Drevesno preiskovanje Monte Carlo

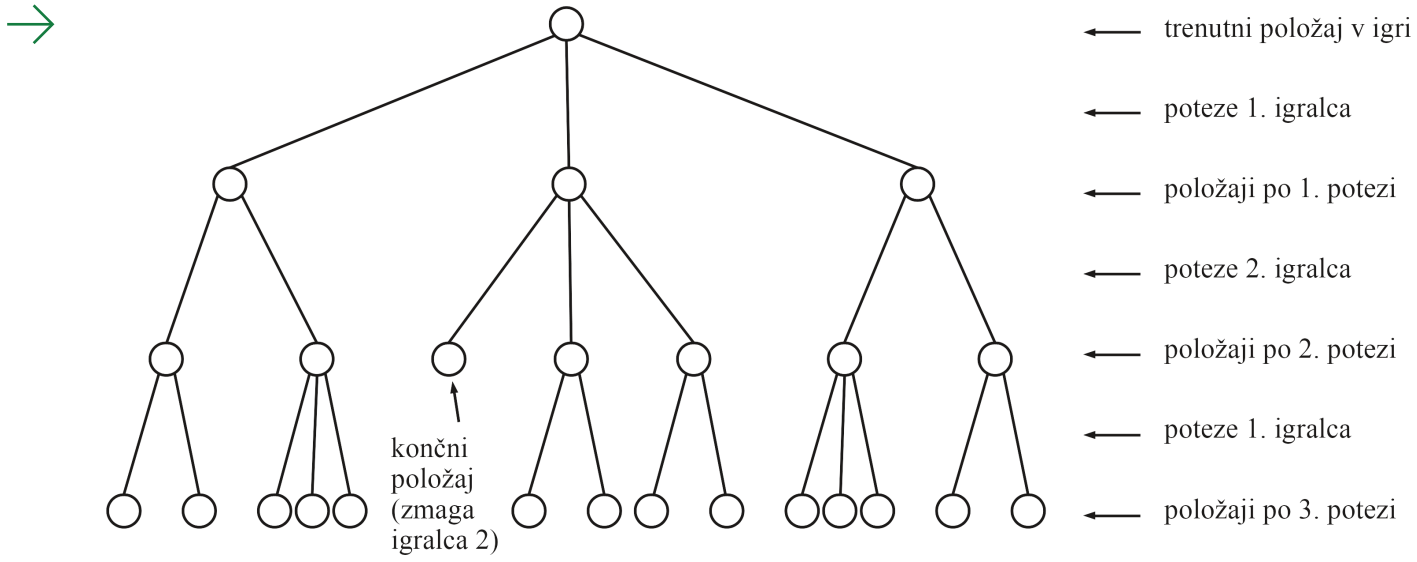
↓↓↓
Damjan Strnad

→ **Bralec tega prispevka se je najbrž že kdaj preizkusil proti računalniku v kateri od iger, kot so šah, dama ali vsaj križci-krožci. V omenjenih igrah je računalnik že zdavnaj presegel človeka, kar je delno zasluga napredka strojne opreme, predvsem pa algoritmov, ki v vsakem stanju igre precej zanesljivo določijo najboljšo naslednjo potezo.**

To naredijo z izgradnjo *drevesa igre* za določeno število potez v prihodnost. Vozlišča v drevesu igre so položaji v igri (npr. položaji na šahovnici), povezave med njimi pa poteze, ki jih izmenično vlečeta nasprotnika (glej sliko 1). Algoritem zgradi drevo igre iz trenutnega položaja, ki je v korenu drevesa, do določene globine (npr. pet mojih in pet nasprotnikovih potez). Položaji v listih drevesa igre so torej položaji, do katerih nas pripeljejo vse možne kombinacije naših in nasprotnikovih potez. Algoritem zatem z uporabo vgrajenega strokovnega znanja o igri oceni te položaje, od katerih so nekateri lahko tudi končni (npr. zmaga enega od igralcev). Ocena položaja pove, za katerega od igralcev je ta vsaj navidez bolj ugoden. Na podlagi ocen položajev se po posebnem postopku določi poteza, ki igralcu na potezi v korenu zagotavlja najboljše nadaljevanje ob najslabšem možnem scenariju (tj. optimalni igri nasprotnika). Osnovna metoda na tem področju je algoritem alfa-beta in njegove številne optimizacije, ki so ob vsesplošni paralelizaciji izvajanja na sodobnih elektronskih napravah in uporabi obsežnih zbirk specifičnega znanja o posamezni igri (t. i. hevristično znanje) prisilili človeka, da je že pred leti priznal premoč računalniku.

Obstajajo pa tudi igre, ki jim na sistematičnem pregledovanju drevesa igre temelječi algoritem alfa-beta in njegove izpeljanke niso kos. Za prvo takšno igro je veljala Arimaa, ki se igra s posebnimi figurami na standardni šahovnici, a je pred dvema letoma najmočnejše človeške igralce, tesno pa vendar, z uporabo algoritma alfa-beta premagal program Sharp. Morda je k temu pripomoglo tudi dejstvo, da Arimaa nikoli ni dosegla pretirane priljubljenosti in je zato tudi število vrhunskih človeških igralcev omejeno.

Druga in precej bolj znana tovrstna igra je go, ki se igra s polaganjem kamenčkov dveh barv na ploščo velikosti 19×19 polj. V zadnjih letih je igra go postala že kar slavna zaradi svoje vloge zadnjega stebra obrambe človeške inteligence pred hladno igralno logiko računalnikov. Slednji so namreč pri igranju igre go dosegali nivo povprečnega amaterja. Vse to se je spremenilo lani, ko je računalniški program po imenu AlphaGo prepričljivo premagal človeškega svetovnega prvaka. Algoritem, ki ga uporablja AlphaGo, je vsekakor daleč prekompleksen, da bi ga v celoti opisali tukaj, zainteresirani bralec pa najde podroben opis v članku [1]. V tem prispevku bomo opisali algoritem za gradnjo drevesa igre, ki je eden od sestavnih elementov programa AlphaGo in se imenuje drevesno preiskovanje Monte Carlo (*Monte Carlo tree search*, MCTS). MCTS ne temelji na hevrističnem ocenjevanju nekončnih položajev v listih drevesa igre kot alfa-beta, temveč kakovost naslednje poteze ocenjuje z velikim številom simulacij naključnih iger računalnika samega s sabo (t. i. *self-play*). Ker pa bi izključno naključne simulacije v omejenem času za potezo težko zanesljivo določile najboljšo potezo, algoritem MCTS kombinira med naključnim vlečenjem potez in prednostno izbiro potez, ki so se v predhodnih simulacijah izkazale kot potencialno dobre v posameznih položajih igre. Temu mehanizmu, ki je →



SLIKA 1.
Drevo igre

prisoten v praktično vseh iskalnih algoritmih umetne inteligence, rečemo uravnotežanje med *raziskovanjem* (naključne poteze) in *izkoriščanjem* (znane dobre poteze).

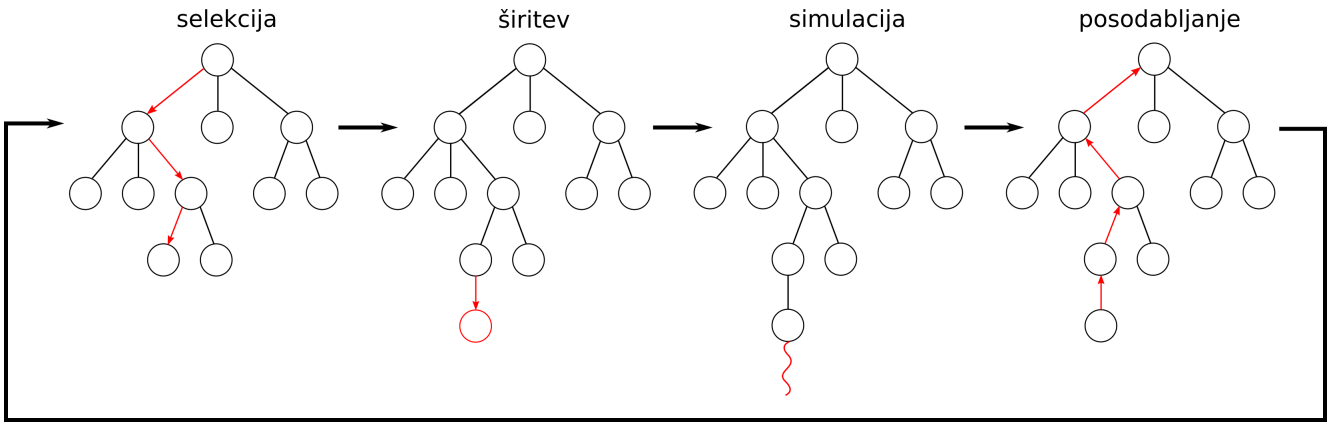
Osnovna značilnost iger, pri katerih se je MCTS uveljavil pred ostalimi algoritmi, je visok povprečni *vejitveni faktor*, tj. povprečno število možnih potez v poljubnem položaju igre. Pri igri Arimaa je ta ocenjeno na več kot 15000, pri igri go pa naj bi znašal skromnejših 250. Druga posebnost algoritmu alfa-beta neprijaznih iger je ta, da je v splošnem težko zanesljivo oceniti, kateri igralec in za koliko je v nekem položaju v prednosti. Z drugimi besedami, hevristično oceno je težko definirati samo na podlagi situacije na plošči.

V nadaljevanju prispevka bomo najprej opisali algoritem MCTS, zatem pa prikazali osnovni koncept delovanja na praktičnem primeru. Ironija praktičnega primera bo v tem, da bomo zanj uporabili igro križci-krožci, ki je morda najpreprostejša od vseh možnih iger in najlepši primer igre, pri kateri uporaba MCTS ni potrebna. Bo pa zato zgled dovolj preprost in bo jasno ponazoril princip delovanja algoritma, ki ga je mogoče prenesti neposredno na vse druge podobne in veliko bolj kompleksne igre.

Drevesno preiskovanje Monte Carlo

Naloga algoritma MCTS je poiskati najboljšo naslednjo potezo za igralca na potezi v trenutnem položaju pri igri. Podobno kot vsi sorodni algoritmi za igranje iger tudi MCTS trenutni položaj predstavi kot koren drevesa igre, drevo pa nato v več ciklih počasi dograjuje. Vsak cikel algoritma je sestavljen iz štirih korakov, ki si sledijo v naslednjem zaporedju (glej sliko 2):

- **Selekcija** – v tem koraku MCTS s pomočjo strategije selekcije izbira poteze, s katerimi se pomika od korena drevesa igre proti listom. Če v nekem vozlišču strategija izbere potezo v smeri naslednjega stanja, ki že ima pripadajoče vozlišče v drevesu igre, se premaknemo v to vozlišče in s selekcijo nadaljujemo. Če pa strategija izbere novo, še neuporabljeno potezo, se izvajanje algoritma prestavi v naslednji korak, tj. v širitev drevesa igre.
- **Širitev** – v tem koraku se ustvari oziroma *razvije* novo vozlišče za doslej neobstoječi položaj v drevesu igre, ki je bil izbran v koraku selekcije. Razvito vozlišče se doda v drevo igre kot nov list, tako da se drevo igre v vsakem ciklu algoritma MCTS poveča za natanko eno vozlišče. Sledi izvajanje naslednjega koraka, tj. simulacije.



SLIKA 2.
Koraki algoritma MCTS

- **Simulacija** – v tem koraku se izvede simulacija igre od trenutnega položaja do zaključka igre, pri čemer za izbiro potez uporabljamo t. i. simulacijsko strategijo. Ta je ponavadi zelo preprosta in izbira poteze za oba nasprotnika povsem naključno, lahko pa vključuje hevristiko in daje prednost potezam, ki običajno veljajo za dobre v določeni igri (npr. igranje v sredino ali vogal pri igri križci-krožci, če ta seveda še ni zaseden). Simulacija se zaključi, ko je dosežen kateri od končnih položajev igre (npr. trije križci ali krožci v vrsti).
- **Posodabljanje** – v fazi posodabljanja se najprej ovrednoti dosežen končni rezultat simulacije s stališča igralca na potezi. Pri igri križci-krožci bi bila možna vrednost igre npr. 1 za zmago, 0.5 za neodločen izid in 0 za poraz. Nato se pomikamo nazaj po verigi obiskanih vozlišč od nazadnje dodanega lista do korena in popravljamo nekatere pri vozliščih shranjene vrednosti. Te vrednosti uporablja strategija selekcije pri izbiri potez v naslednjih ciklih algoritma.

Izvajanje algoritma MCTS lahko omejimo z maksimalnim številom ciklov ali pa z omejitvijo časa za potezo. V slednjem primeru bo MCTS izvajal simulacije do izteka časa za potezo in nato na podlagi vrednosti vozlišč-naslednikov korena v drevesu igre predlagal najboljšo naslednjo potezo.

V zgornjem opisu algoritma manjka nekaj ključnih informacij, predvsem to, kako strategija selekcije izbira poteze pri pomikanju navzdol po obstoječem delu drevesa igre ter kako se povsem na koncu določi najboljša poteza. Da lahko pojasnimo oboje, moramo najprej vedeti, katere informacije se hranijo pri vsakem vozlišču drevesa igre. Pravzaprav jih ni tako veliko. Poleg seznama kazalcev na naslednike, od katerih so tisti na še nerazvite enostavno prazni (nil), in podatka o tem, kateri igralec je v vozlišču na potezi, sta za osnovni MCTS potrebna samo še dva števca. Prvi je števec obiskov vozlišča, ki pove, v koliko dosedanjih ciklih algoritma smo med fazo selekcije šli preko tega vozlišča. Drugi števec je vsota vrednosti iger, ki so šle preko tega vozlišča. Če smo torej neko vozlišče v dosedanjem poteku algoritma obiskali desetkrat in smo pri tem sedemkrat zmagali ter trikrat izgubili, bomo to na kratko zapisali z oznako 7/10 znotraj vozlišča. Slika 3 prikazuje primer dela drevesa igre (celotno drevo bi imelo 21 vozlišč) med igralcem s črnimi figurami, ki je na potezi, in njegovim nasprotnikom z belimi figurami. Algoritem MCTS je doslej izvedel 20 simulacij, od katerih se jih je 14 končalo z zmago za črnega, kar nam pove oznaka 14/20 v korenu drevesa (predpostavimo, da neodločen izid ni možen)[1]. Številke ob vozliščih predstavljajo enolične oznake vozlišč, da se bomo nanje lažje sklicevali. V nadaljevanju bomo za število obiskov vozlišča i uporabljali oznako n_i,

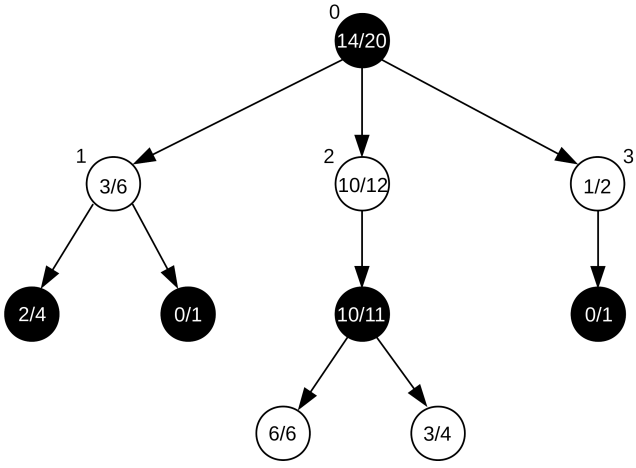
[1] Če je pozoren bralec opazil, da je število obiskov vozlišča enako vsoti obiskov v njegovih naslednikih samo v korenu, naj spomnimo, da vsako razvito vozlišče razen korena prvič obiščemo že ob njegovem razvoju.

za njegovo vrednost pa v_i.

Sedaj se lahko posvetimo problemu izbire potez med fazo selekcije algoritma MCTS. Daleč najbolj priljubljena strategija selekcije v obstoječih izvedbah osnovnega MCTS je strategija UCT (Upper Confidence bound for Trees). Naj bo i indeks trenutnega vozlišča, v katerem v koraku selekcije izbiramo naslednjo potezo za premik po drevesu igre navzdol, in naj bo j indeks kateregakoli od njegovih naslednikov, tudi tistih, ki jih še nismo razvili in torej še nimajo pripadajočega vozlišča v drevesu igre. Vrednost UCT naslednika j izračunamo po naslednji enačbi:

$$UCT_j = \frac{v_j}{n_j} + c \cdot \sqrt{\frac{\ln n_i}{n_j}}. \quad (1)$$

Ob enačbi (1) se velja nekoliko pomuditi. Prvi člen enačbe je razmerje med doseženo vrednostjo naslednika in številom iger, ki so potekale preko njega. Ta bo višja pri tistih vozliščih, ki so sodelovala v večjem številu zmagovalnih iger. Na primeru s slike 3 je, recimo, vrednost prvega člena enačbe (1) za najbolj levega naslednika korena enaka $\frac{3}{6} = 0.5$. Med dvema naslednikoma z istim številom doseženih točk bo višje ocenjen tisti, pri katerem nam je to uspelo v manjšem številu poskusov. In podobno, med dvema vozliščema naslednikoma z istim številom obiskov bo višje ocenjeno tisto, pri katerem smo ob tem dosegli višji izkupiček. Namen prvega člena enačbe (1) je torej favorizirati naslednike, ki so se v dosedanjih simulacijah izkazali za boljše. To je t.i. izkoriščevalski del prej omenjenega mehanizma uravnotežanja med raziskovanjem in izkoriščanjem.



SLIKA 3.
Primer dela drevesa igre za MCTS

Bralcu je najbrž jasno, da drugi člen enačbe (1) potemtakem predstavlja raziskovalni del mehanizma. To lahko potrdimo s tem, da razmislimo o vplivu ulomka pod korenom. V števcu ulomka nastopa število obiskov trenutnega vozlišča i, ki je enako za vse njegove naslednike. V imenovalcu nastopa število obiskov posameznega naslednika. Če je to višje, bo vrednost ulomka, s tem pa tudi celotnega drugega člena, nižja. Drugi člen bo torej favoriziral redkeje obiskane naslednike, kar ustreza raziskovalnemu elementu iskalnih algoritmov. Še nerazvitim naslednikom, pri katerih je n_j v obeh imenovalcih enak 0, priredimo vrednost $UCT = 1$. Eksperimentalno določena konstanta c je utež, s katero lahko poudarimo raziskovalno ali izkoriščevalno plat strategije UCT oziroma določamo ravnotežje med njima.

Strategija selekcije UCT v vsakem vozlišču izbere potezo v smeri naslednika, ki ima najvišjo ali najnižjo vrednost UCT. Če je takih naslednikov več (npr. vsi še nerazviti nasledniki), potem med njimi izbere naključno. V smeri najvišje ocenjenega naslednika igramo v tistih vozliščih, v katerih je na potezi igralec, ki je na potezi tudi v korenu (torej igralec, ki dejansko gradi drevo igre). V smeri najnižje ocenjenega naslednika igramo v vozliščih, kjer je na potezi nasprotnik. Bralec naj sam razmisli, zakaj je takšna strategija selekcije smiselna. Kot namig naj samo ponovimo, da vrednosti v v vozliščih odražajo višino izkupička s stališča igralca, ki je na potezi v korenu.

Oglejmo si, kako bi strategija selekcije izbirala poteze na primeru s slike 3 pri $c = 1$. V korenu izračunamo vrednosti UCT za naslednike in dobimo:

$$UCT_1 = \frac{3}{6} + \sqrt{\frac{\ln 20}{6}} = 1{,}207,$$

$$UCT_2 = \frac{10}{12} + \sqrt{\frac{\ln 20}{12}} = 1{,}333,$$

$$UCT_3 = \frac{1}{2} + \sqrt{\frac{\ln 20}{2}} = 1{,}724.$$

Strategija selekcije v korenu izbere potezo v smeri najvišje ocenjenega naslednika, kar je vozlišče 3. To stori tudi v primeru, če ima koren še kakšne nerazvite naslednike, katerih vrednost UCT je enaka 1. Ker vozlišče 3 ni končni položaj, v nadaljevanju stra-

tegija selekcije naključno izbere med njegovimi še nerazvitimi nasledniki, katerih vrednost UCT je enaka 1. Vrednost UCT že razvitega naslednika je namreč nižja, saj znaša $\sqrt{\ln 2} = 0,833$.

Preostane samo še vprašanje izbire končne poteze. Ko se čas za izbiro poteze izteče, vrnemo potezo v smeri največkrat obiskanega naslednika korena. Zakaj ne v smeri najbolje ocenjenega? Lahko tudi to, izkaže se namreč, da gre v večini primerov za eno in isto vozlišče, sicer pa se je izbira najpogosteje obiskanega naslednika korena izkustveno bolj uveljavila v praksi.

Zgled križci-krožci

Da teoretično razlago podkrepimo še s praktičnim zgledom, vzemimo primer dobro znane igre križci-krožci. Naj bo trenutni položaj v igri takšen, kot je prikazan v korenu drevesa na sliki 4, na potezi pa je križec. Pri izbiri naslednikov bomo upoštevali, da so preko horizontale, vertikale ali diagonale prezrcaljena stanja igre enakovredna. Med zrcalno enakovrednimi nasledniki bomo zato vedno upoštevali le enega, kar bo poenostavilo obravnavo in prikaz. Prav tako zaradi poenostavitve računanja uporabimo $c = 1$.

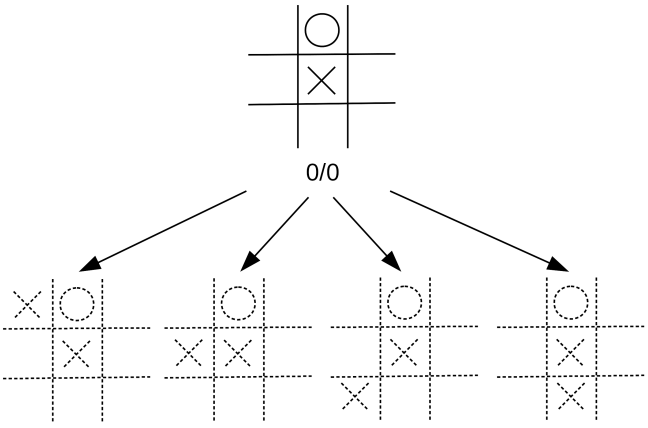
V prvem ciklu algoritma MCTS v korenu izbiramo med štirimi nerazvitimi nasledniki (izrisanimi črtkano na sliki 4) z oceno UCT enako 1. Med njimi zato izberemo naključno in denimo, da je izbrana poteza križca v sredino levega roba.

Drevo igre razširimo z novim vozliščem, iz katerega izvedemo naključno simulacijo igre. Predpostavimo, da se ta konča v položaju, kjer je zmagovalec križec (slika 5). Izid simulacije zato ovrednotimo z 1 in novorazvitemu listu ter korenu po poti nazaj posodobimo vrednosti n in v. V našem primeru to storimo tako, da oba števca povečamo za ena (slika 6).

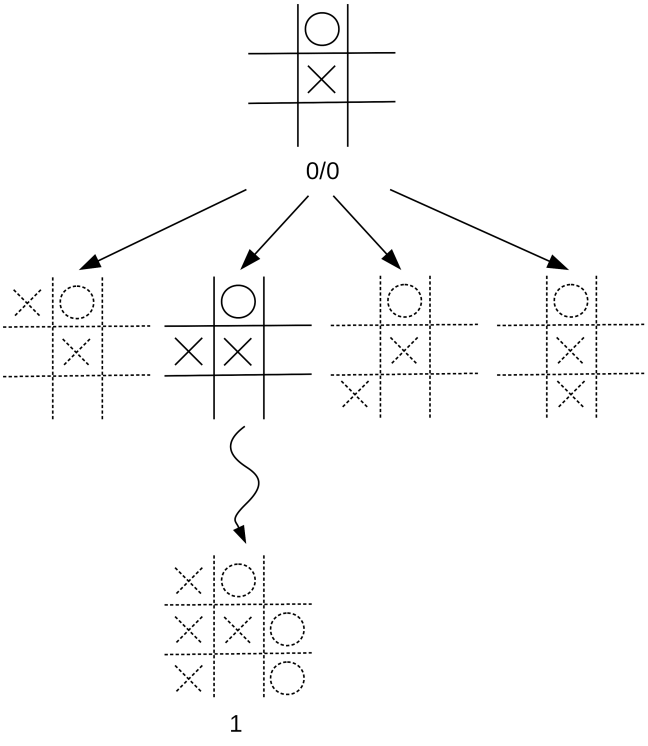
V drugem ciklu algoritma v korenu s strategijo UCT spet izbiramo med štirimi nasledniki. Trije nerazviti nasledniki b, d in e imajo še vedno vrednost UCT enako 1, prav tolikšno pa ima že razviti naslednik c, saj velja:

- $UCT_c = \frac{1}{1} + \sqrt{\frac{\ln 1}{1}} = 1.$

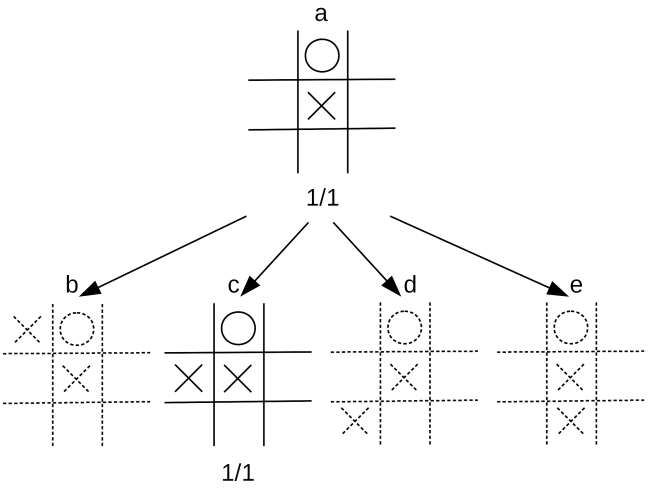
Če tokrat naključno izberemo naslednika e in od tam



SLIKA 4.
V korenu strategija selekcije naključno izbere med štirimi nerazvitimi nasledniki.

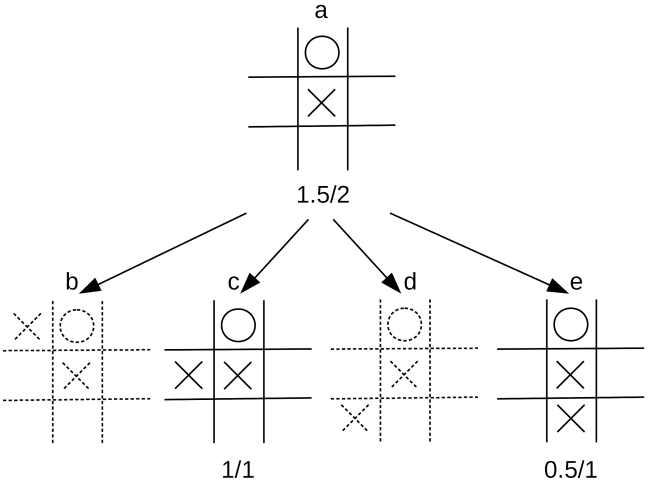


SLIKA 5.
Naključna simulacija iz novega lista se konča v položaju, kjer je zmagovalec igralec na potezi, tj. križec.

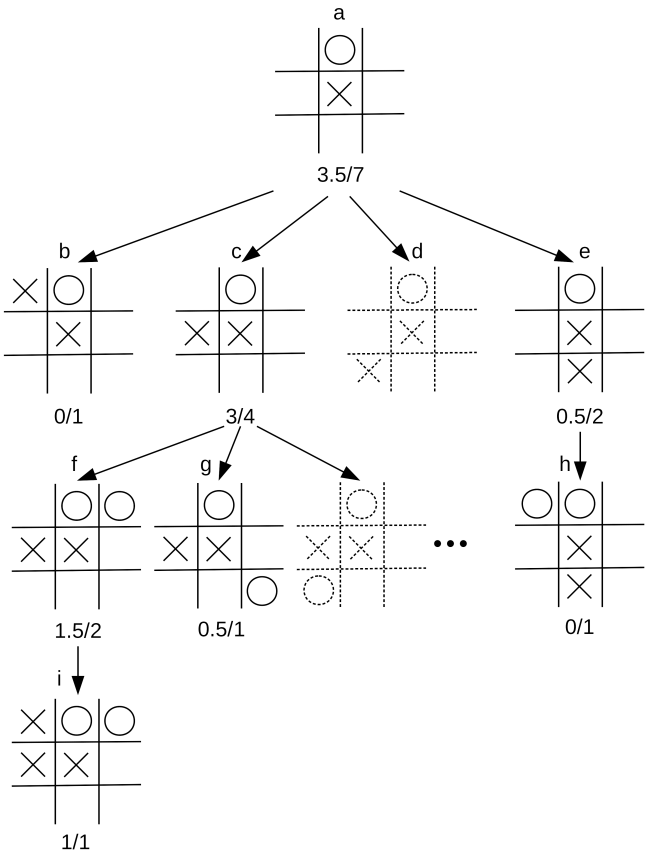


SLIKA 6.
V fazi osvežitve novemu listu in korenu povečamo oba števca.

naprej s simulacijo dosežemo neodločen končni izid, po posodobitvi vrednosti vozlišča e in korena dobimo situacijo s slike 7.



SLIKA 7.
Stanje drevesa igre po dveh ciklih algoritma MCTS



SLIKA 8.
Stanje drevesa igre po sedmih ciklih algoritma MCTS

V tretjem ciklu bomo v korenu izbirali med neraziskanima naslednikoma b in d z vrednostjo $UCT = 1$, naslednikom c z vrednostjo $UCT_c = 1 + \sqrt{\ln 2/1} = 1{,}833$ ter naslednikom e z vrednostjo $UCT_e = 0.5 + \sqrt{\ln 2/1} = 1{,}333$. Izbrano bo vozlišče c, v katerem se bomo potem odločali med šestimi možnimi nadaljevanji, od katerih še nobeno ni vključeno v drevo igre. Izbira novega lista na globini 2 drevesa igre bo torej spet naključna.

Da se primer ne zavleče preveč, skočimo na zadnji cikel algoritma – naj bo to cikel 8. Predpostavimo, da je drevo igre po sedmih izvedenih ciklih takšno, kot ga prikazuje slika 8. Vrednosti UCT vozlišč naslednikov korena so naslednje:

- $UCT_b = \frac{0}{1} + \sqrt{\frac{\ln 7}{1}} = 1,395,$

  $UCT_c = \frac{3}{4} + \sqrt{\frac{\ln 7}{4}} = 1,447,$

  $UCT_d = 1$

  $UCT_e = \frac{0.5}{2} + \sqrt{\frac{\ln 7}{2}} = 1,236.$

Izberemo torej potezo v smeri vozlišča c, katerega ocena je najvišja. V vozlišču c se odločamo med naslednikom f z oceno $UCT_f = 1{,}5/2 + \sqrt{\ln 4/2} = 1{,}583$, naslednikom g z oceno $UCT_g = 0{,}5/1 + \sqrt{\ln 4/1} = 1{,}677$ ter še nerazvitimi štirimi nasledniki z oceno $UCT = 1$. Ker je v vozlišču c na potezi krožec, strategija selekcije izbere potezo z najnižjo oceno UCT, torej naključno izbere enega od še nerazvitih naslednikov. Ne glede na izid simulacije bo po osmih ciklih najpogosteje izbran naslednik korena še vedno vozlišče c. Če bi se torej v tem trenutku morali odločiti za potezo, bi bila to poteza v sredino levega roba.

Literatura

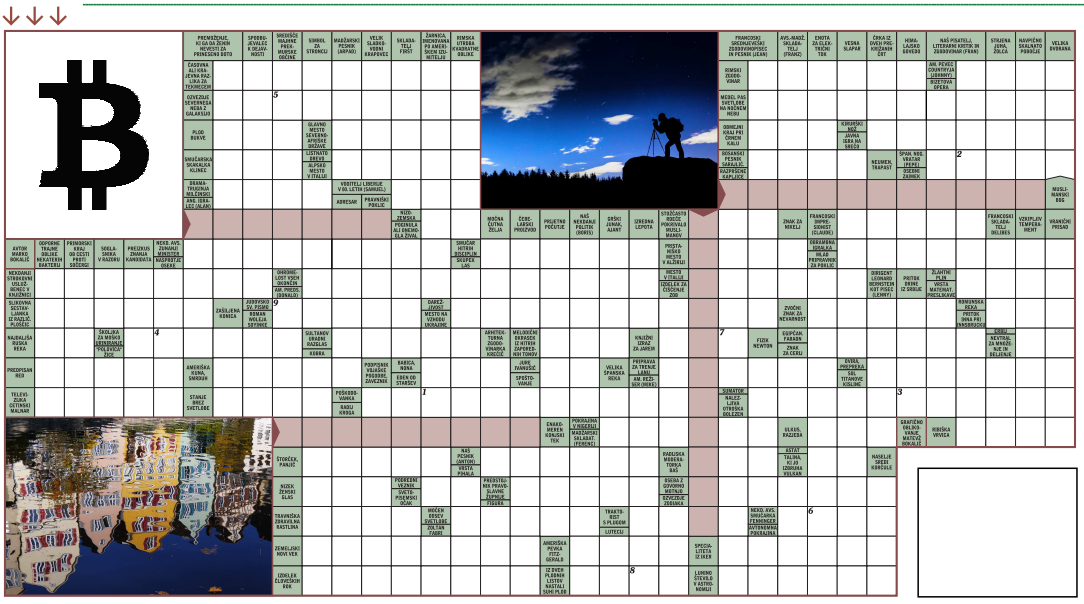
[1] D. E. Knuth in R. W. Moore, *An analysis of alpha-beta pruning*, Artificial Intelligence, 6(4), str. 293-326, 1975.

[2] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis in S. Colton, *A survey of Monte Carlo tree search methods*, IEEE Transactions on Computational Intelligence and AI in games, 4(1), str. 1-43, 2012.

[3] D. J. Wu, *Designing a winning Arimaa program*, ICGA Journal, 38(1), str. 19-41, 2015.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot in S. Dieleman, *Mastering the game of Go with deep neural networks and tree search*, Nature, 529(7587), str. 484-489, 2016.

×××



REŠITEV NAGRADNE KRIŽANKE

PRESEK 44/5

→ Pravilna rešitev nagradne križanke iz pete številke Preseka je **Nevidnost**. Izmed pravilnih rešitev so bili izžrebani Anže Mihelčič iz Kresnic, Vlasta Pospeh Fischer iz Celja in Jože Pavlišič iz Črnomlja, ki so razpisane nagrade prejeli po pošti.

×××