"* uz optimizacije koje uključuju *"AlphaBeta pruning"* i *"transposition tables*".

Riješeni Connect4 je igra u kojoj prvi igrač pobjeđuje. Savršenom igrom prvi igrač sigurno pobjeđuje na 41. potezu ili prije njega ako mu je prvi potez u srednjem stupcu.

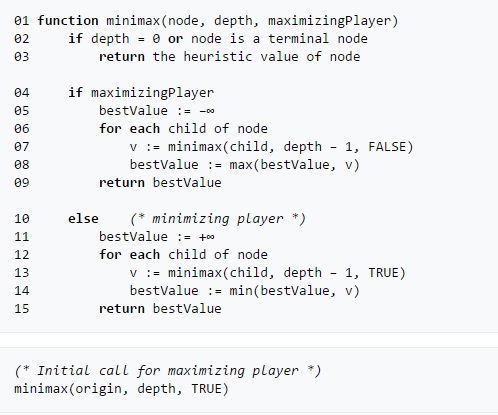
# MinMax algoritam

MinMax (minimax) je rekurzivni algoritam koji odabire idući potez u igrama s 2 ili više igrača kao što su križić-kružić, Connect4 i šah.

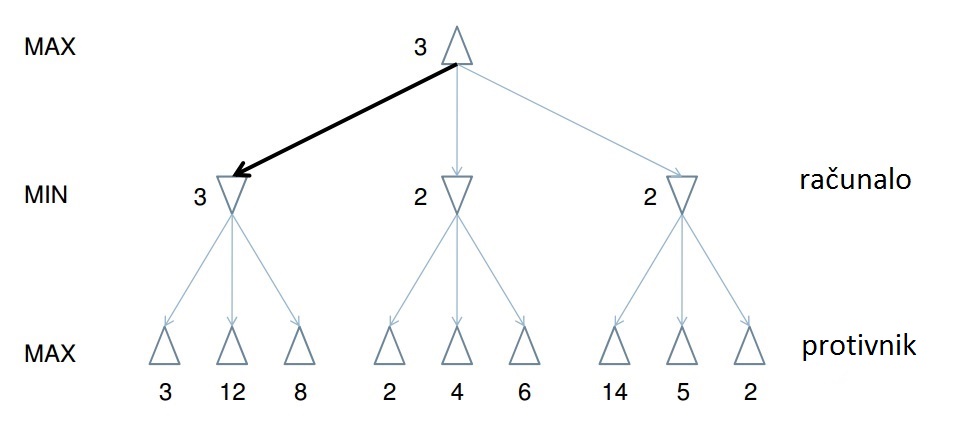
Sa svakim položajem tj. stanjem u igri povezujemo određenu vrijednost koja pokazuje koliko dobro bi bilo za igrača da odigra taj potez tj. korisnost poteza.

Tu vrijednost izračunavamo pomoću evaluacijske funkcije (funkcije korisnosti) koja daje vrijednost terminalnih čvorova. Nakon toga igrač odabire potez koji maksimizira minimalnu vrijednost pozicije koja proizlazi iz protivnikovih mogućih sljedećih poteza ( tj. minimalni potez u slučaju da igra protivnik, a maksimalni potez u slučaju da igra računalo).

### Pseudokod



### Izgled stabla



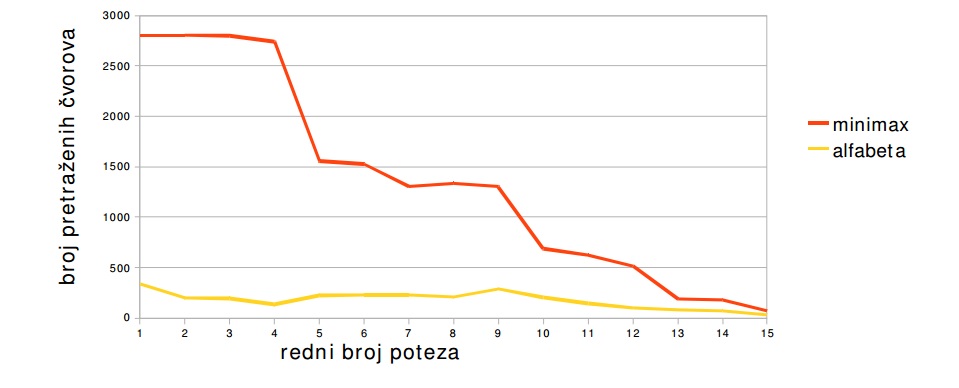
Pretpostavimo da je riječ o igri s 2 igrača te da je evaluacijska funkcija već izračunala vrijednosti na listovima sa slike. Ako je red da igra protivnik MinMax će uzeti najmanju vrijednost, a ako je red na računalo uzet će najveću.

### Ocjena algoritma

MinMax algoritam računa minmax izbor za trenutno stanje. Koristi jednostavni rekurzivni poziv za svu djecu trenutnog čvora. Rekurzija se spušta sve do terminalnih čvorova I onda se vraća gore računajući minmax vrijednosti svakog čvora do početnog. MinMax algoritam provodi potpuno pretraživanje stabla igre, i to pretraživanje u dubinu. Ako je maksimalna dubina stabla m, te u svakom trenutku postoji maksimalno b legalnih poteza, tada je vremenska složenost algoritma O(b^m). Prostorna složenost je O(b\*m) ako se svi čvorovi otvaraju odjednom, ili O(b) ako se otvaraju pojedinačno.

Problem sa MinMax algoritmom je očito njegova prevelika složenost. Tehnika kojom tu složenost možemo smanjiti naziva se AlfaBeta obrezivanje. AlfaBeta obrezivanje će uvijek donijeti istu odluku kao i sam MinMax algoritam, ali uz pretraživanje manjeg broja čvorova. Kako? Trik je u tome što, da bismo donijeli minmax odluku o nekom čvoru, nije potrebno otvarati sve njegove nasljednike.

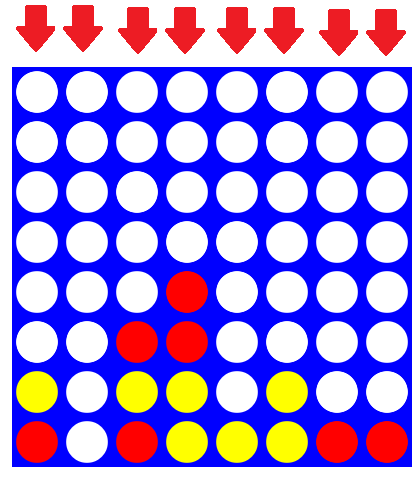
* POTPUNOST - ne
* OPTIMALNOST - ne
* VREMENSKA SLOŽENOST - O(b^m)
* PROSTORNA SLOŽENOST - O(b\*m)
* GLAVNI PROBLEM - vrijeme (pretražuje sve čvorove)



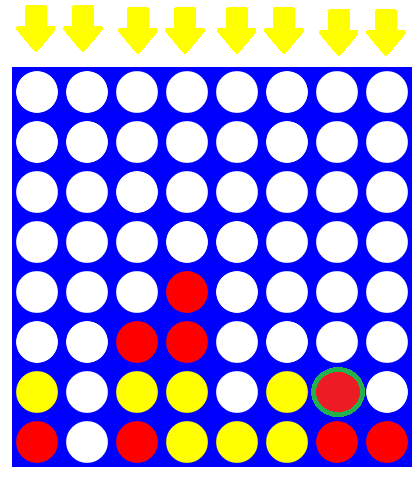
### Kako algoritam radi za Connect4 ?

Pogledajmo kako MinMax radi u igri Connect Four u kojoj računalo treba igrati protiv igrača, gdje je računalo crveni, a protivnik žuti.

* Svaki put kada je red da računalo igra, ono u obzir uzima sve moguće poteze koje može odigrati.



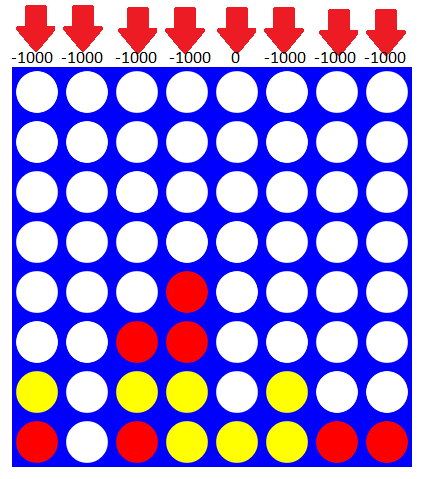
* Računalo se sada pretvara da je svaka od opcija zapravo odigrana i gleda gdje iduće protivnik može odigrati. Npr. kada pogledamo što bi se dogodilo da je računalo odigralo u predzadnji stupac, tada protivnik može odigrati slijedeće:



* Svaki od ovih žutih poteza se analizira na isti način, I gleda se gdje iduće računalo može odigrati. Iznos koliko puta će se ovaj postupak ponoviti naziva se dubina pretrage.

Sada vidimo da će stablo igre biti razgranato na 7 grana za svaki čvor. Dakle, nakon tri poteza postoji 7\*7\*7=343 mogućnosti, tj. 343 lista. Kada se program poziva prvi put, šalje mu se parametar dubine. Svaki put kada se novi sloj poteza razmatra, dubina se smanjuje za 1. Kada dođe do dubine 0, poziva se funkcija evaluacije.

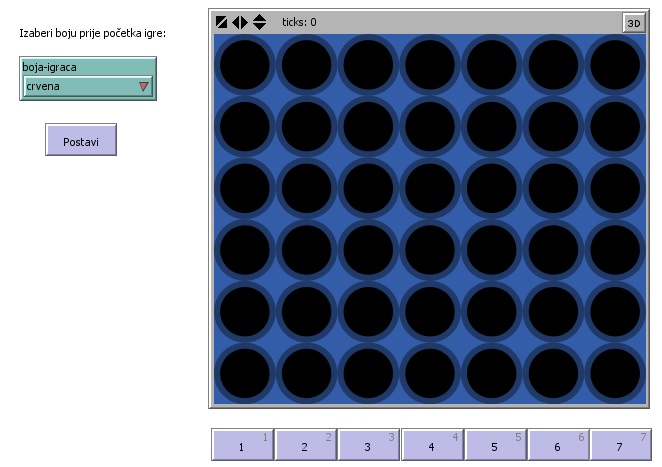
Pobjednički potezi za računalo se ocjenjuju s +∞, dok se pobjednički potezi za protivnika ocjenjuju s -∞. Svaki put kada se novih 7 poteza razmatra, ako igra računalo, uzima se potez s najvećom vrijednosti, a ako igra protivnik, uzima se potez s najmanjom vrijednosti. Nakon što je potez odabran, on se šalje natrag gore u stablo i koristi za procjenu poteza prije njega zajedno sa 6 drugih ocjena koje su također poslane natrag. Algoritam se zove MinMax jer je računalo u potrazi za smanjenem svojih maksimalnih gubitaka.

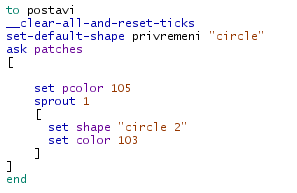
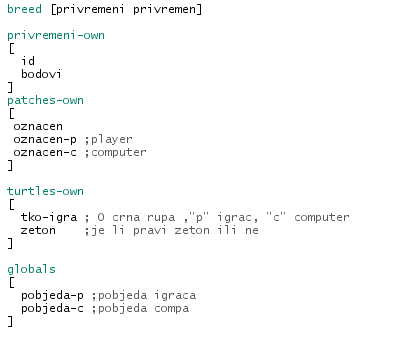


Na prethodnoj slici računalo testira svih 7 žutih poteza I zaključuje da je najbolji potez za žutog igrača 5. stupac jer taj rezultira pobjedom žutih i vrijednošću od –1000. Ta vrijednost se vraća natrag i rezultira stanjem na slici. Sada će računalo odigrati u 5. stupac jer će odabrati maksimalnu vrijednost tj. 0.

# Kako smo mi to implementirale?

### Postavljanje igre

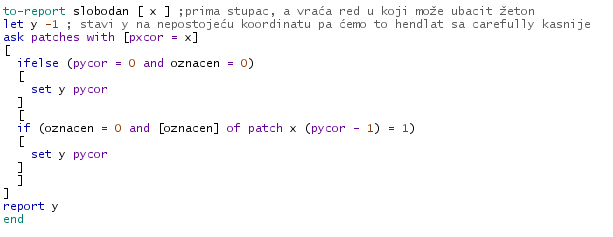




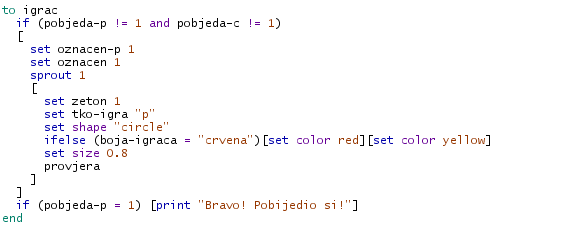
Vrsta privremeni predstavljat će imaginarne žetone, tj. poteze koji idući mogu nastupiti. Umjesto nove vrste zetoni, dodale smo kornjačama svojstvo zetoni koje će označavati radi li se o pravim žetonima ili nečem drugom.

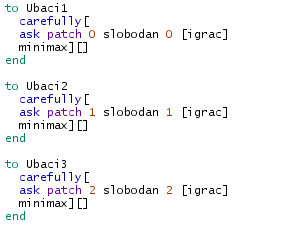
### Kako igrač igra?

Igrač ubacuje žetone klikom na dugmad ili na tipkovnici odabirom broja stupca u koji želi igrati. Kako žetoni ne smiju "letjeti", napravljena je metoda Slobodan koja prima koordinatu stupca, a vraća koordinatu reda u koji žeton smije upasti.



Metoda igrac pozivat će se u svakom od sedam dugmadi:



 i tako dalje za njih 7...

### Funkcija evaluacije

Privremeni žetoni imaju svojstvo bodovi u koje ćemo spremati vrijednosti evaluacijske funkcije. Naša metoda povjera služi kao evaluacijska funkcija i kao provjera je li netko pobjedio. Ta metoda zato mora biti dobro definirana za žetone (turtles) i za privremene žetone. Ako je riječ o pravom žetonu, onda će provjeriti pobjedu, a ako je riječ o privremenom, postavit će mu bodove. Sve kornjače imaju svojstvo tko-igra da znamo je li riječ o žetonu računala ("c") ili igrača ("p").

Ako je privremeni žeton računala zaključio da je pobijedio, postavit će svoje bodove na +10, a ako je privremeni žeton igrača zaključio da je pobijedio, postavit će svoje bodove na –10, inače bodovi će ostati 0.

Ako je pravi žeton računala zaključio da je pobijedio, postavit će globalnu varijablu pobjeda-c na 1, analogno ako je igrač pobijedio, postavit će pobjeda-p na 1.



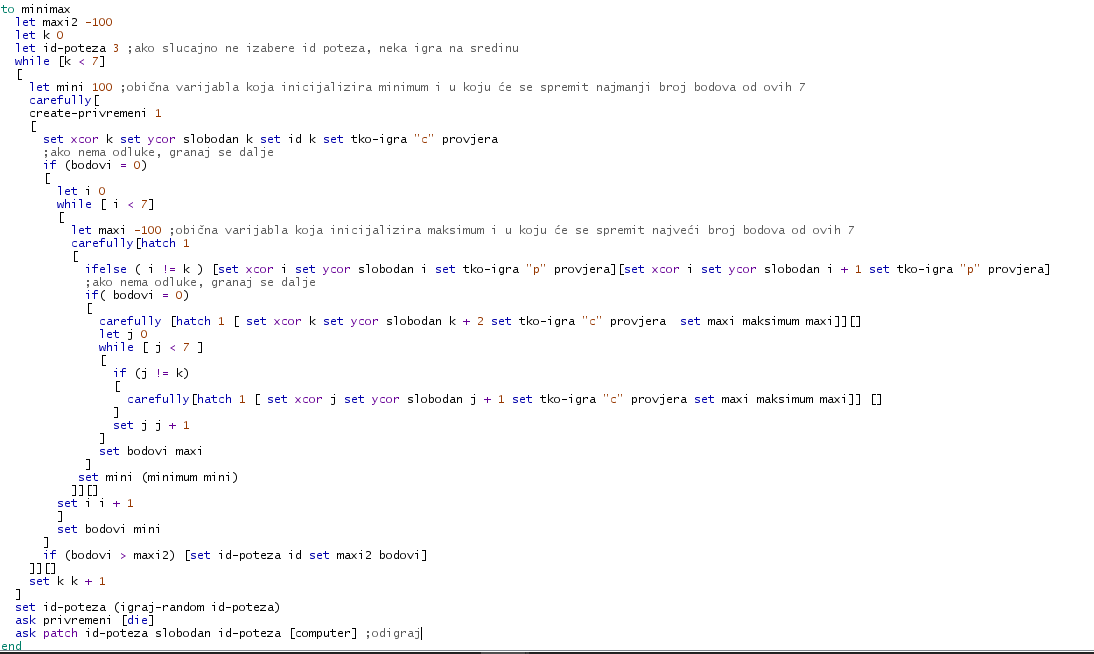
### MinMax

Naš minimax algoritam gleda 3 koraka unaprijed, tj. stablo igre ima 7\*7\*7 listova. Umjesto rekurzijom, cijeli algoritam je postignut pomoću while petlji. Svaki čvor stabla (privremeni) while petljom stvara novih 7 grana (privremenih).

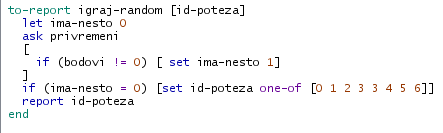
Umjesto do kraja grane, naš algoritam ide do prvog čvora (privremenog) koji ima bodove različite od nula, tako da nismo morale koristiti *"AlphaBeta pruning".*

Kada je red na računalo (tko-igra = "c"), računa se maksimalna vrijednost od danih 7 čvorova. Kada je red na igrača (tko-igra= "p"), računa se minimalna vrijednost od danih 7 čvorova. Izračunata maksimalna/minimalna vrijednost se zatim šalje roditelju i to se ponavlja u svakoj while petlji. Na kraju ostaje 7 vraćenih vrijednosti (bodova) pa kako je red da igra računalo, računamo maksimalnu vrijednost od tih 7 (maxi2).

Kako bi računalo znalo gdje će odigrati, prvih 7 privremenih koji pripadaju računalu pamte stupac u koji su ubačeni (id). Varijabla id-poteza će spremiti id od onoga koji ima maxi2 bodova.

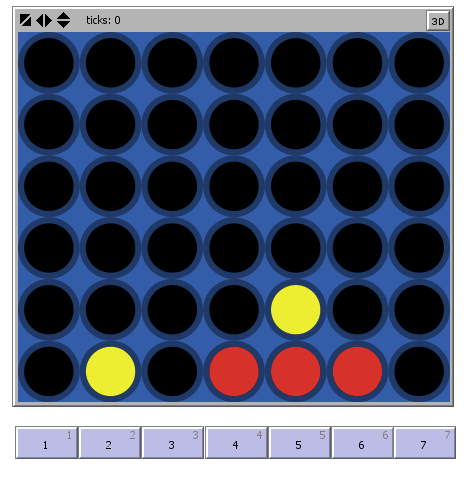


Na kraju, kako računalo ne bi igralo uvijek u prvi stupac kada mu svi privremeni imaju bodove 0, napravljena je metoda igraj-random u kojoj je malo veća mogućnost da igra na sredinu.



### Fatalni slučaj

Kada računalo vidi da će sigurno izgubiti, neće više pokušati spriječiti protivnika već će igrati random. Ovu "grešku" nismo mijenjale, jer računalo očito zna da će izgubiti pa se ponaša kao da ga više "nije briga".



U slučaju na slici, računalo (žuti) neće igrati ni u 3. ni 7. stupac već u 1. To je iz razloga što minimax algoritam sada vraća -10 za sve stupce, pa će kao maxi2 vrijednost ostati –10 iz prvog stupca. Zato je metoda igraj-random nadograđena da i u ovom slučaju igra random.