Veštačka inteligencija i igre

Andrej Ivašković

Matematička gimnazija, Nedelja informatike

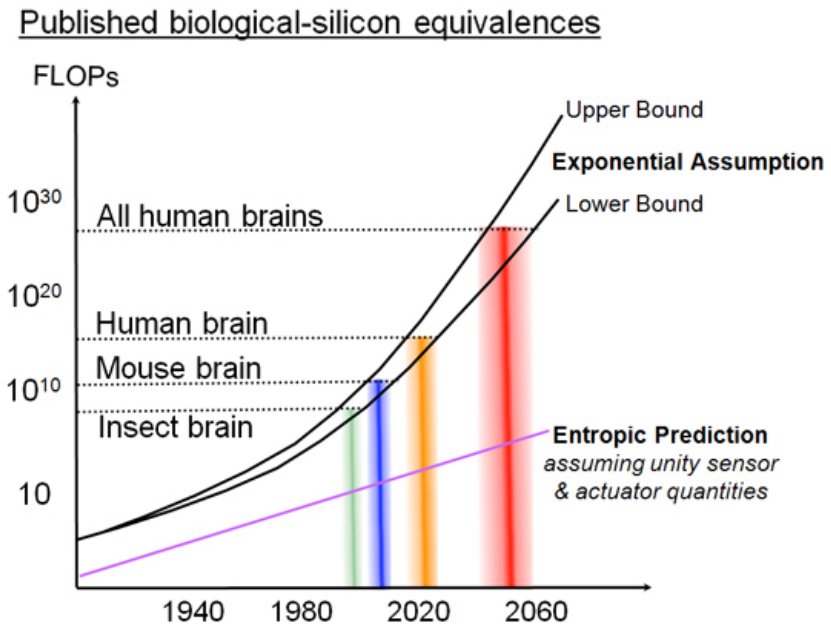
30. mart 2015.

O čemu ćemo pričati ovde

- Polovina prijavljenih je navelo ovo predavanje kao najzanimljivije!
- Kako uopšte definisati naš zadatak?
- Ono što računari rade uspešno često prestajemo da nazivamo veštačkom inteligencijom.
- Neke oblasti:
  - mašinsko učenje (i gomila primena i tehnika!);
  - obrada prirodnih jezika;
  - automatsko dokazivanje teorema.

Filozofska i etička pitanja

- Da li je veštačka inteligencija nemoguća?
- Da li bismo smeli da se bavimo daljim istraživanjima u ovoj oblasti?
- Krajem 2014. oglasili se Stiven Hoking, Elon Mask, Bil Gejts...



Šta je bilo rečeno?

O čemu NEĆEMO pričati

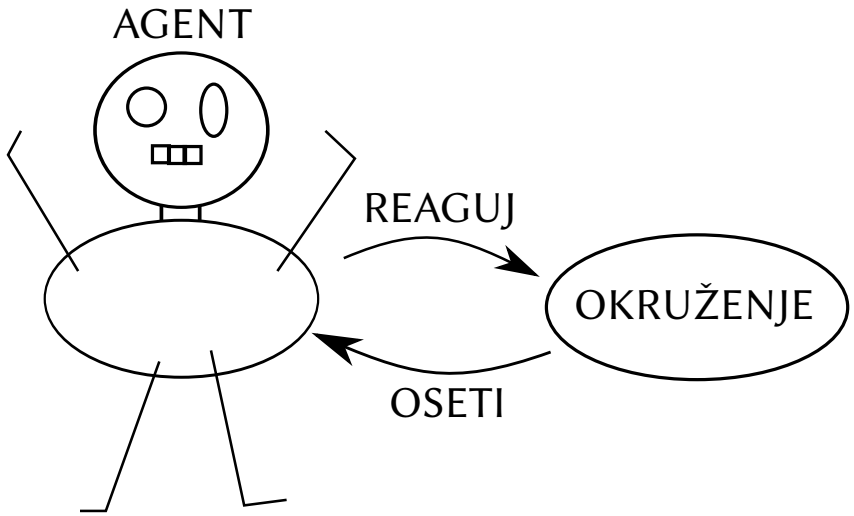
- Kako napraviti balansirano ponašanje za "veštačku inteligenciju"u računarskih igrama.
- Tjuringov test.
- Algoritmi i heuristike mašinskog učenja:
  - analiza klastera;
  - genetički algoritmi;
  - neuralne mreže.
- Hrpa matematika koja je inače neophodna za veštačku inteligenciju.
- Moji planovi za osvajanje sveta pomoću zlih robota.

Naš cilj danas

- Definisati **agente**.
- Pokazati neke standardne ideje iz veštačke inteligencije na sledećim primerima igara:
  - naučiti mašinu da igra **iks-oks**;
  - naučiti mašinu da igra **šah**, korišćenje pretraga stabala i **minimax** algoritma;
  - igranje igre **go**.
- Kako napraviti algoritam koji agentu u nepoznatom svetu određuje optimalno ponašanje?

Agenti

- Agentom nazivamo **bilo kakav uređaj** koji je u mogućnosti da prima nadražaje iz okoline i reaguje na njih.



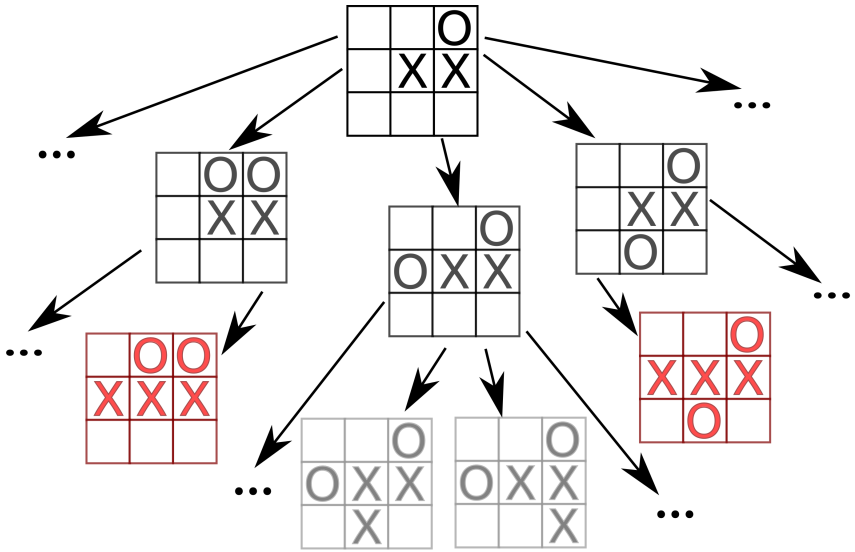
- Kako ocenjujemo agente?
- Kakvo sve može da bude okruženje?

Programiranje agenata

- Možemo li da održavamo tabelu koja će povezivati nizove draži sa reakcijama?
- Agent je **autonoman** ukoliko njegovo ponašanje na neki način zavisi od njegovih prethodno primljenih draži.
- Agent treba da održava:
  - opis trenutnog stanja okruženja;
  - informacije o tome kako se okruženje menja nezavisno od agenta;
  - informacije o tome kako agentove radnje menjaju okruženje.
- Agent bira reakcije u zavisnosti od **cilja** i **preferiranog puta**.
- Ravnoteža između primene naučenog i učenja.

Struktura igre **iks-oks**

- Svi ste (nadam se) upoznati sa tim kako ova igra funkcioniše.



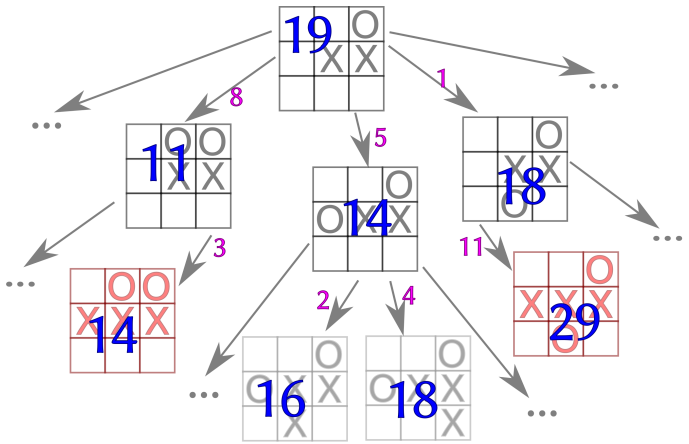
- Postoji mali konačan broj **stanja** igre koja možemo da predstavimo čvorovima acikličnog usmerenog grafa, a grane ovog grafa su potezi. Radi jednostavnosti ćemo koristiti stablo (do nekih stanja može da se dođe na više načina).
- Šta dalje radimo sa ovim stablom?

Određivanje pobedničke strategije

- U ovakvim igrama možemo da odredimo za svako stanje da li je **pobedničko**, **gubitničko** ili **nerešeno**, gde npr. stanje zovemo pobedničkim ukoliko pri optimalnoj igri oba igrača igrač koji je na potezu sigurno pobeđuje.
- Nakon izgradnje stabla, obilazimo stablo u *post-order* ("odozdo na gore"):
  - Za listove stabla znamo kojoj vrsti stanja odgovaraju.
  - Za ostale čvorove razmatramo slučajeve:
    1. Ukoliko postoji potez koji iz nekog stanja S dolazi do gubitničkog stanja L, tada je S pobedničko.
    2. Ukoliko svi potezi mogući u stanju S vode do pobedničkih stanja, S je gubitničko stanje.
    3. U suprotnom, stanje S je nerešeno.

Malo uopštenje...

- Ovaj princip možemo da koristimo u velikom broju igara sa malim brojem stanja (uključujući i one koje imaju samo jednog igrača npr. Rubikova kocka).
- Razmotrimo varijantu u kojoj svaki potez nosi poznati određeni broj poena, a cilj svakog igrača je da se ostvari što je moguće veća prednost u odnosu na drugog igrača.

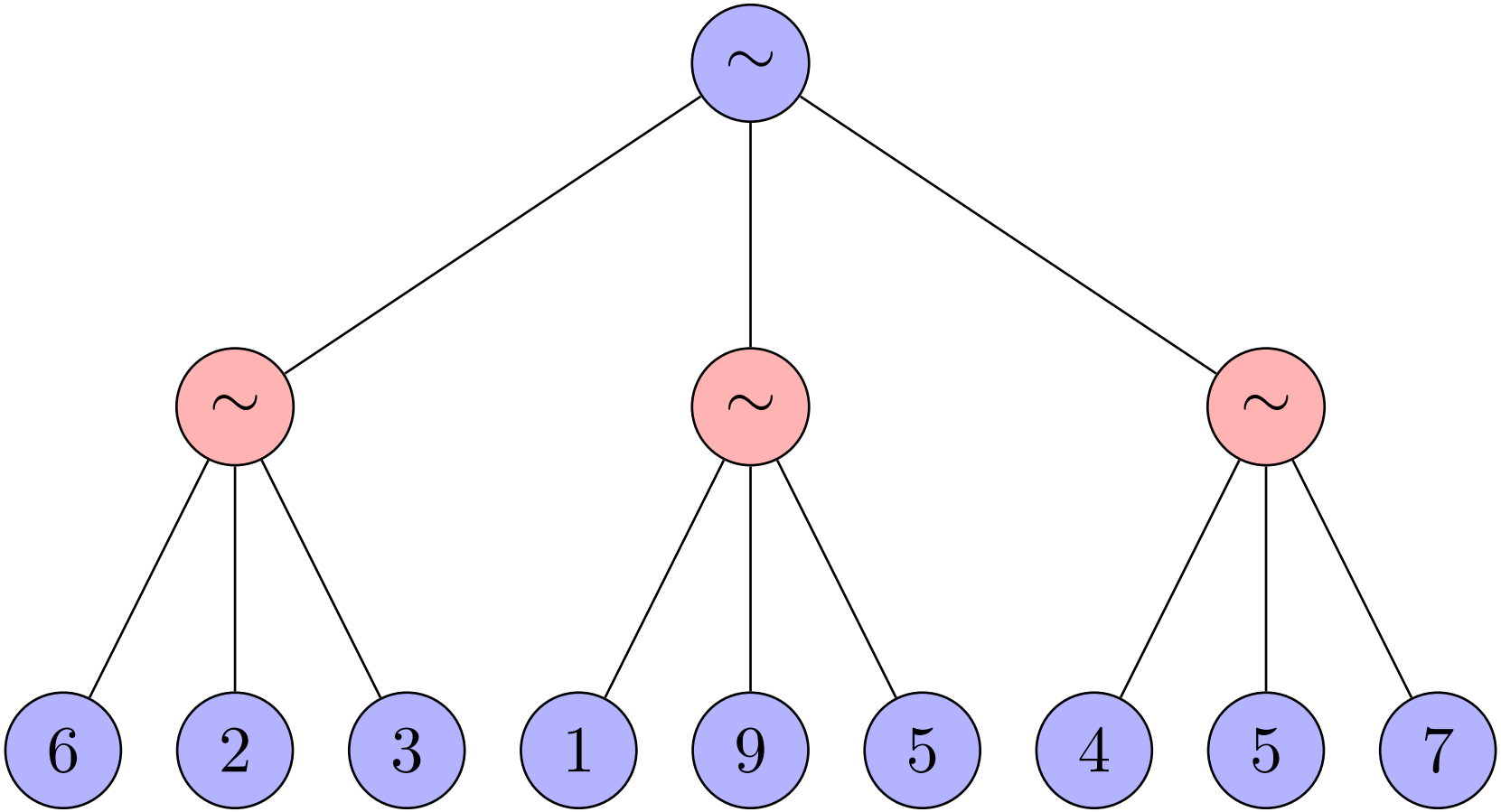


- Osnovna ideja algoritma ostaje ista, ali sada je "rekurentna veza" drugačija.

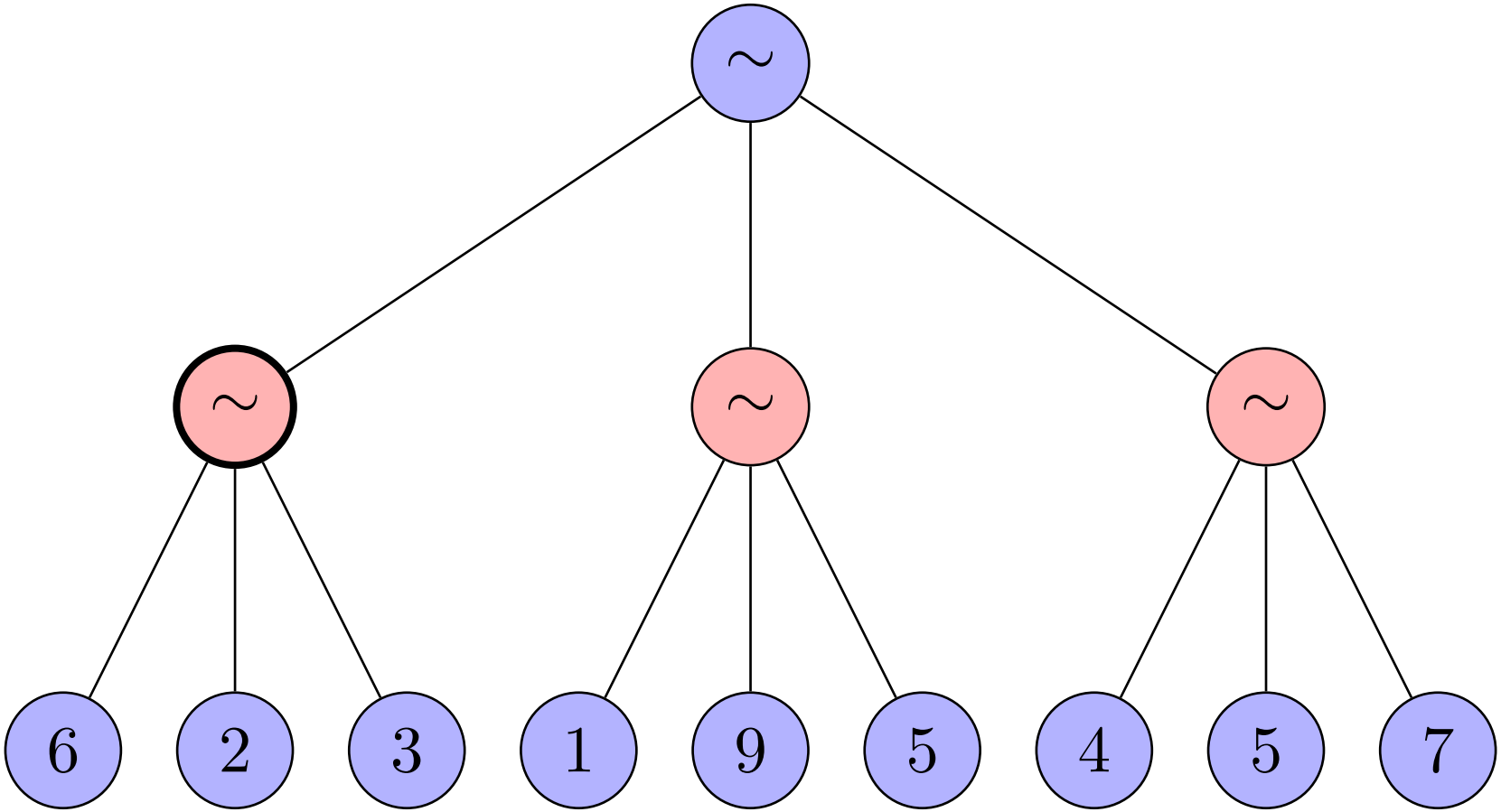
Minimax algoritam

- Za svako stanje znamo koji je igrač na potezu. Jednog igrača zovemo **Max**, a drugog **Min**.
- Posmatrajmo razliku između poena koje ima Max i poena koje ima Min. Pri optimalnoj strategiji oba igrača:
  - Min želi da minimizuje ovu razliku, pri čemu smatra da Max igra optimalno;
  - Max želi da maksimizuje ovu razliku, pri čemu smatra da Min igra optimalno;
  - ova dva cilja se uravnoteže: **minimalan gubitak Min-a je jednak maksimalnom dobitku Max-a** (**minimax teorema**).
- Sada ćemo probati da razmotrimo primer jedne "apstraktne" igre.

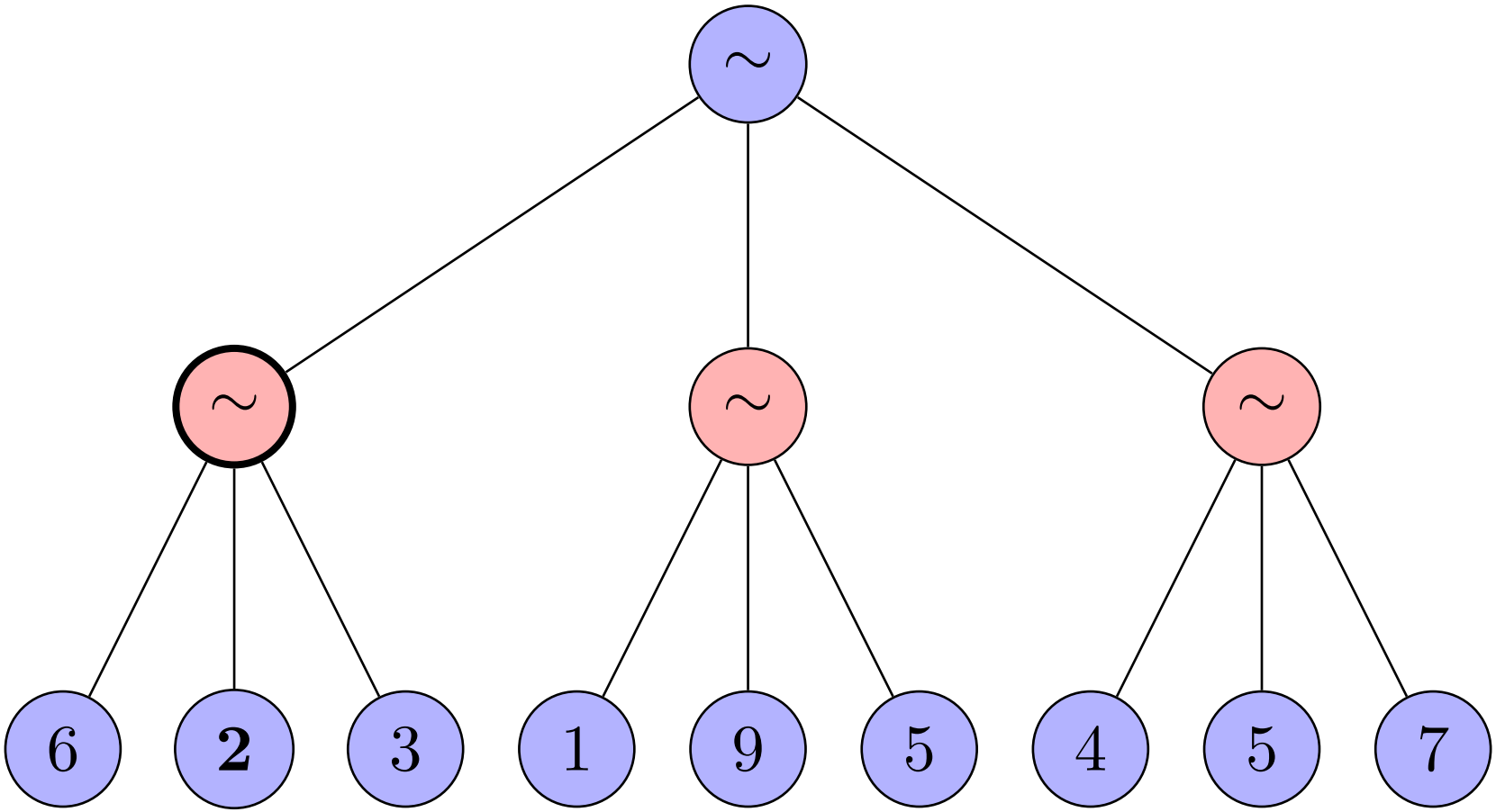
Ilustracija minimax algoritma



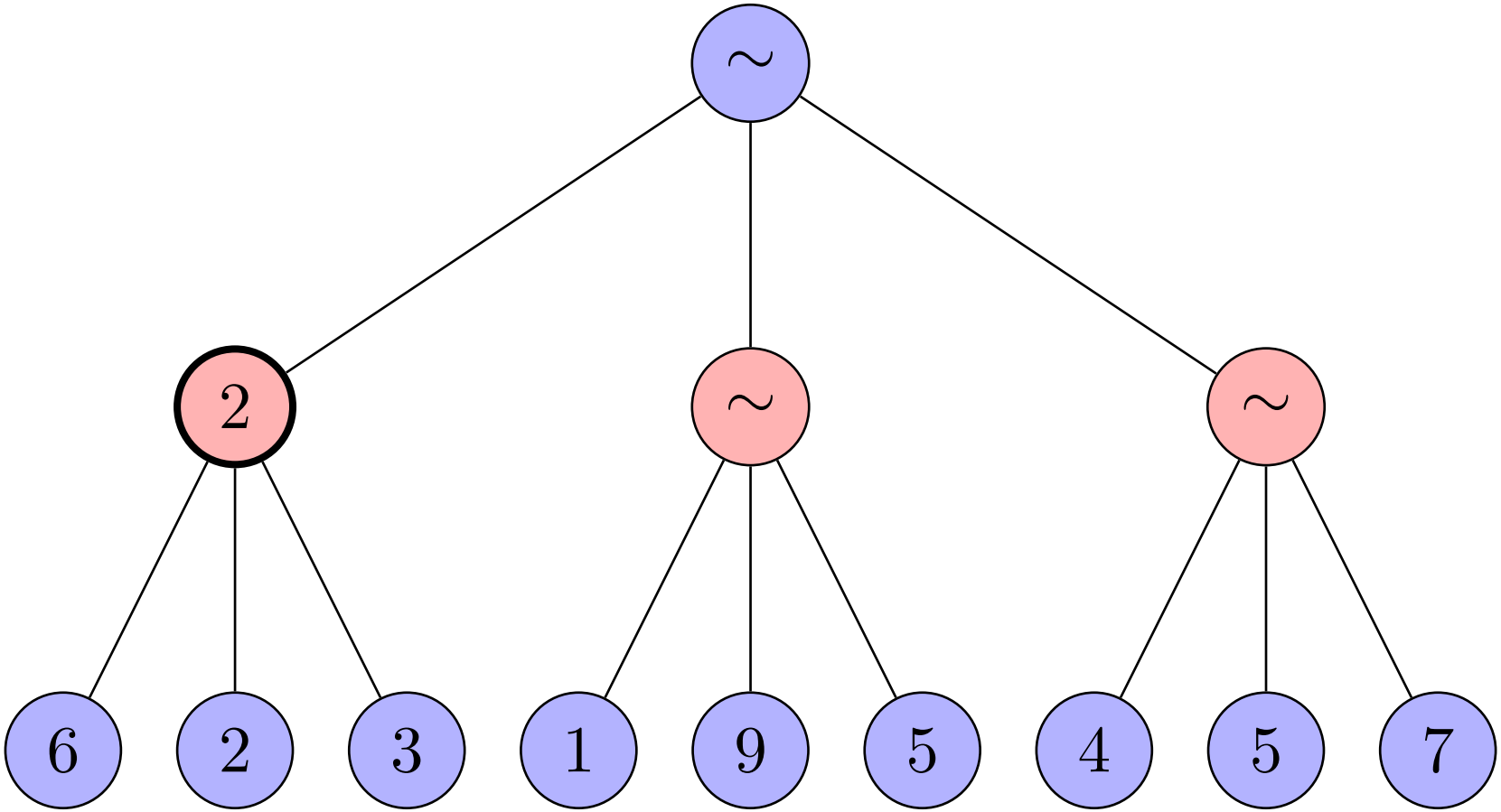
Ilustracija minimax algoritma



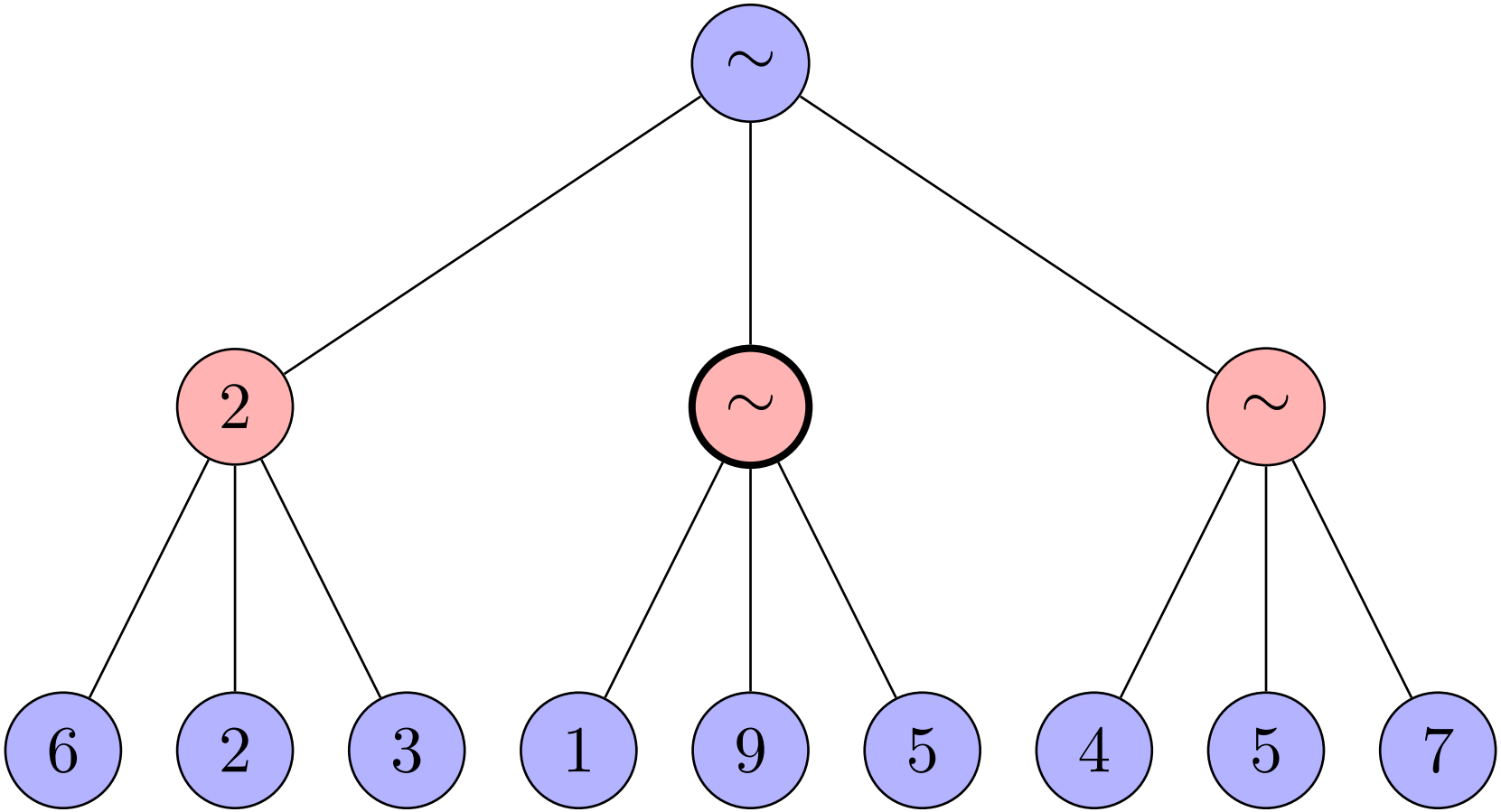
Ilustracija minimax algoritma



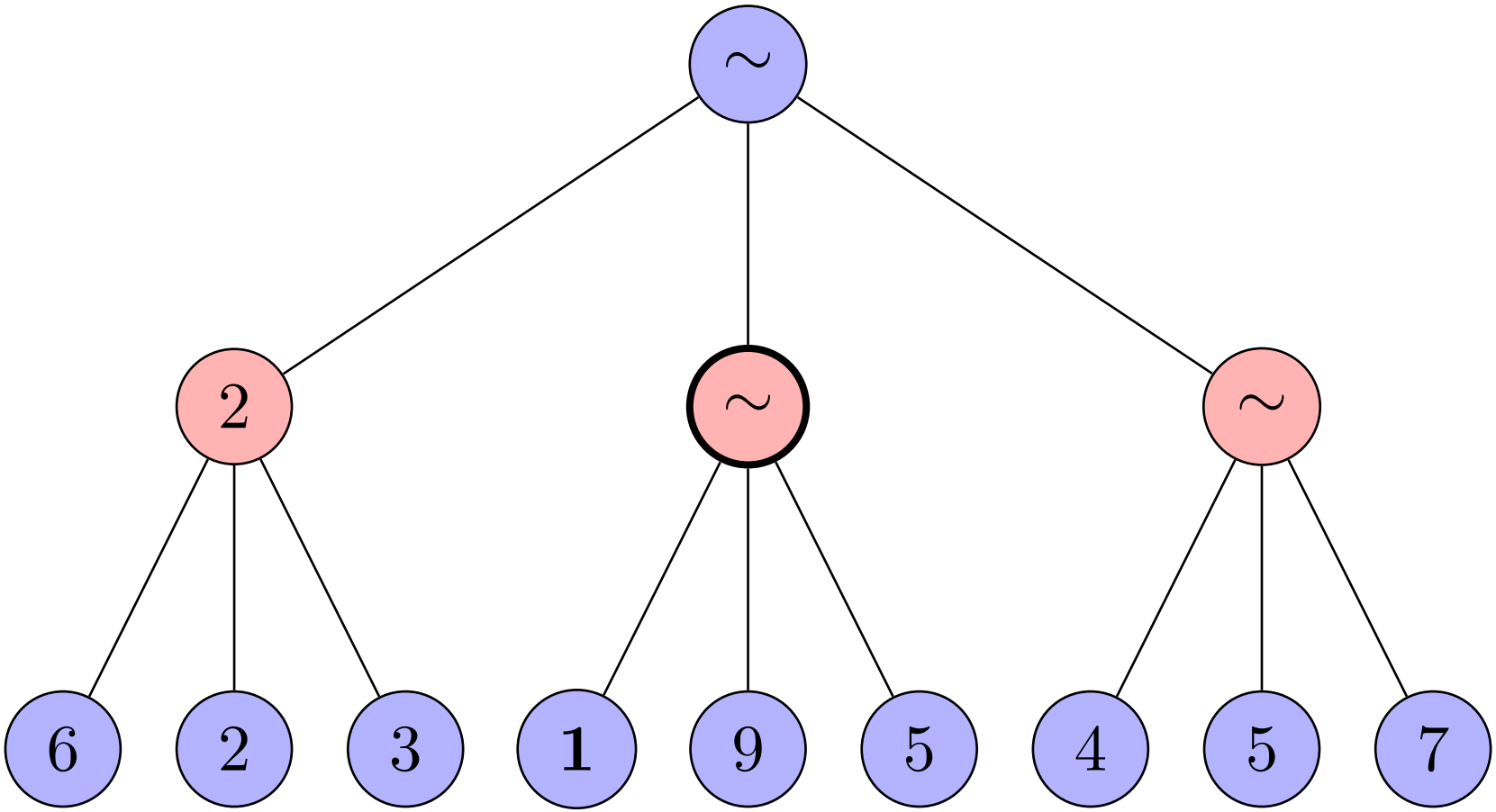
Ilustracija minimax algoritma



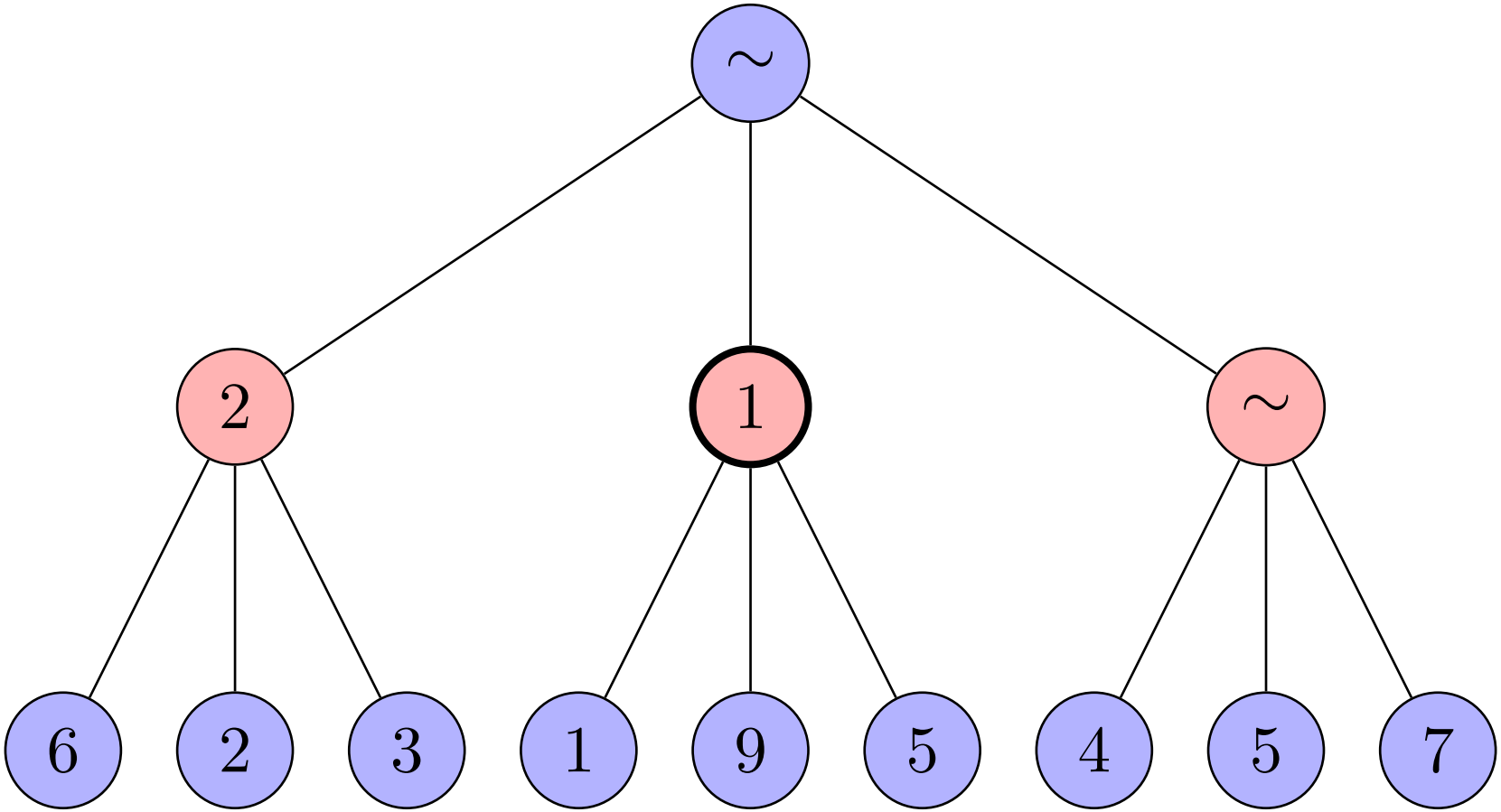
Ilustracija minimax algoritma



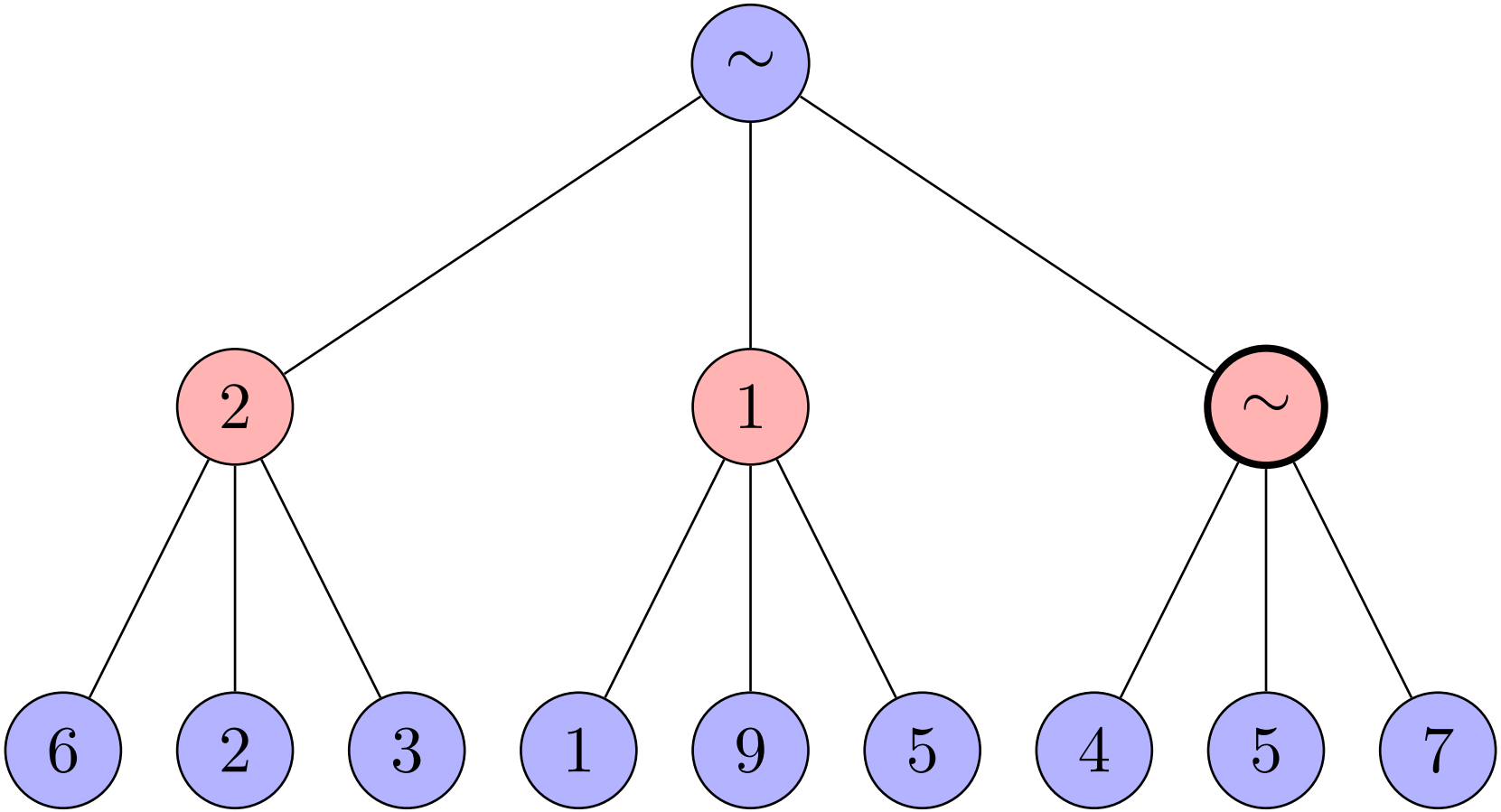
Ilustracija minimax algoritma



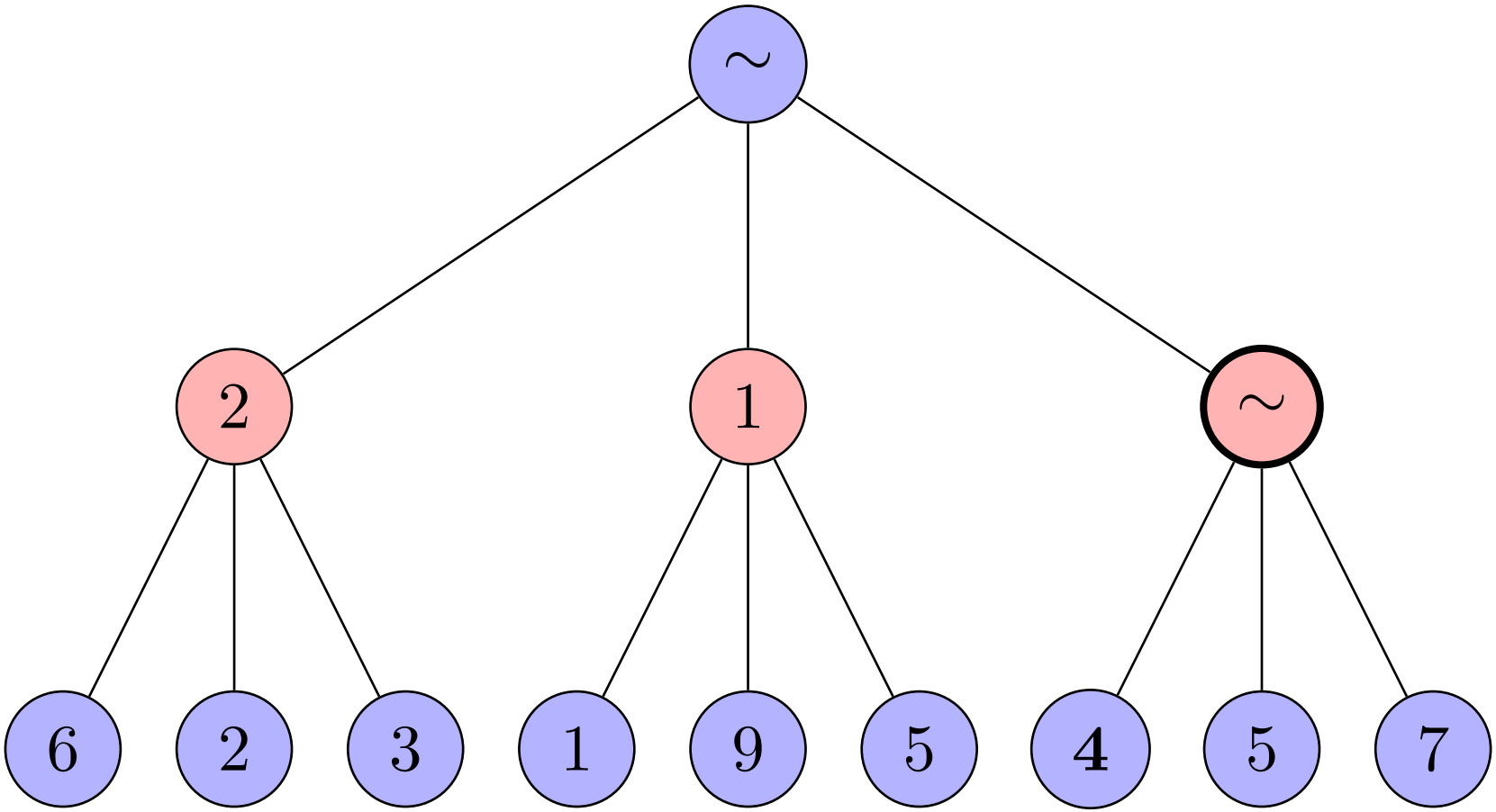
Ilustracija minimax algoritma



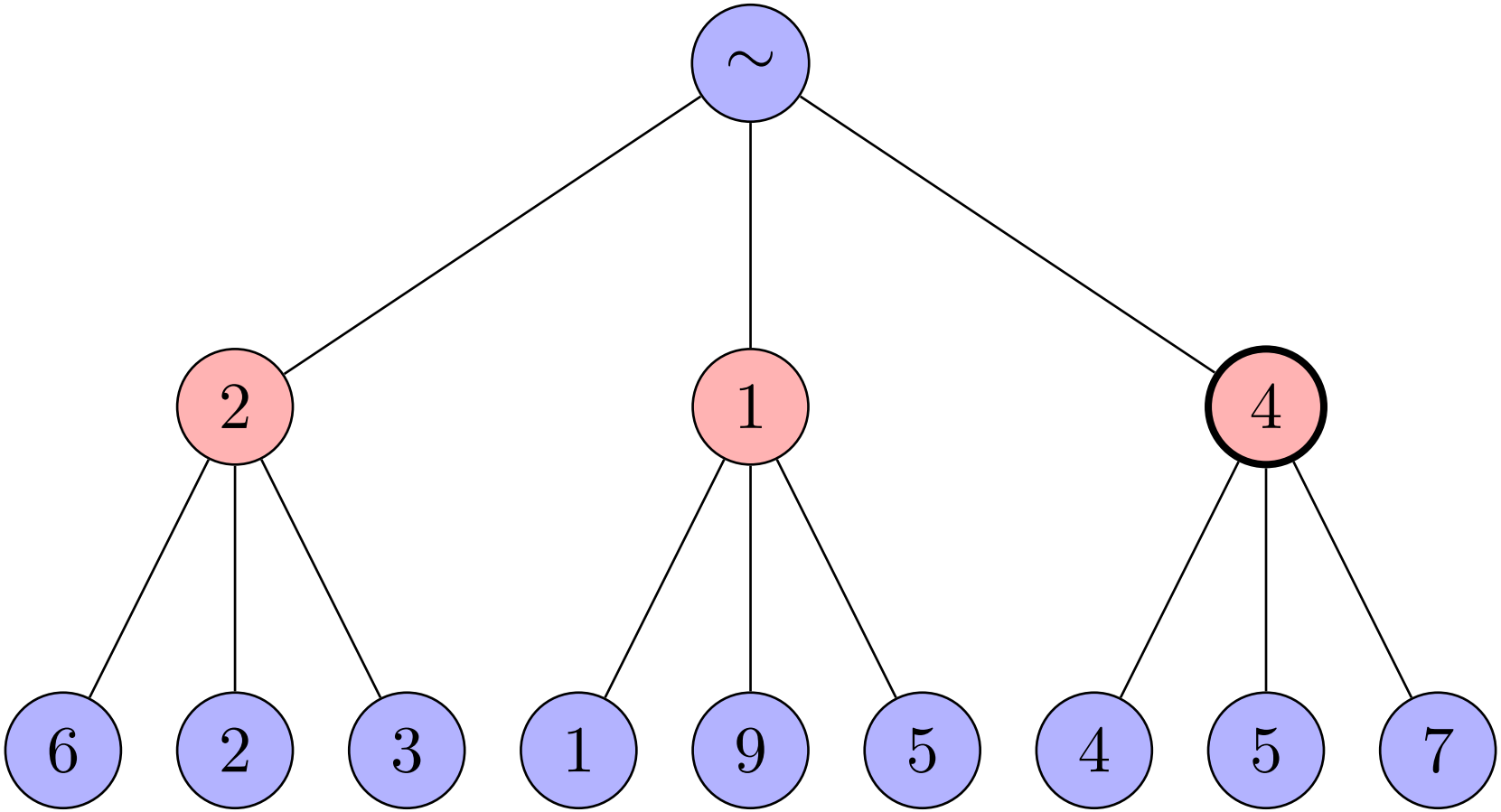
Ilustracija minimax algoritma



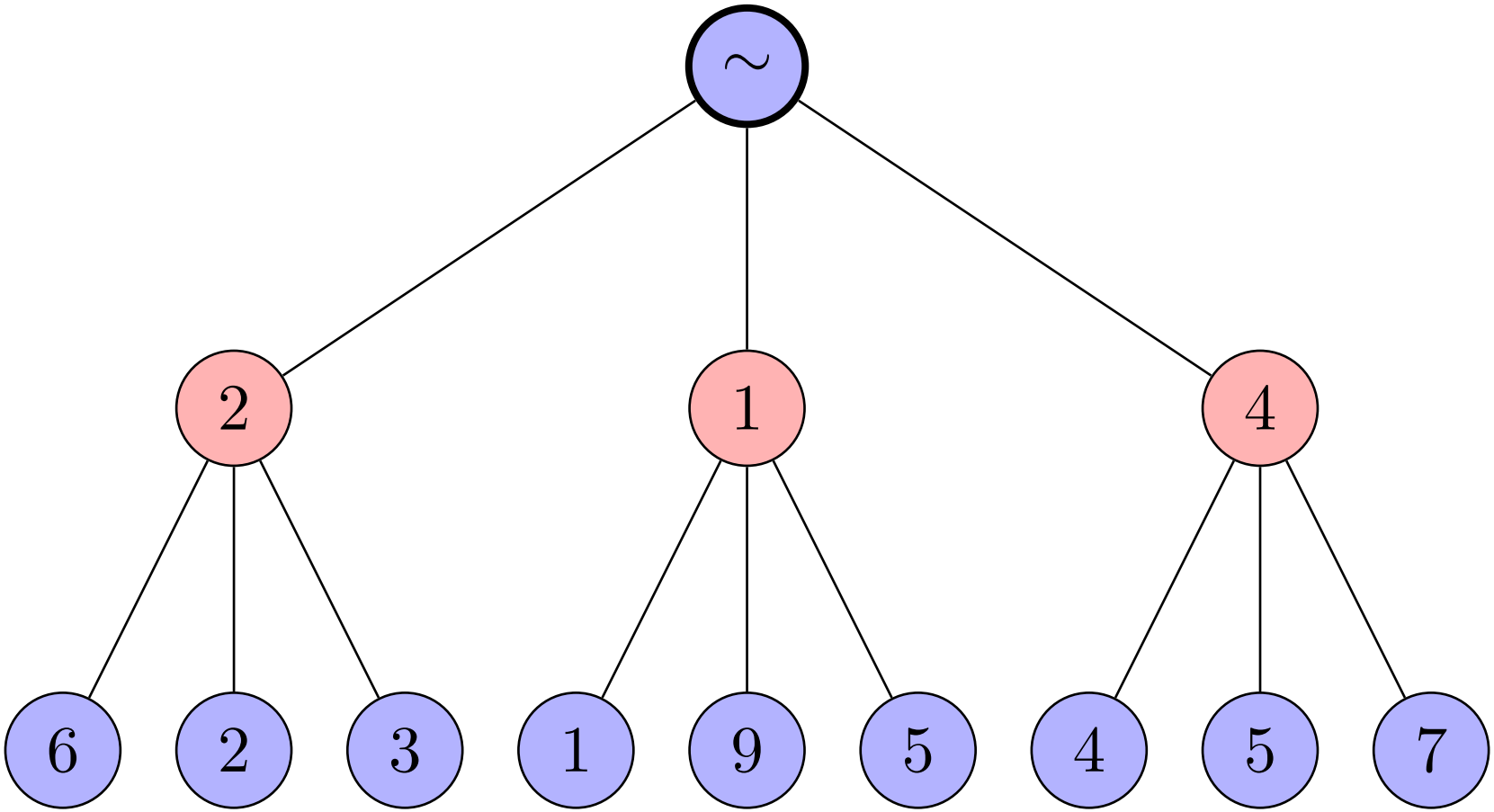
Ilustracija minimax algoritma



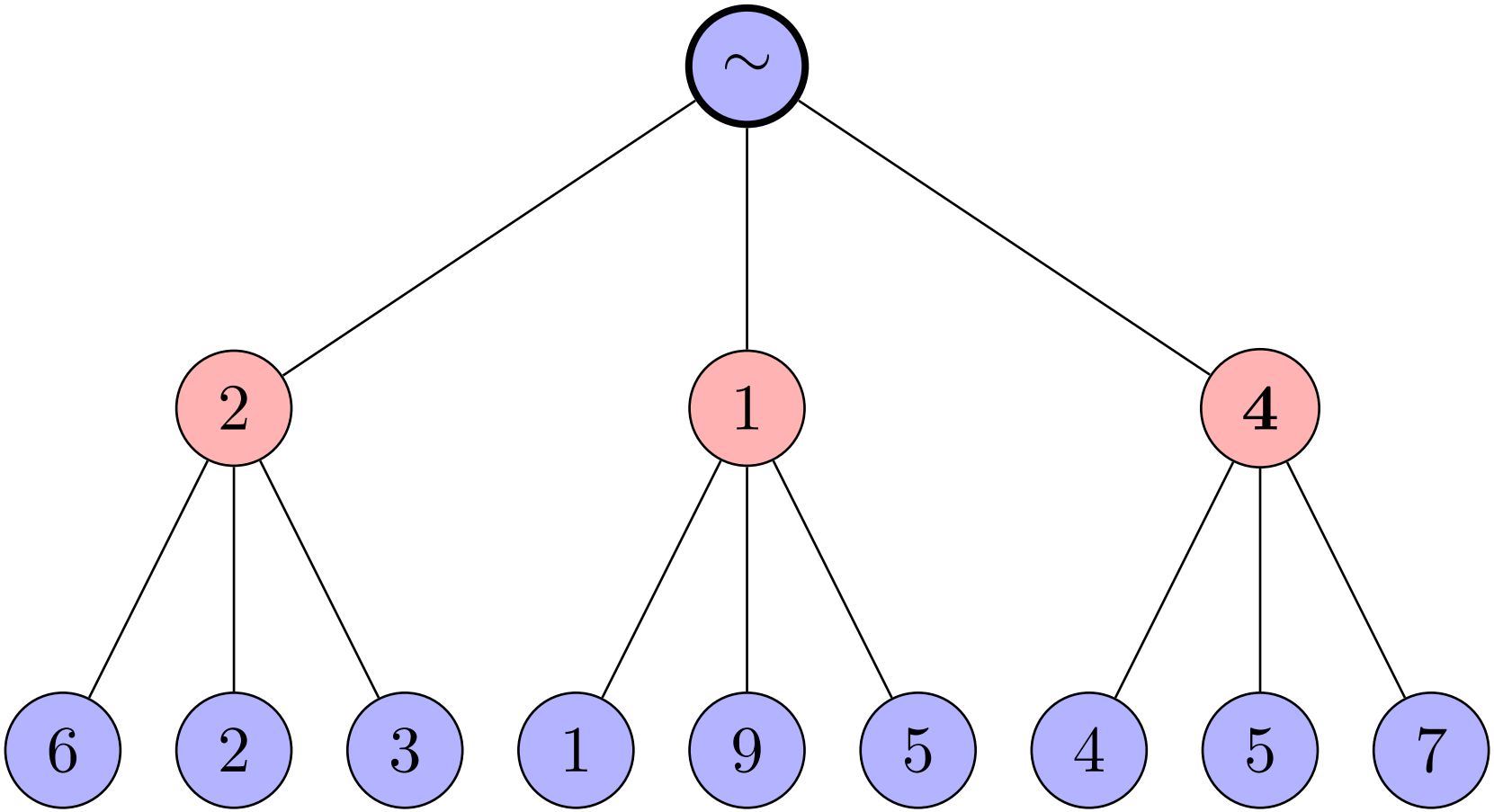
Ilustracija minimax algoritma



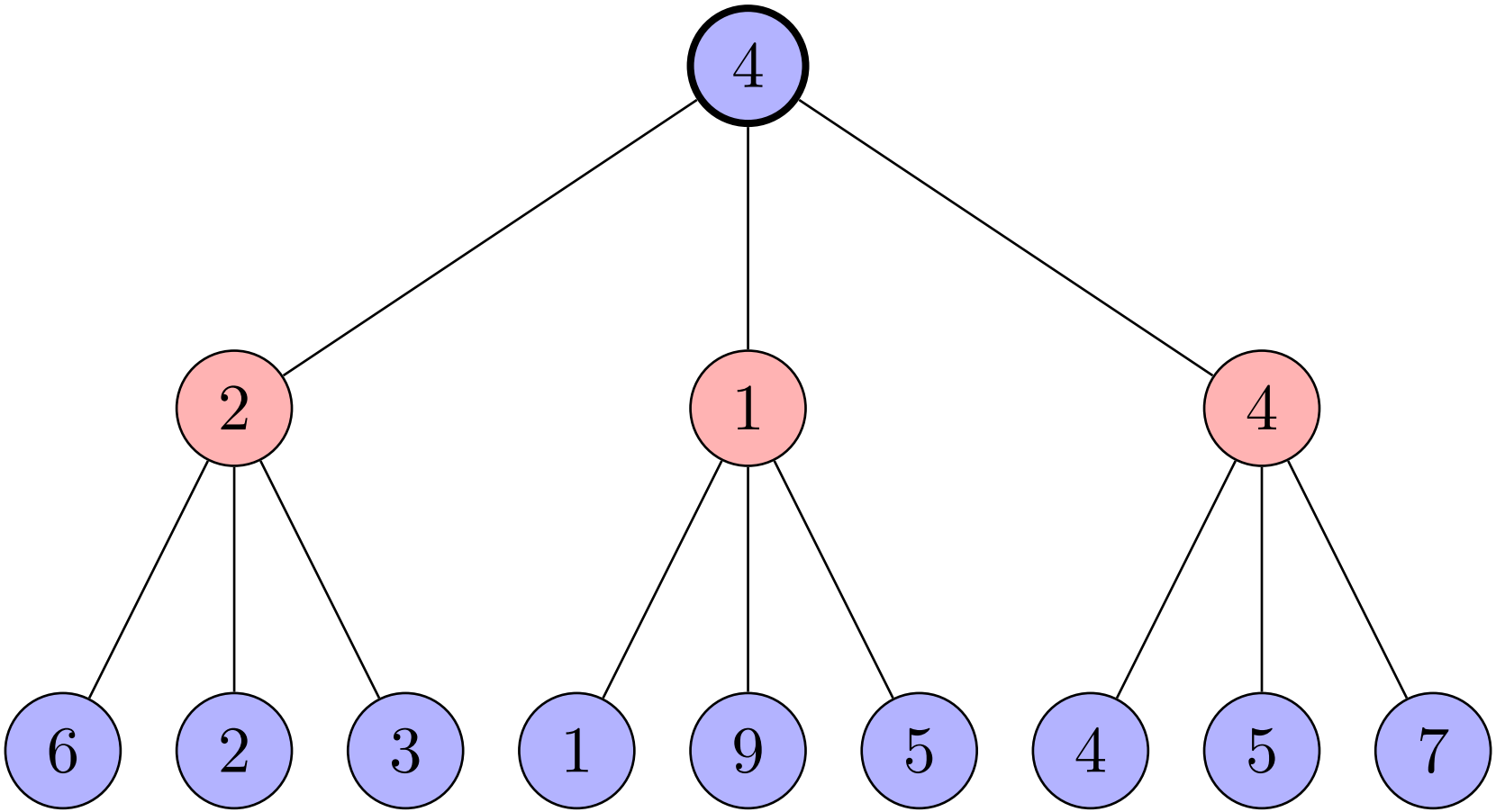
Ilustracija minimax algoritma



Ilustracija minimax algoritma



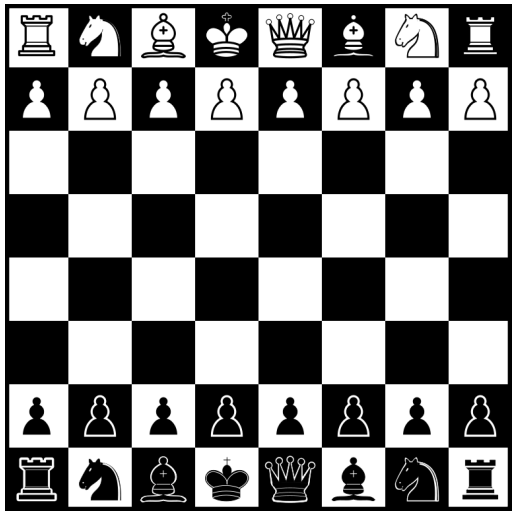
Ilustracija minimax algoritma



Gde dolazi do problema?

- Šta ako su ciklusi dozvoljeni?
- Šta ako do ciklusa ne sme da dođe, ali ga graf igre dozvoljava?
- Šta ako igra ima mnogo veliki broj stanja?
- Jasno je da moramo da pribegnemo nekim **heuristikama**.
- Definišemo ***ply***: broj poteza "unapred" koje razmatramo u svakom trenutku. Po nekim procenama, dobar igrač šaha (ali ne majstor) ima ≈ 3 ply.
- Ali i dalje ne možemo da prikažemo čitavo stablo...

Malo o šahu



- Igra je stara ≈ 1500 godina.
- Istraživanja početka četrdesetih godina, prvi programi razvijani krajem pedesetih.
- Već sedamdesetih počeli da pobeđuju vrhunske igrače.
- 1996. godine *Deep Blue* igra šest partija protija Garija Kasparova sa rezultatom 4–2 (u korist Kasparova). Naredne godine Deep Blue pobeđuje rezultatom $3\frac{1}{2}$–$2\frac{1}{2}$.

Deep Blue

- Razvijan sedam godina od strane IBM-a.
- 256 procesora od po 120 MHz, rezultat je da se u jednoj sekundi obavi $1.138 \cdot 10^{10}$ floating point operacija.

Šta kaže IBM, kako ovo radi?

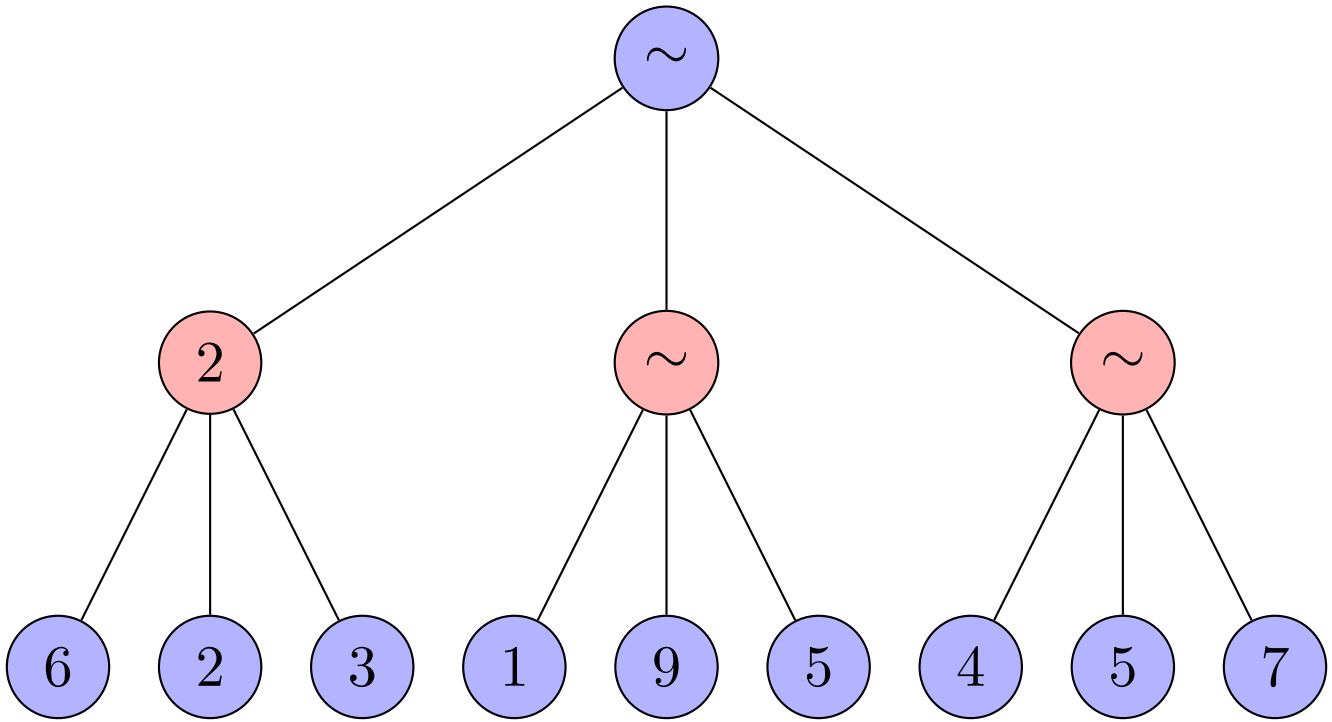
- Definišemo **funkciju procene** (*evaluation function*) eval koja daje "vrednost" svakom položaju i zavisi od sledećih parametara:
  - **materijalna vrednost**: kolika je naša prednost u odnosu na protivnika u smislu "jačine" figura; računa se kao $\sum w_i f_i$, gde je w_i jačina neke figure (1 je za pešaka, 5 je za topa itd), f_i određuje kom igraču pripada (± 1);
  - **poziciona vrednost**: sabrati po svim figurama broj napadnutih polja;
  - **vrednost bezbednosti kralja**: ocenjuje koliko su razna polja "bezbedna", pa onda uzima u obzir njihovu udaljenost od kralja;
  - **vrednost tempa**: zavisi od toga koliko protivnik unapređuje svoj položaj.
- Funkcije procene su česte u igrama ovog tipa, ali je njihovo smišljanje u opštem slučaju težak problem.

Šta sada sa eval?

- Primenimo minimax algoritam, ali ne na celom stablu, već sa ograničenom dubinom (ply agenta). Listovi ovog stabla sadrže vrednosti relevantne za minimax.
- Međutim, čak i uz ovu aproksimaciju, na jednom savremenom računaru je neophodno $\approx 2\,\mathrm{min}$ za 3–4 ply. Kako da pobedimo velemajstore?
- Problem je u tome što razmatramo dosta neoptimalnih stanja u minimax algoritmu. Moramo nekako da ih se otarasimo!

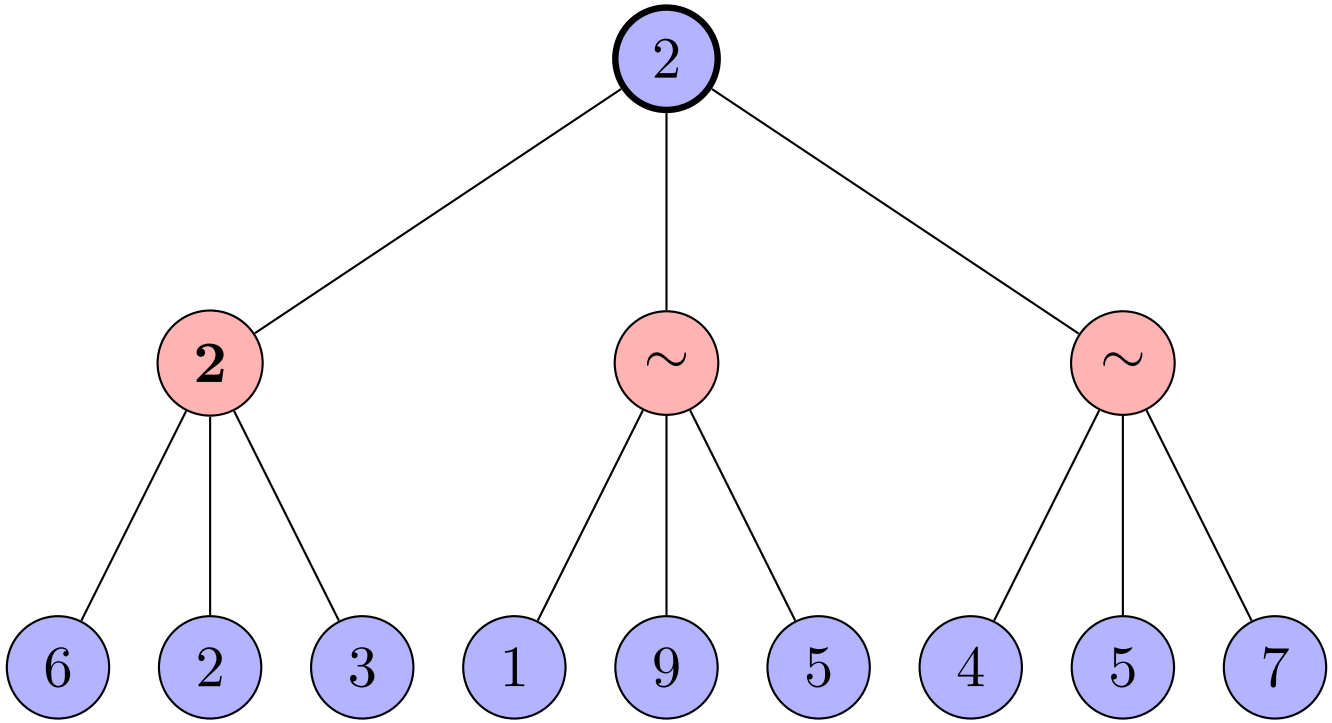
α–β rez

- Vratimo se primeru izvršavanja minimax algoritma.
- Možemo da postepeno ažuriramo koren, a neki čvorovi impliciraju da ne vredi razmatrati potez...



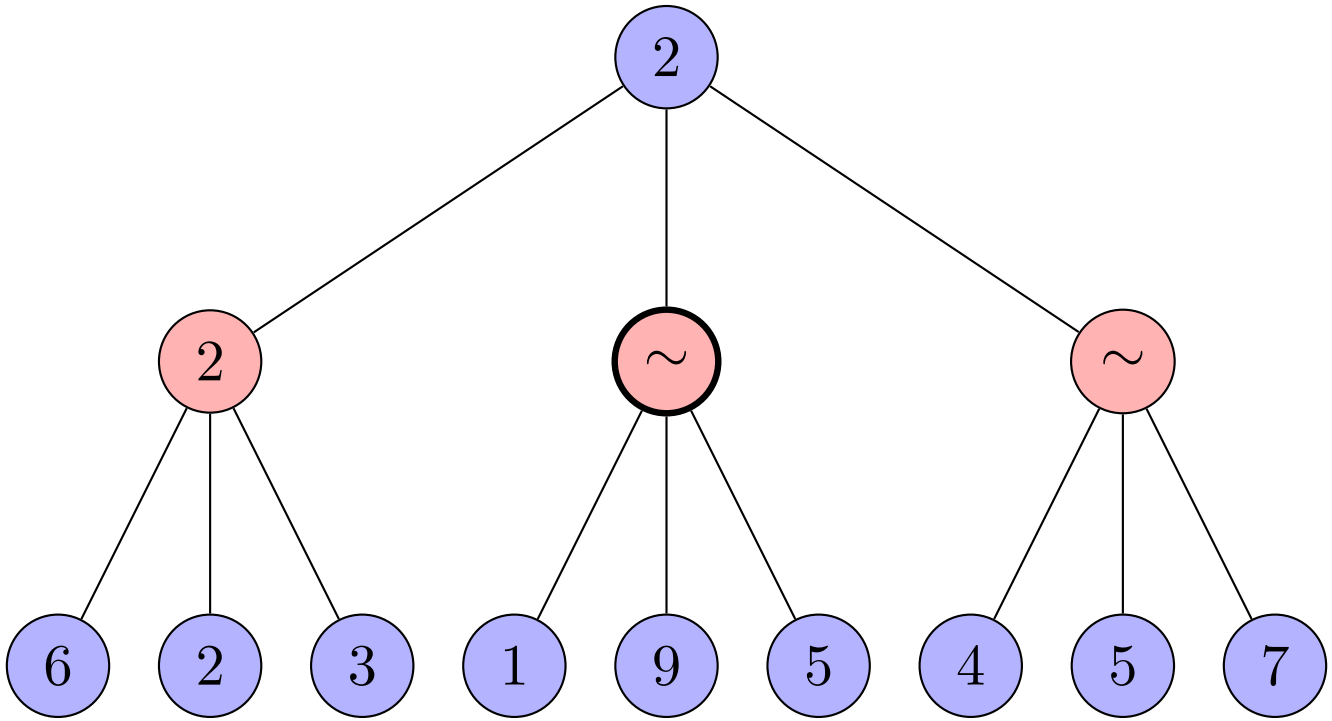
α–β rez

- Vratimo se primeru izvršavanja minimax algoritma.
- Možemo da postepeno ažuriramo koren, a neki čvorovi impliciraju da ne vredi razmatrati potez...



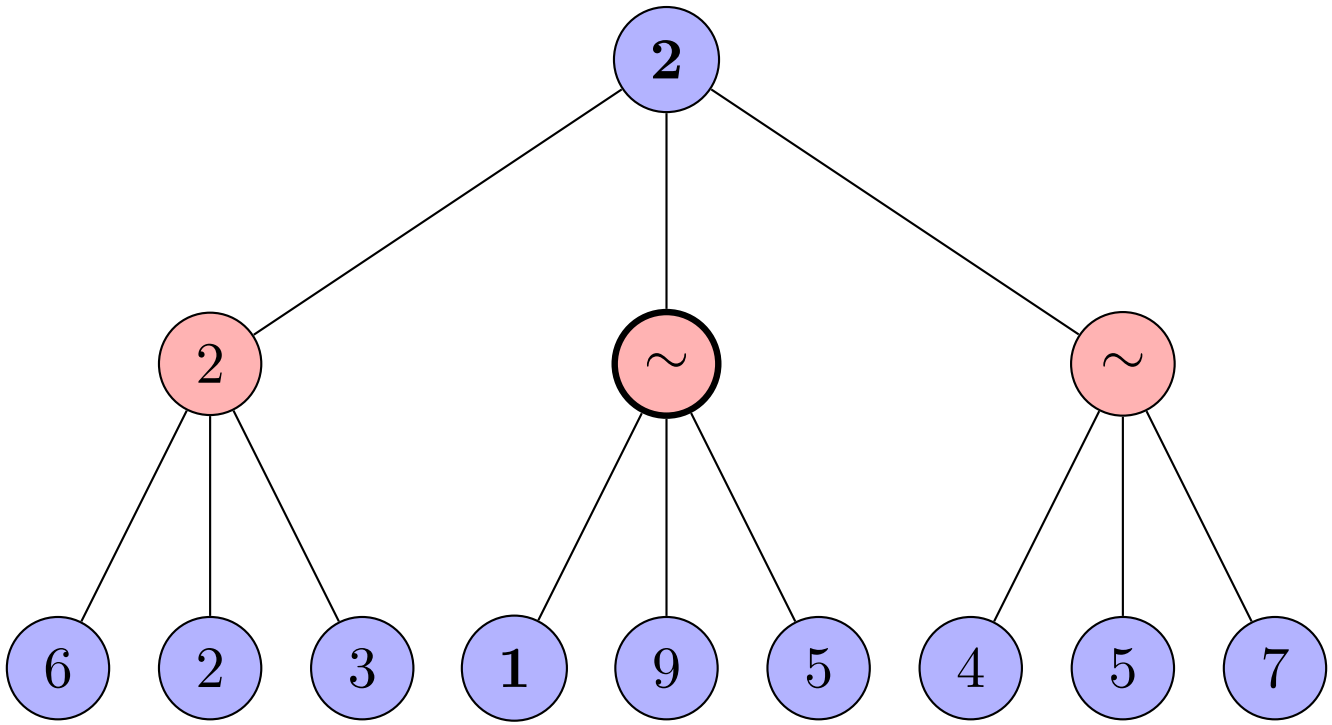
α–β rez

- Vratimo se primeru izvršavanja minimax algoritma.
- Možemo da postepeno ažuriramo koren, a neki čvorovi impliciraju da ne vredi razmatrati potez...



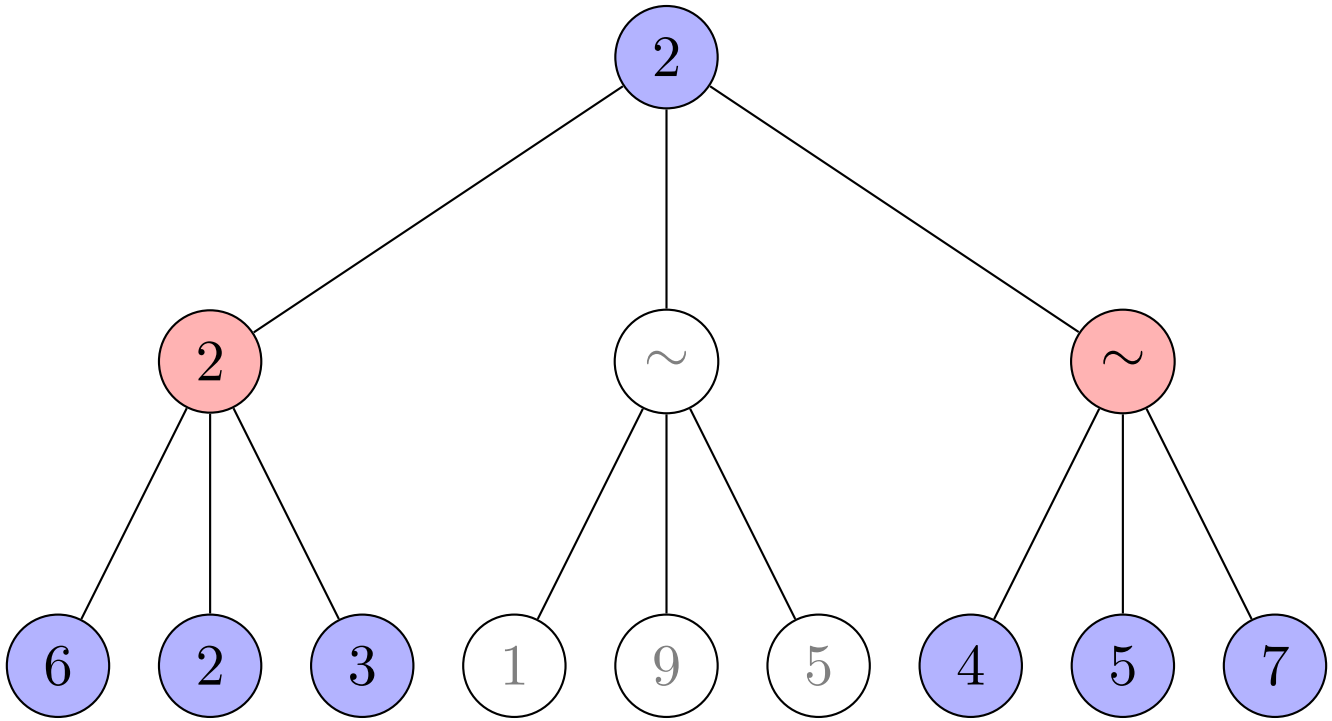
α–β rez

- Vratimo se primeru izvršavanja minimax algoritma.
- Možemo da postepeno ažuriramo koren, a neki čvorovi impliciraju da ne vredi razmatrati potez...



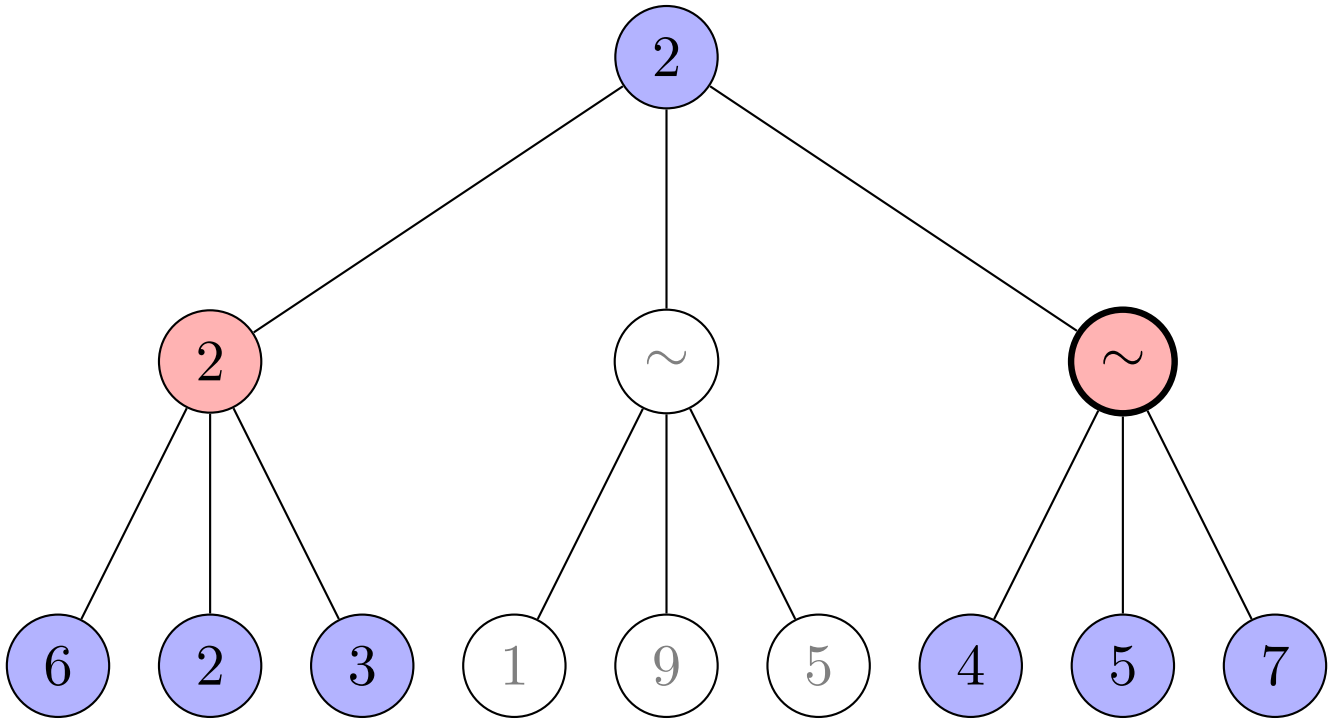
α–β rez

- Vratimo se primeru izvršavanja minimax algoritma.
- Možemo da postepeno ažuriramo koren, a neki čvorovi impliciraju da ne vredi razmatrati potez...



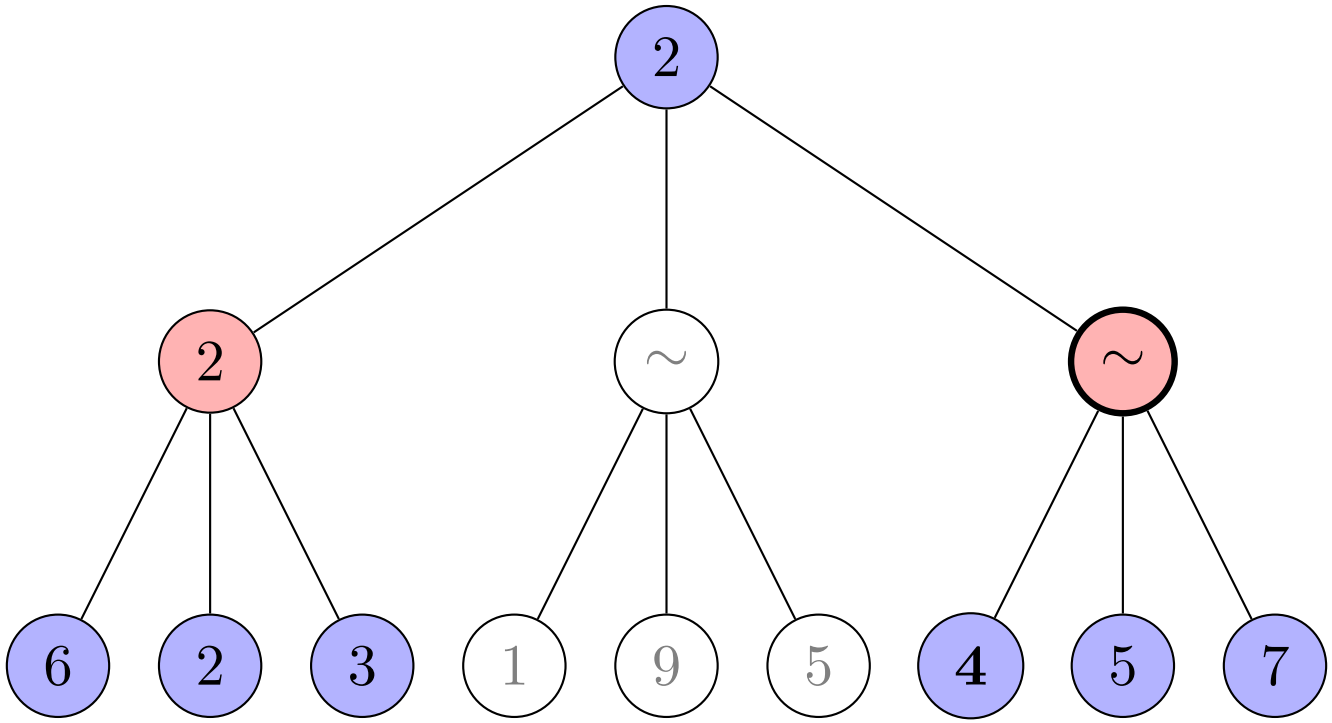
α–β rez

- Vratimo se primeru izvršavanja minimax algoritma.
- Možemo da postepeno ažuriramo koren, a neki čvorovi impliciraju da ne vredi razmatrati potez...



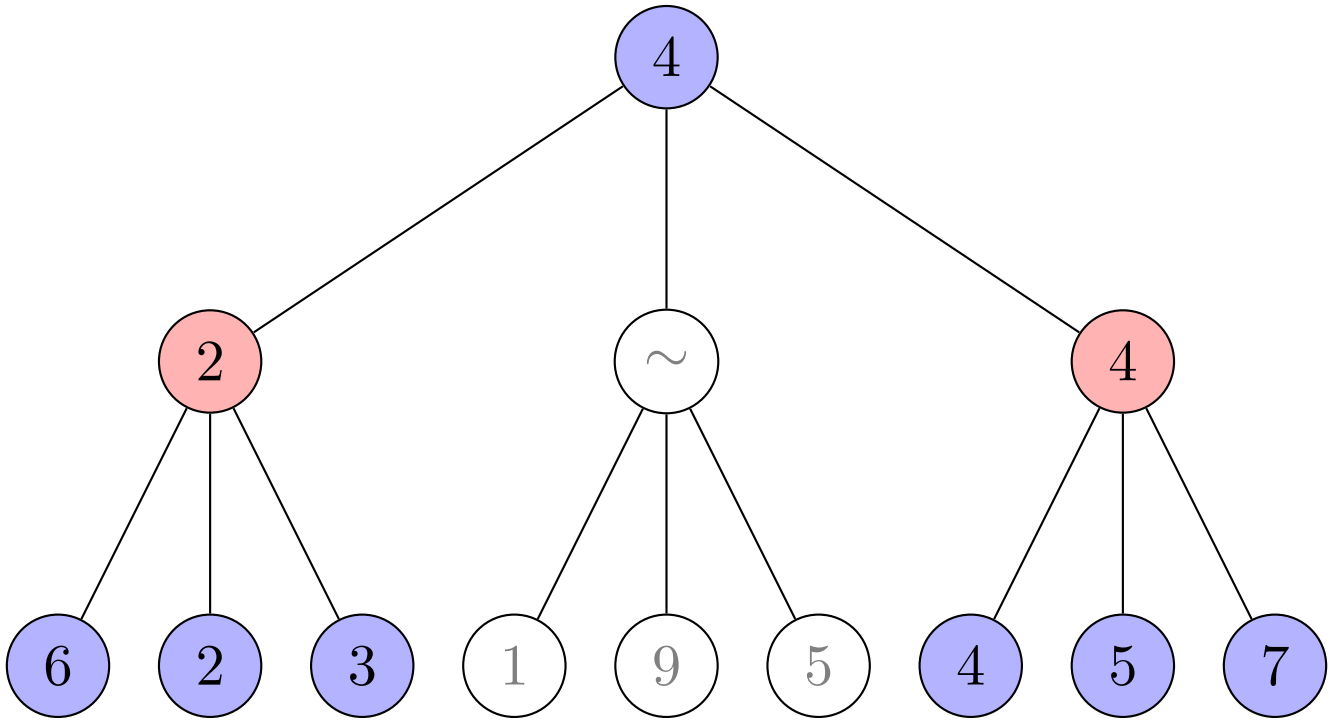
α–β rez

- Vratimo se primeru izvršavanja minimax algoritma.
- Možemo da postepeno ažuriramo koren, a neki čvorovi impliciraju da ne vredi razmatrati potez...



α–β rez

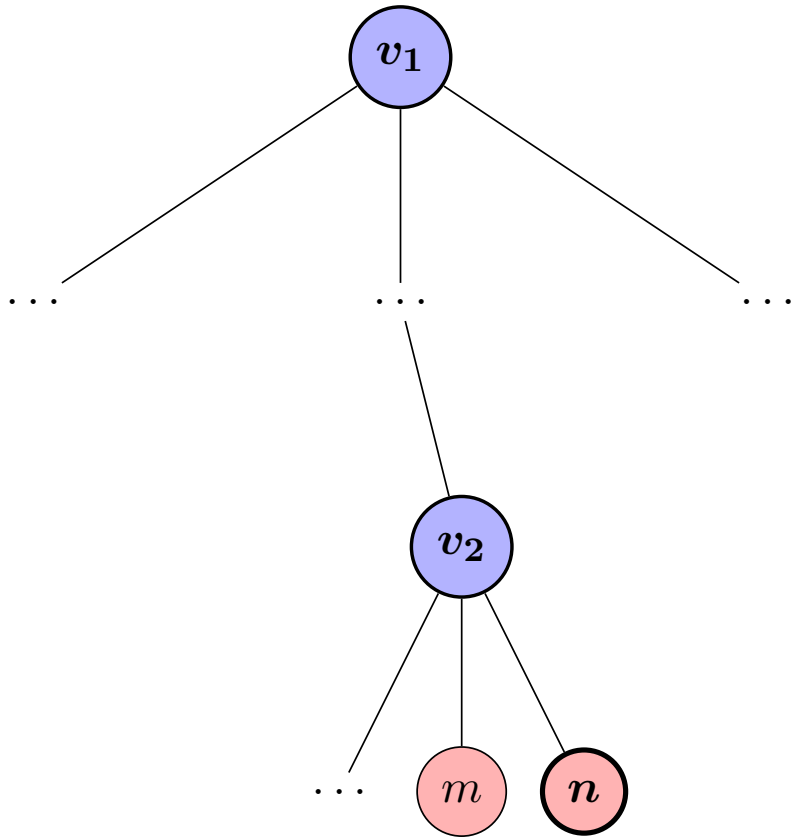
- Vratimo se primeru izvršavanja minimax algoritma.
- Možemo da postepeno ažuriramo koren, a neki čvorovi impliciraju da ne vredi razmatrati potez...



α–β rez u opštem slučaju

- U stvari je reč o **pretrazi po dubini** (DFS), pri čemu pri razmatranju nekog čvora razmatramo samo čvorove na putu do njega.
- Na tom putu, neka je α najveća vrednost koju može da postigne Max, a β najmanja vrednost koju može da postigne Min (na samom početku se podesi $\alpha = -\infty$, $\beta = +\infty$).
- Koristićemo funkcije `player(Node* v)` i `opponent(Node* v)`, a na početku se pozove `player(root)`.

- Vrednosti α i β su podložni promenama, način kako se vidi iz razmatranja $n < m$ i $n < \alpha$.
- Nemamo puno slučajeva...



Pseudokod za player(Node* v)

```
1  int player(Node* v)
2  {
3      if (cutOff(v)) return eval(v);
4      value = -INFINITY;
5      foreach (u : v -> children)
6      {
7          value = max(value, opponent(u));
8          if (value > BETA) return value;
9          if (value > ALPHA) ALPHA = value;
10     }
11     return value;
12 }
```

Kod za `opponent(Node* v)` je analogan...

Koliko je dobar α–β rez?

- Ignorišimo detalje oko predstavljanja stanja, interesuje nas isključivo vreme.
- Za početak, minimax je imao vremensku složenost $O\left(q^p\right)$, gde je q faktor račvanja (broj mogućih poteza u svakom stanju), a p traženi ply.
- Za α–β rez ponašanje dosta zavisi od redosleda kojim se obilaze čvorovi. U savršenom slučaju bismo mogli da imamo $O\left(q^{\frac{p}{2}}\right)$.
- Za α–β rez može da se dokaže da je asimptotska složenost:

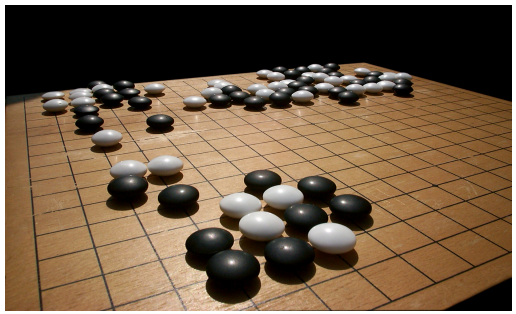
$$O\left(\left(\frac{q}{\log q}\right)^p\right),$$

  odnosno, za praktične vrednosti q približno $O\left(q^{\frac{3p}{4}}\right)$.
- U praksi uspevamo da odredimo redoslede koji nam pomažu da udvostručimo ply.

"The Fathers of the field had been pretty confusing: John von Neumann speculated about computers and the human brain in analogies sufficiently wild to be worthy of a medieval thinker and Alan M. Turing thought about criteria to settle the question of whether Machines Can Think, a question of which we now know that it is about as relevant as the question of whether Submarines Can Swim."

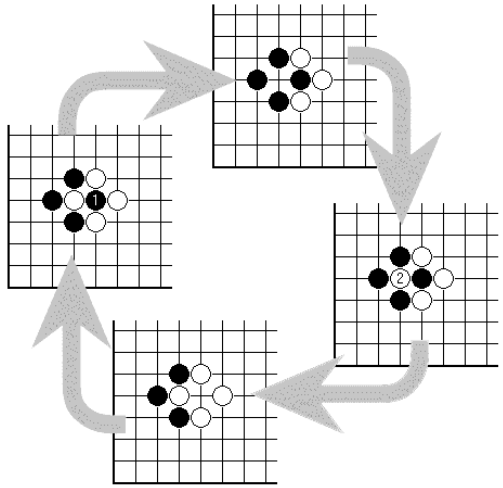
Edsger W. Dijkstra

Pravila igre go



- Data je tabla sa 19 horizontalnih i 19 vertikalnih linija, na početku je prazna.
- Crni i Beli imaju žetone koje naizmenično stavljaju na tablu. Crni igra prvi.
- Žetoni se stavljaju na preseke linija.
- Igrač može da preskoči jedan potez bilo kada, ali mora da žrtvuje jedan svoj žeton.
- Igra se završava nakon dva uzastopna preskočena poteza, pri čemu je Beli taj koji završava igru.

Pravila igre go

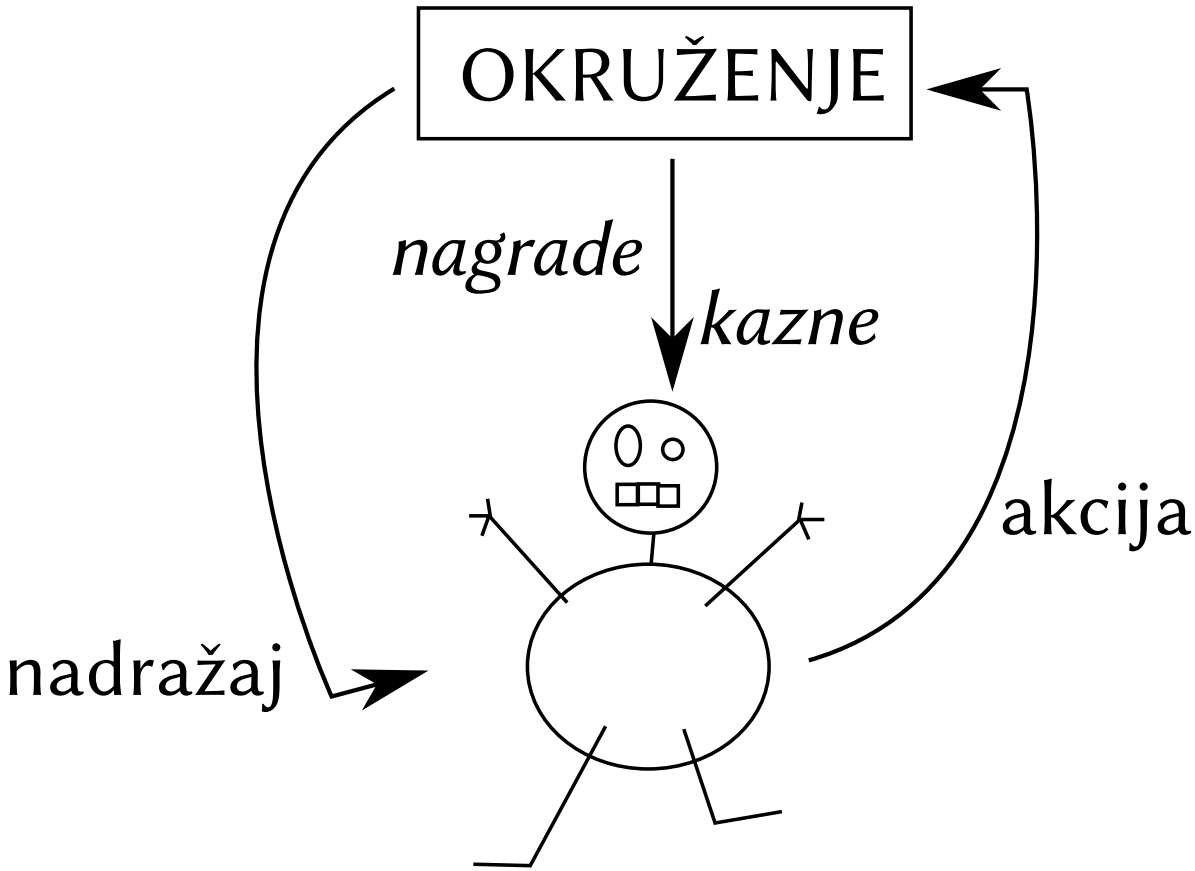


- Teritorija nekog igrača podrazumeva broj preseka na kojima su žetoni tog igrača ili koji su žatvoreni" žetonima tog igrača.
- Pobeđuje igrač sa više teritorije.
- Ukoliko nakon poteza igrača A neka grupa žetona B postaje "potpuno okružena" žetonima A, ta grupa se sklanja sa table.
- **Superko pravilo**: ne sme da se ponovi nijedan položaj koji se javio ranije u toku igre.

Teškoće

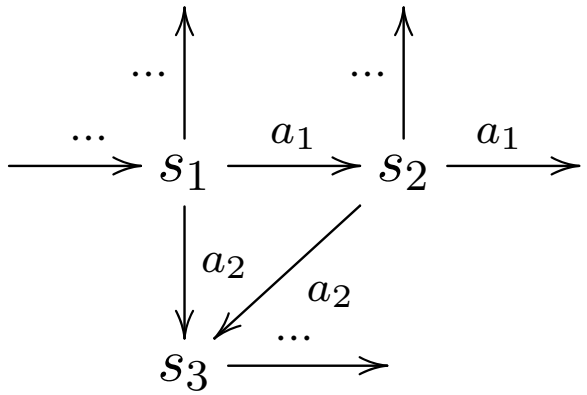
- Tabla je mnogo veća nego u šahu, mnogo više mogućnosti.
- Predstavljanje stanja je problematično, mora da se zadovolji superko pravilo.
- Vizuelna priroda igre: ljudima ovo ide bolje!
- U suštini, jedino što sada možemo da radimo u ovom trenutku je da iskoristimo "još nepotpuniju" varijantu α–β reza, gde se razmatra samo nasumično odabran podskup poteza: spada u **Monte Karlo** algoritme.
- Ipak, postoje neki uspesi: 2013. pobeđuje 9. dan igrača uz početnu prednost od tri žetona...

Šta ovo podrazumeva?



Okruženje i reakcije agenata

- Pokušaćemo da se bavimo učenjem koje neće ni na koji način biti u vezi sa teorijom verovatnoće i matematičkom analizom (*ha ha ha!*).
- Okruženje je zadato nekim stanjima $s_1, s_2, \ldots, s_n$.
- Agent može u svakom stanju da donese iste odluke, nazovimo ih akcijama $a_1, a_2, \ldots, a_m$.
- Uvodimo pretpostavku da je okruženje deterministično, te možemo da modelujemo grafom:



Funkcije promene stanja i nagrade

- Skup stanja: $S = \{s_1, s_2, \ldots, s_n\}$.
- Skup akcija: $A = \{a_1, a_2, \ldots, a_m\}$.
- Pošto je svet deterministički, za zadato stanje i akciju je jedinstveno određeno naredno stanje. Stoga je prirodno uvesti **funkciju promene stanja**: $\mathcal{S} : S \times A \to S$.
- Agent želi da uradi neki zadatak. Smatraćemo da umemo da prevedemo ovaj zadatak u "maksimizuj neki broj poena". Kažemo da primena akcije u nekom stanju donosi neku nagradu.
- Zato uvodimo **funkciju nagrade**: $\mathcal{R} : S \times A \to \mathbb{R}$.
- Dakle, koje su značenja $\mathcal{S}(s, a)$ i $\mathcal{R}(s, a)$?

Funkcija polise

- Agent na početku **nema informacije** o funkcijama $\mathcal{S}$ i $\mathcal{R}$.
- U toku učenja treba da formira **funkciju polise** $p : S \to A$ koja za svako stanje određuje koju akciju bi trebalo sprovesti. Kako sada odabrati optimalnu polisu?
- Ako je startno stanje $s \in S$, za datu polisu p definišimo:
  - $s'_1 = s$, $s'_2 = \mathcal{S}\left(s'_1, p\left(s'_1\right)\right), \ldots, s'_k = \mathcal{S}\left(s'_{k-1}, p\left(s'_{k-1}\right)\right), \ldots$;
  - $r_k = \mathcal{R}\left(s'_k, p\left(s'_k\right)\right)$.
- Tražimo neku ocenu polise p. Jedna mogućnost je naredna, gde je γ naša procena odnosa nagrade koja je dostupna sada i nagrade koja je dostupna posle (**discount factor**):

$$V^p\left(s\right) = \sum_{k=1}^{\infty} \gamma^{k-1} r_k = r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots$$

$\mathcal{Q}$ funkcija

- Želimo da nađemo p_{opt}, gde je $p_{\text{opt}}(s) = \arg\max_p \{V^p(s)\}$ za dato startno stanje s. Uvedimo i oznaku $V_{\text{opt}} = V^{p_{\text{opt}}}$.
- Nalaženje optimalne polise nije teško ukoliko znamo $\mathcal{S}$ i $\mathcal{R}$...
- Definišimo funkciju $\mathcal{Q}$, gde $\forall s \in S, a \in A$:

$$\mathcal{Q}(s,a) = \mathcal{R}(s,a) + \gamma V_{\text{opt}}(\mathcal{S}(s,a))$$

- Značenje $\mathcal{Q}$ odgovara kvalitetu neke akcije ukoliko nakon nje pratimo optimalnu polisu.
- Naučiti $\mathcal{Q}$ je dovoljno za ustanovljavanje optimlne polise jer:

$$p_{\text{opt}}(s) = \arg\max_a \{\mathcal{Q}(s,a)\}$$

Ko se usuđuje da ispiše formalan dokaz?

Nagoveštaj iterativnog postupka

- Primetimo $V_{\text{opt}}(s) = \max_{\alpha}\{\mathcal{Q}(s,\alpha)\}$.
- Stoga, kada ovo uvrstimo u definiciju $\mathcal{Q}$:

$$\mathcal{Q}(s,a) = \mathcal{R}(s,a) + \gamma \max_{\alpha}\{\mathcal{Q}(\mathcal{S}(s,a),\alpha)\}$$

- Ovo nagoveštava da, ako je $\mathcal{Q}'$ **trenutna procena** za $\mathcal{Q}$, da će

$$\mathcal{R}(s,a) + \gamma \max_{\alpha}\{\mathcal{Q}(\mathcal{S}(s,a),\alpha)\}$$

biti bolja procena za $\mathcal{Q}(s,a)$ od $\mathcal{Q}'(s,a)$. Pri tome je $\mathcal{Q}'$ tabela aktuelnih procena za $\mathcal{Q}$ za sve elemente $S \times A$.

Algoritam $\mathcal{Q}$-učenja

Na početku je $\mathcal{Q}'$ tabela popunjena nasumično odabranim vrednostima.

1. Trenutno stanje je s, isprobaj neku akciju a (odabir a je magija verovatnoće!).
2. Uradi a, sleduje nagrada $\mathcal{R}(s, a)$.
3. Posmatraj novo stanje $\mathcal{S}(s, a)$.
4. Ažuriranje:
$$\mathcal{Q}'(s, a) \leftarrow \mathcal{R}(s, a) + \gamma \max_{\alpha} \left\{ \mathcal{Q}'(s, a), \alpha \right\}$$
5. Idi na 1.

Šta nam ovo govori?

- Agent pokušava da maksimizuje nagradu, ali "ne razume" šta, u stvari, radi.
- Koliko je ovo slično čoveku?
- Možemo simpatične stvari da uradimo ako na ispravan način omogućimo definisanje i čitanje $\mathcal{R}$.
- Google DeepMind, učenje bez konteksta (jedini ulazni podaci su pikseli!):