

Uczenie ze wzmocnieniem w klasycznych grach

Dlaczego klasyczne gry?

- Pomimo prostych zasad ich rozwiązanie zazwyczaj wymaga poznania złożonych strategii.
- Są doskonałym poligonem doświadczalnym i benchmarkiem dla nowych algorytmów.
- Wiele problemów napotkanych podczas ich rozwiązywania ma przełożenie na rzeczywiste zagadnienia.

Strategia optymalna

- to taka strategia, która prowadzi do najkorzystniejszego rezultatu z punktu widzenia danego gracza, przy założeniu, że strategie wszystkich jego przeciwników są znane i ustalone:

$$\pi_*^i(\pi^{-i})$$

- O **równowadze Nasha** mówimy gdy każdy z graczy wybiera strategię optymalną względem stałych i niezmiennych (optymalnych) strategii innych graczy.

Gra o sumie stałej

- Większość typowych gier planszowych jest grami o sumie stałej i doskonałej informacji, często też są dwuosobowe.
- Dzięki temu znalezienie w nich równowagi jest szczególnie proste.

Minimax

- Dla dwuosobowej gry funkcja wartości wyznacza oczekiwaną skumulowaną nagrodę pod warunkiem, że gracze grają za pomocą strategii: $\pi = (\pi_1, \pi_2)$:

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

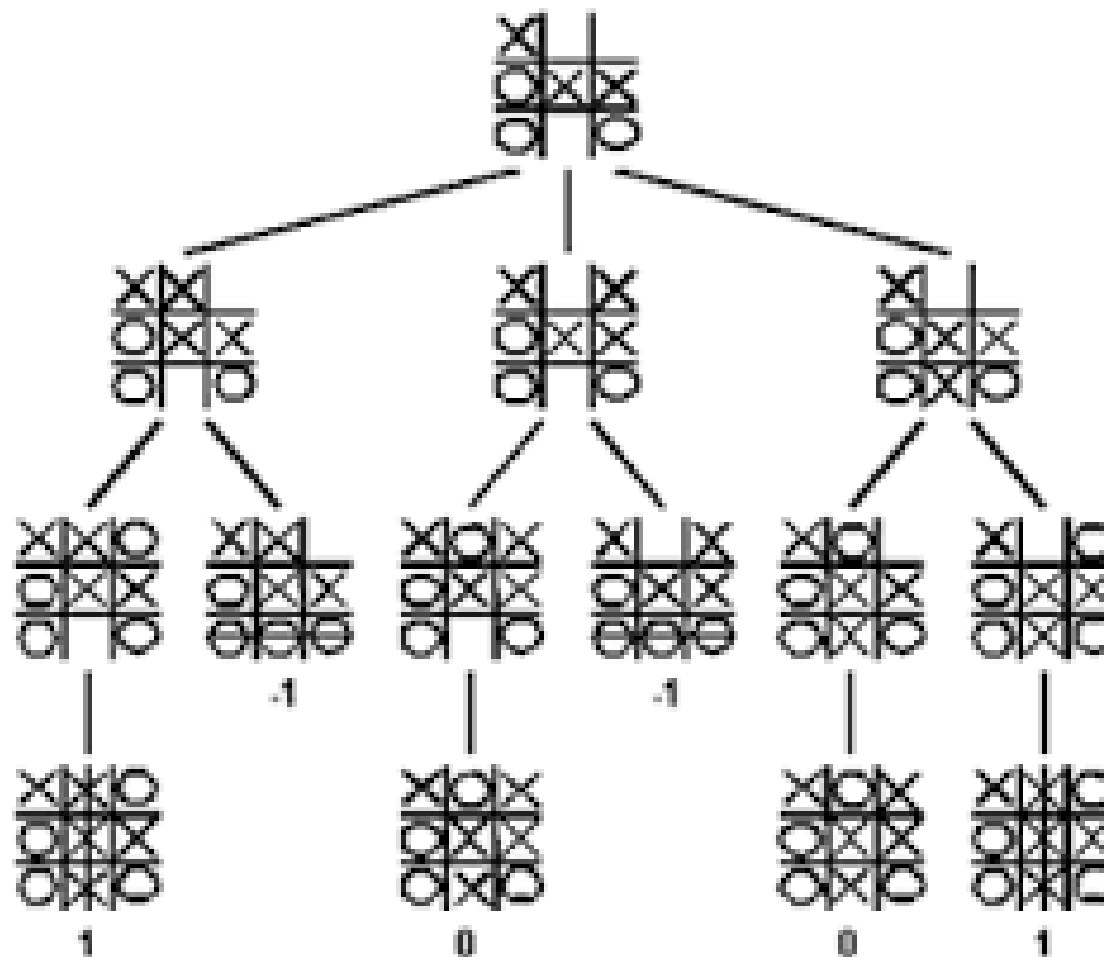
- Na mocy **twierdzenia o minimaksie** (J. von Neumann, 1928) istnieje taka unikalna funkcja wartości v_* , że:

$$v_*(s) = \max_{\pi_1} \min_{\pi_2} v_\pi(s)$$

- Strategię rozwiązującą powyższy problem nazywamy **strategią maksyminową**.
- Jest ona **równowagą Nasha** takiej gry.

Minimax

- Rozwiązanie możemy znaleźć poprzez przeszukiwanie w wszerz (breadth first search) lub w głąb (depth first search) drzewa gry.



Źródło: <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Game%20Trees/Game%20Trees.html>

Przeszukiwanie Minimax

Dla zadanego modelu (drzewa) \mathcal{M} o głębokości T . Dla każdego $t = T, T - 1, T - 2, \dots, 1$ i $k = \left\lfloor \frac{t}{2} \right\rfloor$:

- Gdy jesteś na poziomie nieparzystym (gracz 1 wybiera akcję):

- Wybierz a_{2k+1} takie, że:

$$a_{2k+1} = \operatorname{argmax}_{a \in A} V_{\pi}(s_{2k+1})$$

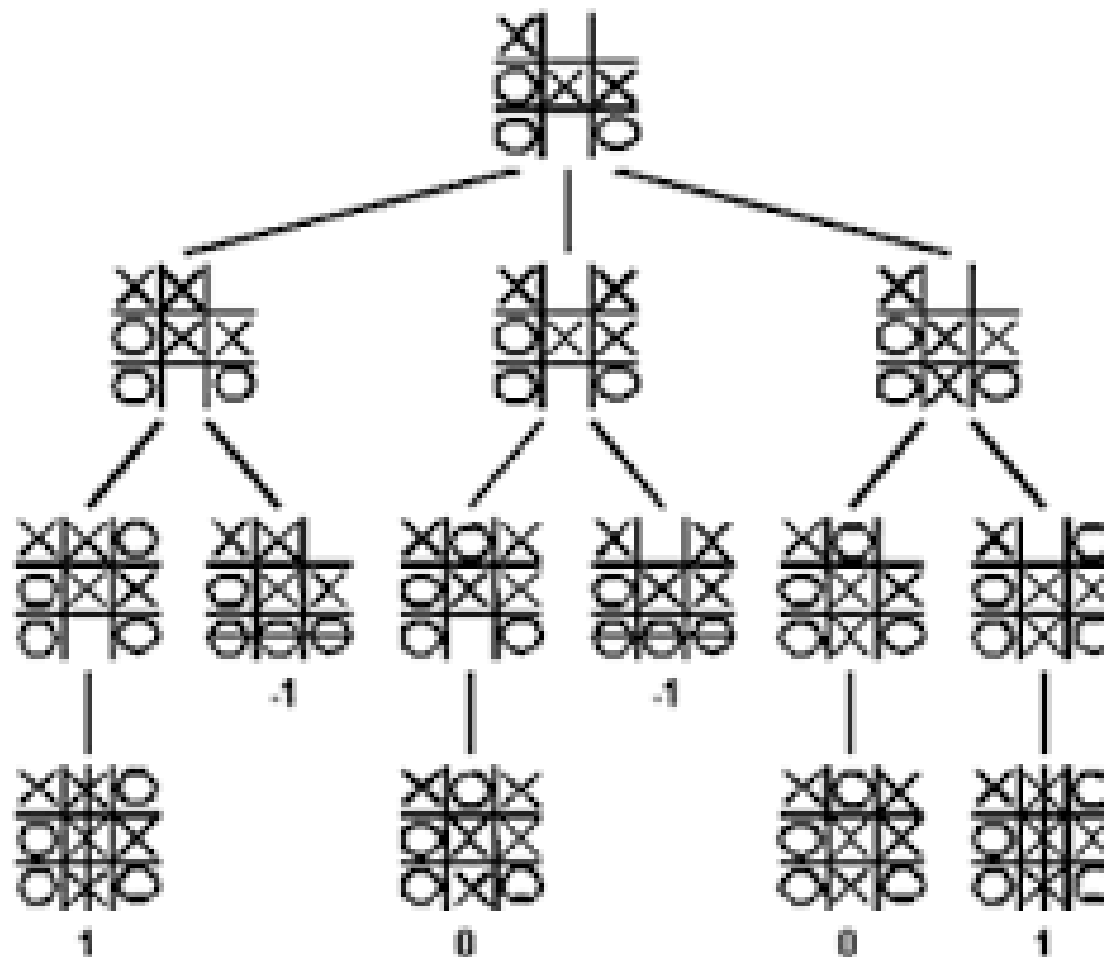
- Gdy jesteś na poziomie parzystym (gracz 2 wybiera akcję):

- Wybierz b_{2k} takie, że:

$$b_{2k} = \operatorname{argmin}_{b \in A} V_{\pi}(s_{2k})$$

Minimax

- Rozwiązanie możemy znaleźć poprzez przeszukiwanie w wszerz (breadth first search) lub w głąb (depth first search) drzewa gry.
- Ale złożoność drzewa rośnie wykładniczo.
- W przypadku dużych gier (np. szachy lub go) przeszukanie takiego drzewa jest niemożliwe.

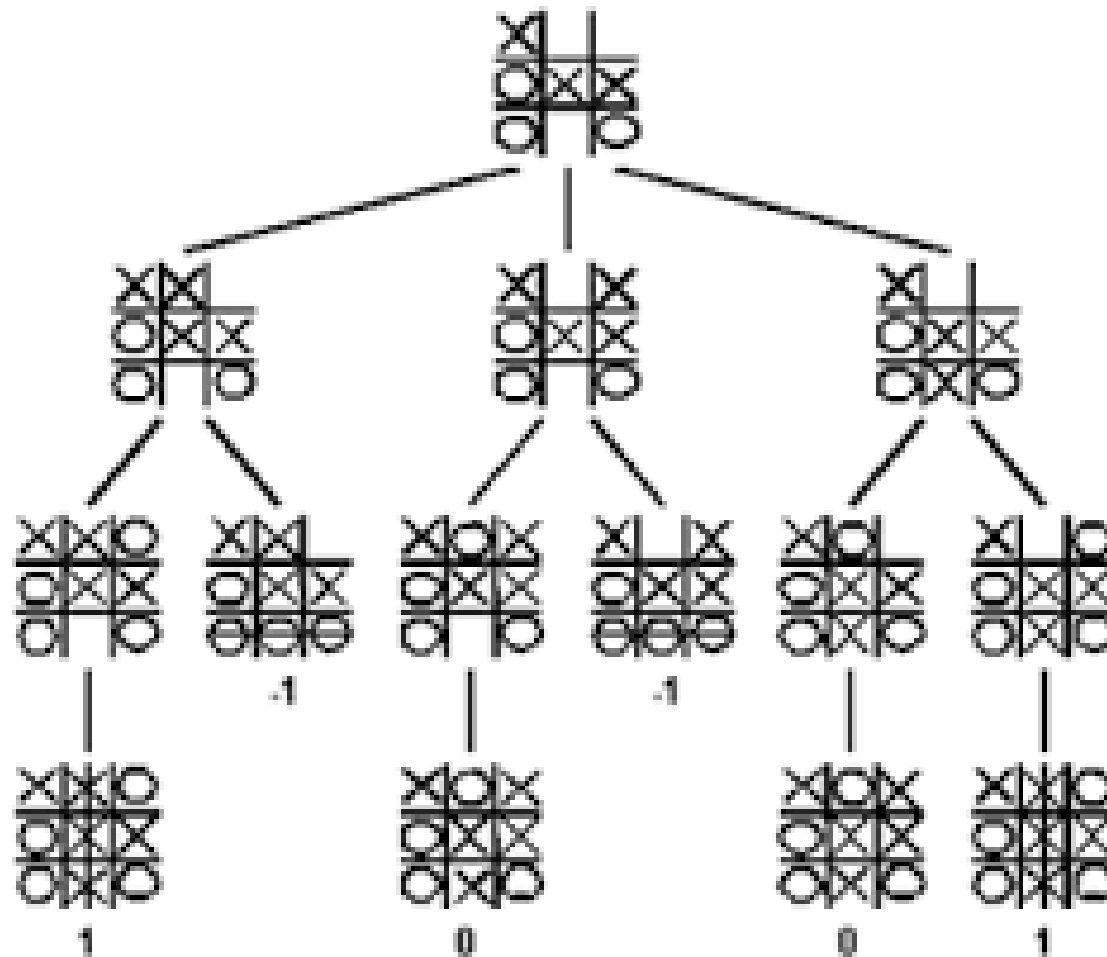


Źródło: <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Game%20Trees/Game%20Trees.html>

Minimax

- Tym co można zrobić jest przeszukiwanie do pewnego określonego poziomu a następnie aproksymacja wartości poddrzewa.
- Np. korzystając z aproksymacji funkcji wartości :

$$\hat{v}_{\pi}(s_t, \theta) \approx v_{\pi}(s_{t+1})$$



TD Learning

- Np. korzystając z aproksymacji funkcji wartości :

$$\hat{v}_{\pi}(s_t, \theta) \approx v_{\pi}(s_t)$$

- Przy takiej reprezentacji problemu możemy go rozwiązać korzystając z prostego algorytmu TD(0):

$$\Delta\theta = \eta \left(\hat{v}_{\pi}(s_{t+1}, \theta) - \hat{v}_{\pi}(s_t, \theta) \right) \nabla_{\theta} (\hat{v}_{\pi}(s_t, \theta))$$

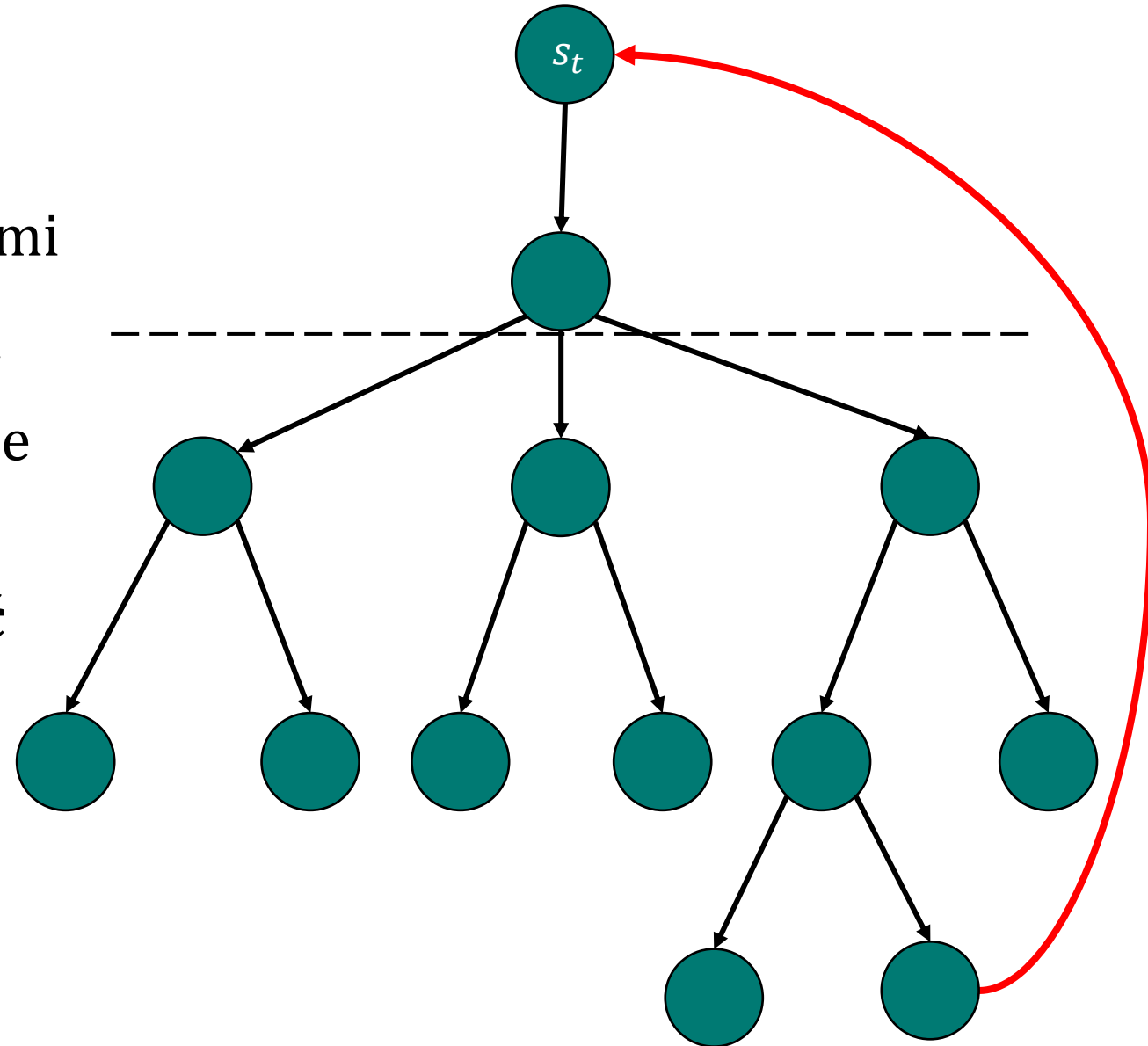
- Jak oszacować $\hat{v}_{\pi}(s_{t+1}, \theta)$?
 - Rozbudowując drzewo do poziomu liści i propagując wartość $v_{\pi}(s_{leaf})$ w tył.
 - Przyjmując reprezentację stanu $x(s_t)$, która np. mierzy *przewagę* gracza.

TD Learning

- Algorytm TD(0) jest w stanie poradzić sobie z relatywnie prostymi grami, ale przy bardziej złożonych grach, które wymagają planowania na wiele kroków w przód (np. szachy) nie jest w stanie efektywnie nauczyć się optymalnego rozwiązania.
- Zamiast tego możemy wykorzystać jeden z algorytmów, które efektywnie eksploatują drzewiastą strukturę problemu, np.:

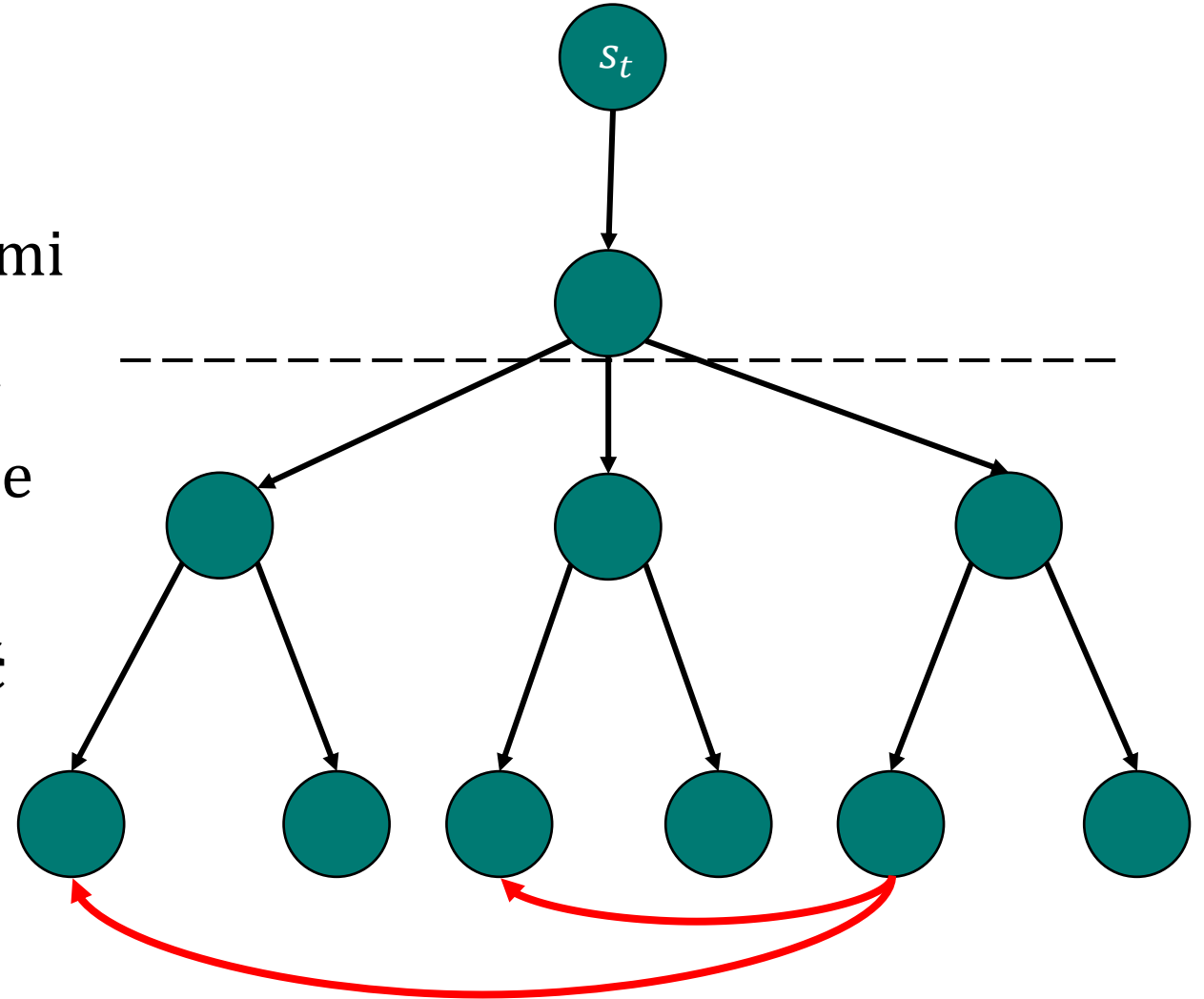
TD Learning

- Algorytm TD(0) jest w stanie poradzić sobie z relatywnie prostymi grami, ale przy bardziej złożonych grach, które wymagają planowania na wiele kroków w przód (np. szachy) nie jest w stanie efektywnie nauczyć się optymalnego rozwiązania.
- Zamiast tego możemy wykorzystać jeden z algorytmów, które efektywnie eksploatują drzewiastą strukturę problemu, np.:
 - **TD-Root** (Samuel 1959)



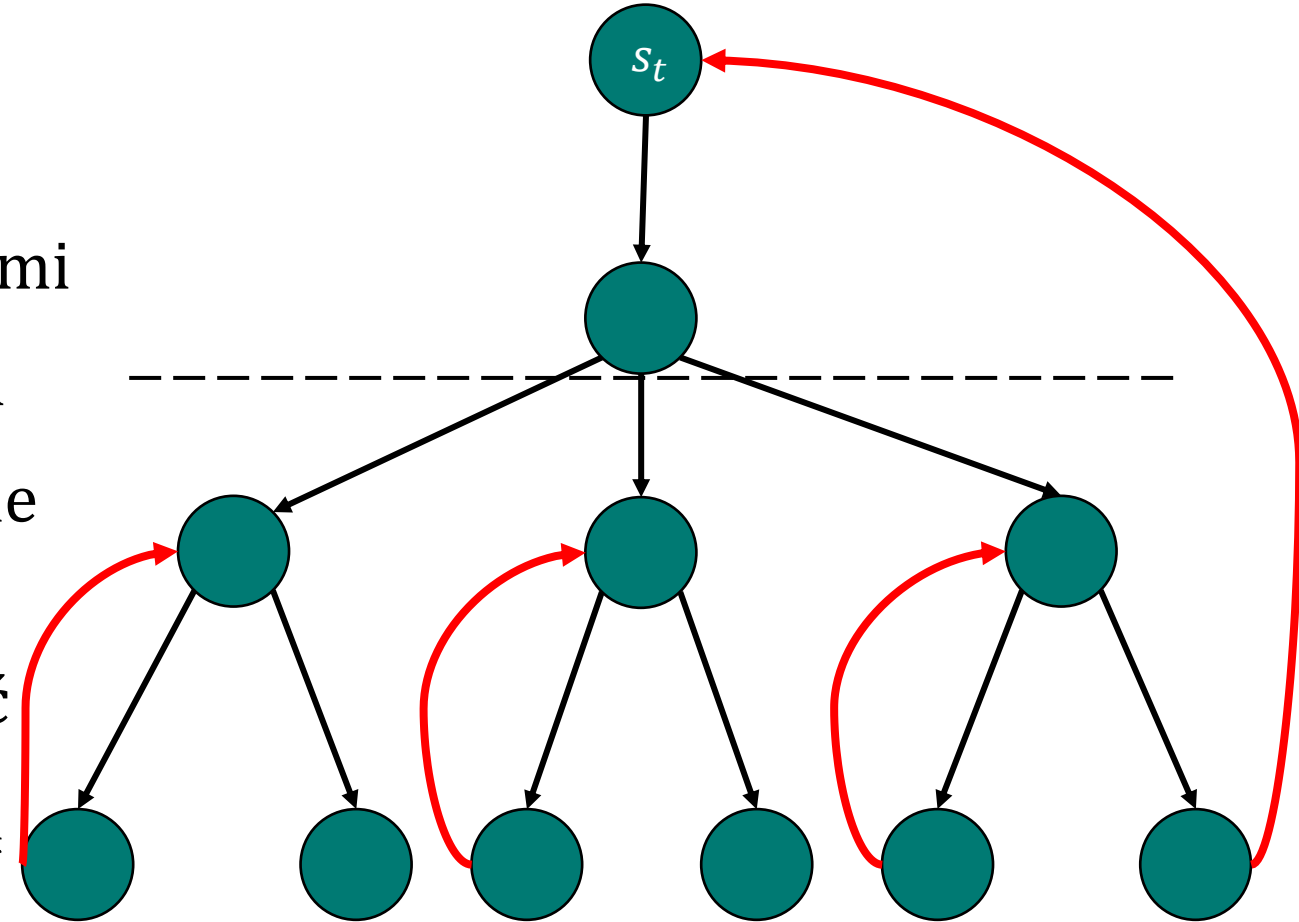
TD Learning

- Algorytm TD(0) jest w stanie poradzić sobie z relatywnie prostymi grami, ale przy bardziej złożonych grach, które wymagają planowania na wiele kroków w przód (np. szachy) nie jest w stanie efektywnie nauczyć się optymalnego rozwiązania.
- Zamiast tego możemy wykorzystać jeden z algorytmów, które efektywnie eksploatują drzewiastą strukturę problemu, np.:
 - TD-Root (Samuel 1959)
 - **TD-Leaf** (Baxter et al. 1998)



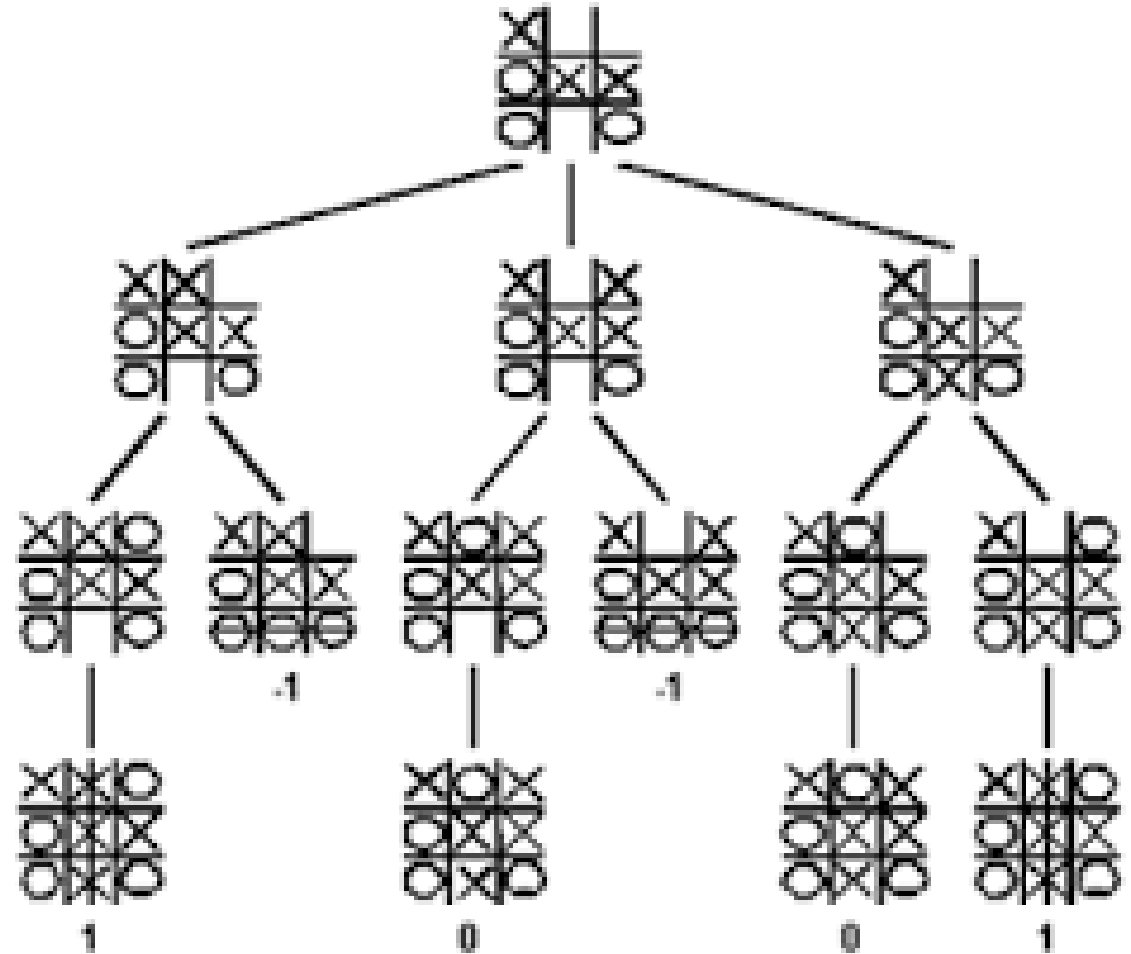
TD Learning

- Algorytm TD(0) jest w stanie poradzić sobie z relatywnie prostymi grami, ale przy bardziej złożonych grach, które wymagają planowania na wiele kroków w przód (np. szachy) nie jest w stanie efektywnie nauczyć się optymalnego rozwiązania.
- Zamiast tego możemy wykorzystać jeden z algorytmów, które efektywnie eksploatują drzewiastą strukturę problemu, np.:
 - TD-Root (Samuel 1959)
 - TD-Leaf (Baxter et al. 1998)
 - **TreeStrap** (Veness et al. 2009)



TD Learning

- Możemy nauczyć agenta optymalnej strategii poprzez granie z samym sobą.



Źródło: <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Game%20Trees/Game%20Trees.html>

Przeszukiwanie Monte Carlo

- Dla zadanego modelu \mathcal{M} i strategii przeszukiwania π :
- W aktualnym stanie s_t dla każdej akcji $a \in A$:
 - Wygeneruj K epizodów zaczynających się od aktualnego stanu s_t :

$$\{s_t, a, r_{t+1}^k, s_{t+1}^k, a_{t+1}^k, \dots, s_T^k\}_{k=1}^K \sim \mathcal{M}, \pi$$

- Uaktualnij wartość funkcji wartości $Q_\pi(s_t, a)$:

$$Q_\pi(s_t, a) += \frac{1}{K} \sum_{k=1}^K G_t$$

- Zachłannie wybierz kolejną (rzeczywistą) akcję:

$$a_t = \operatorname{argmax}_{a \in A} Q_\pi(s_t, a):$$

Monte-Carlo Tree Search

- Monte-Carlo Tree Search jest dużo efektywniejszym sposobem przeszukiwania drzewa niż bazowe przeszukiwanie Monte-Carlo.
- Dzięki niemu możemy znaleźć rozwiązanie dla faktycznie złożonych gier.

Monte-Carlo Tree Search

- Dla zadanego modelu \mathcal{M} i strategii przeszukiwania π :
- Korzystając ze strategii π , takiej że:

$$a^* = \operatorname{argmax}_{a \in A} (Q(s, a) + \sqrt{\frac{2 \ln k}{N_k(s, a)}})$$

wykonaj k ruchów zaczynając się od stanu początkowego s_0 :

$$(s_0, a_0, r_1, s_1, a_1, \dots, s_k) \sim \mathcal{M}, \pi$$

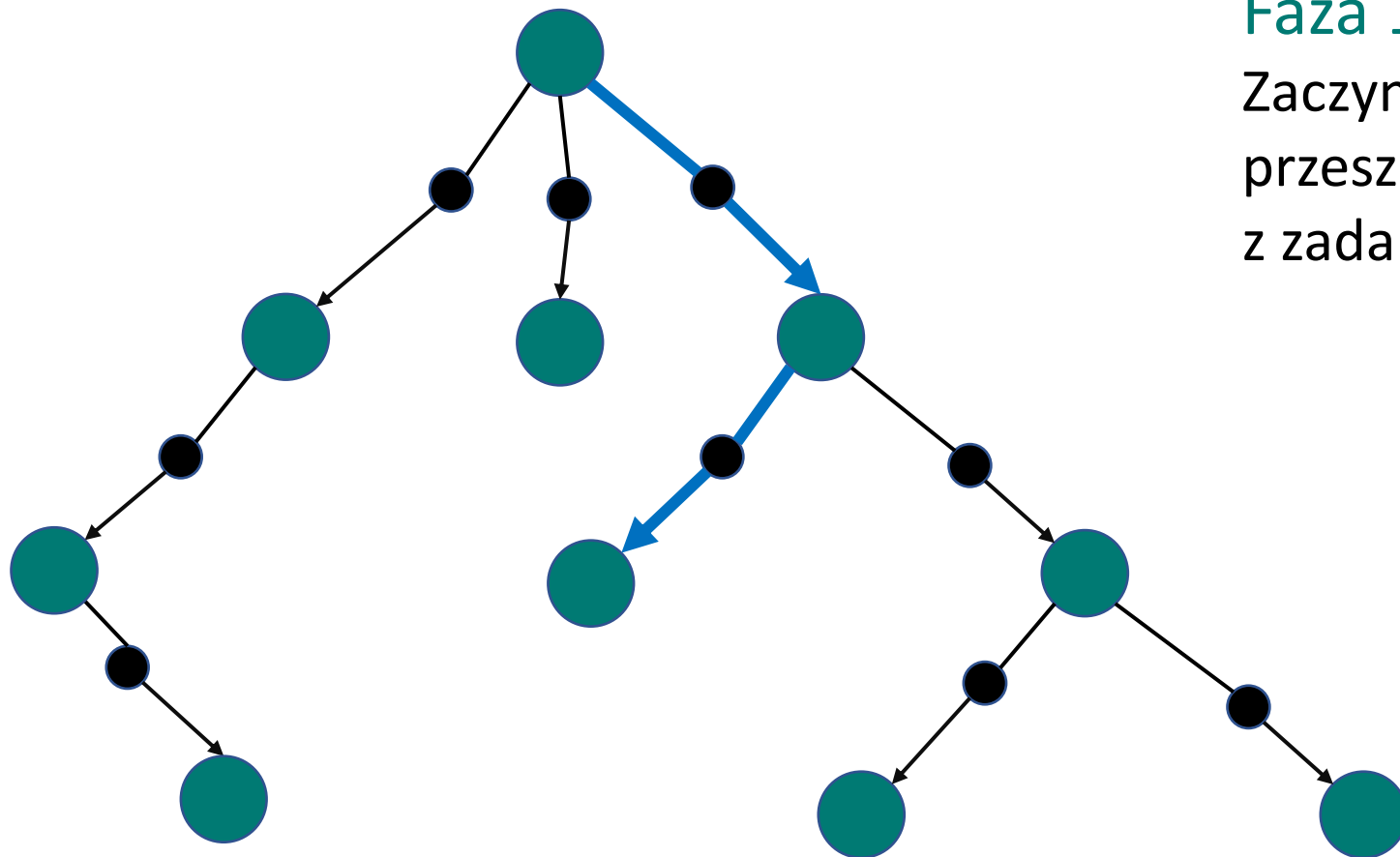
- (**Opcjonalnie**) Losowo wybierz kolejną akcję a_k , przejdź do stanu s_{k+1} .
- Wygeneruj resztę epizodu korzystając z losowej strategii przeszukiwania μ :

$$(s_{k+1}, a_{k+1}, r_{k+2}, s_{k+2}, a_{k+2}, \dots, s_T) \sim \mathcal{M}, \mu$$

- Uaktualnij wartość funkcji wartości $Q(s, a)$ dla każdego s, a w drzewie \mathcal{M} :

$$Q(s, a) += \frac{1}{N(s, a)} \sum_{k=1}^K \sum_{i=t}^T I(s_i, a_i = s, a) G_i$$

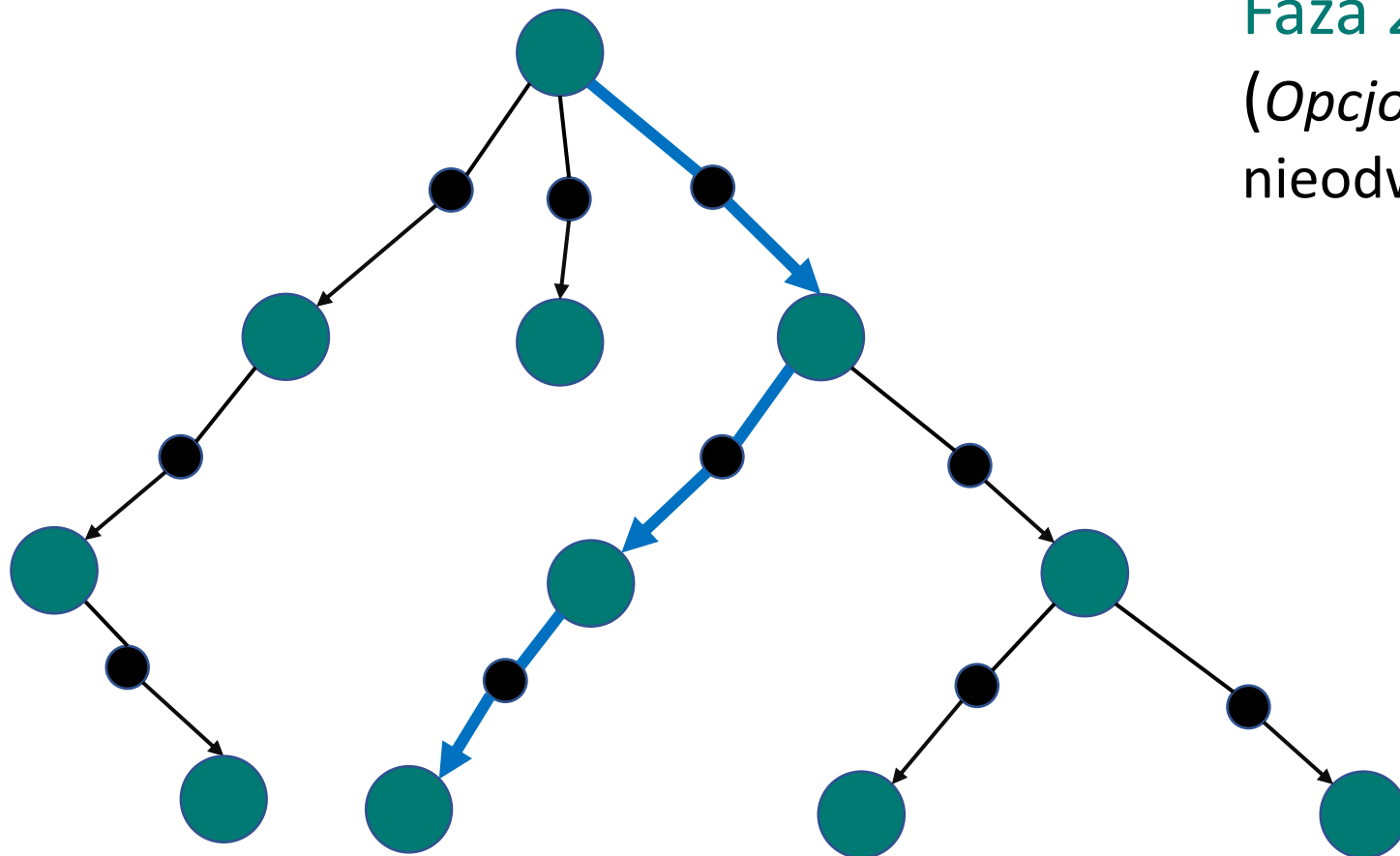
Monte-Carlo Tree Search



Faza 1: Wybór

Zaczynając w korzeniu drzewa
przeszukaj je w głąb na k korzystając
z zadanej strategii przeszukiwania π .

Monte-Carlo Tree Search



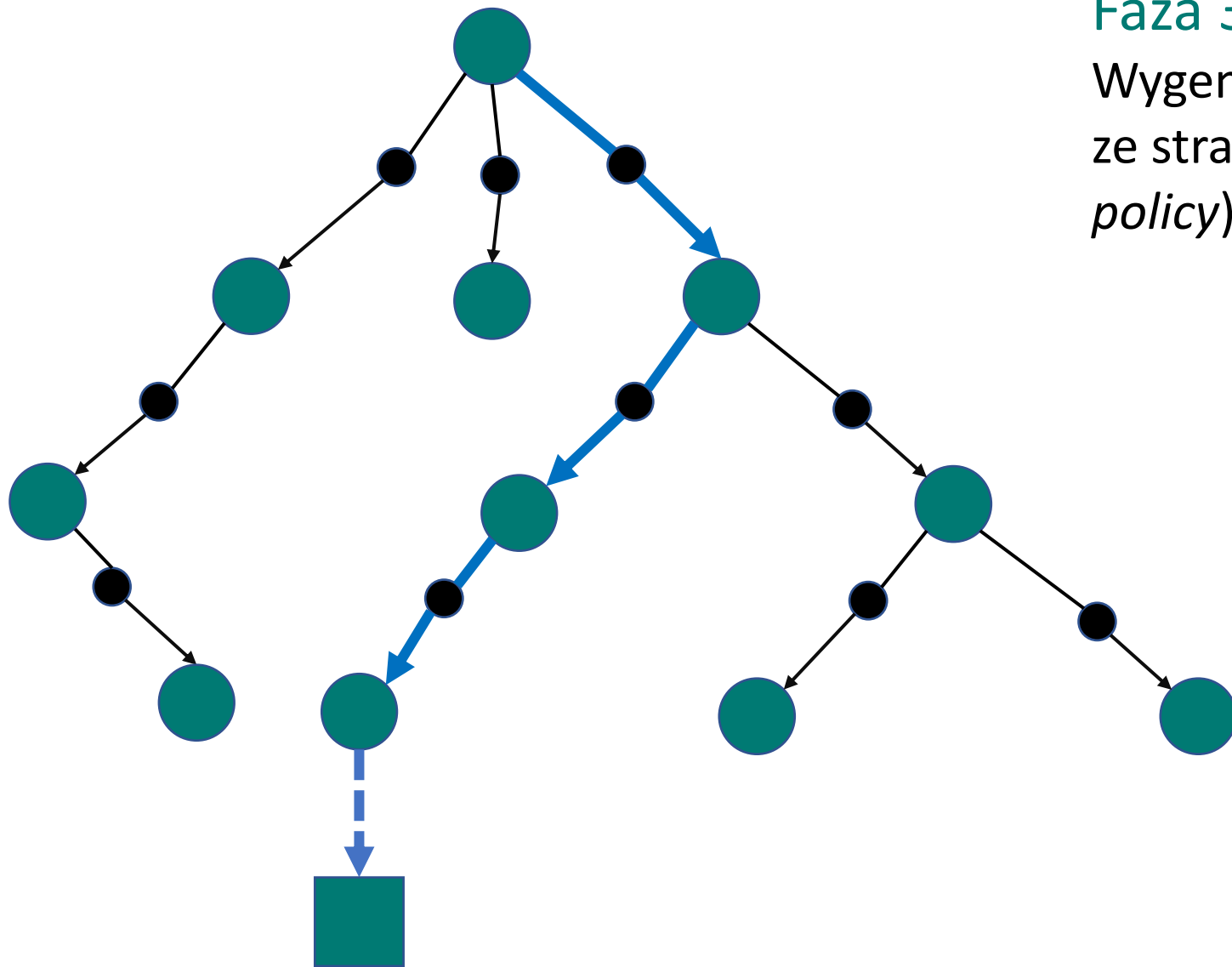
Faza 2: Rozrost

(Opcjonalnie) Wybierz losowo nieodwiedzony jeszcze wierzchołek.

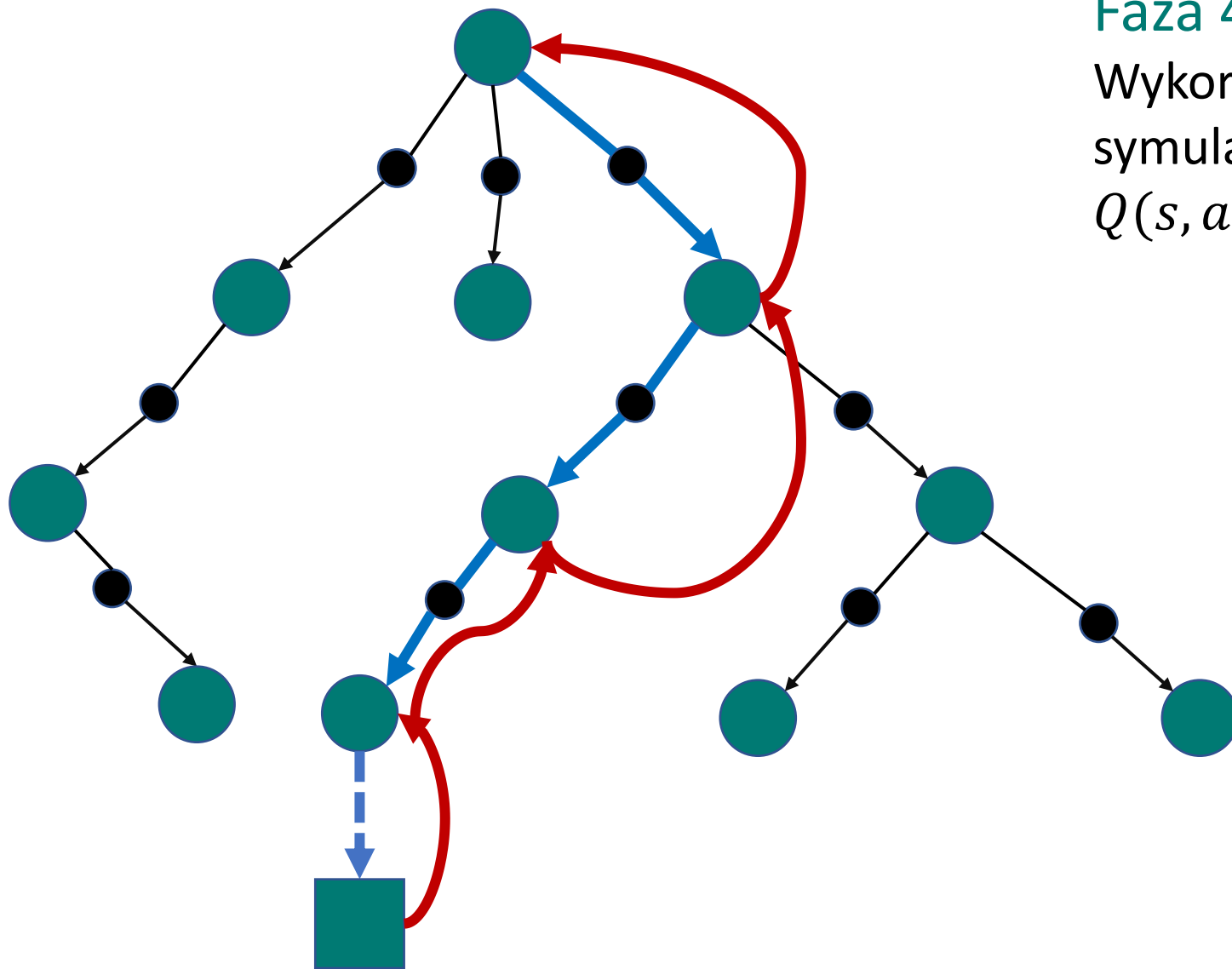
Monte-Carlo Tree Search

Faza 3: Symulacja

Wygeneruj resztę epizodu korzystając ze strategii przeszukiwania (*rollout policy*) μ :



Monte-Carlo Tree Search



Faza 4: Propagacja wsteczna

Wykorzystując otrzymany wynik symulacji uaktualnij wartości funkcji $Q(s, a)$.

Monte-Carlo Tree Search

- Algorytm zbiega do optymalnego sposobu przeszukiwania drzewa:

$$Q(S, A) \rightarrow q_*(S, A)$$

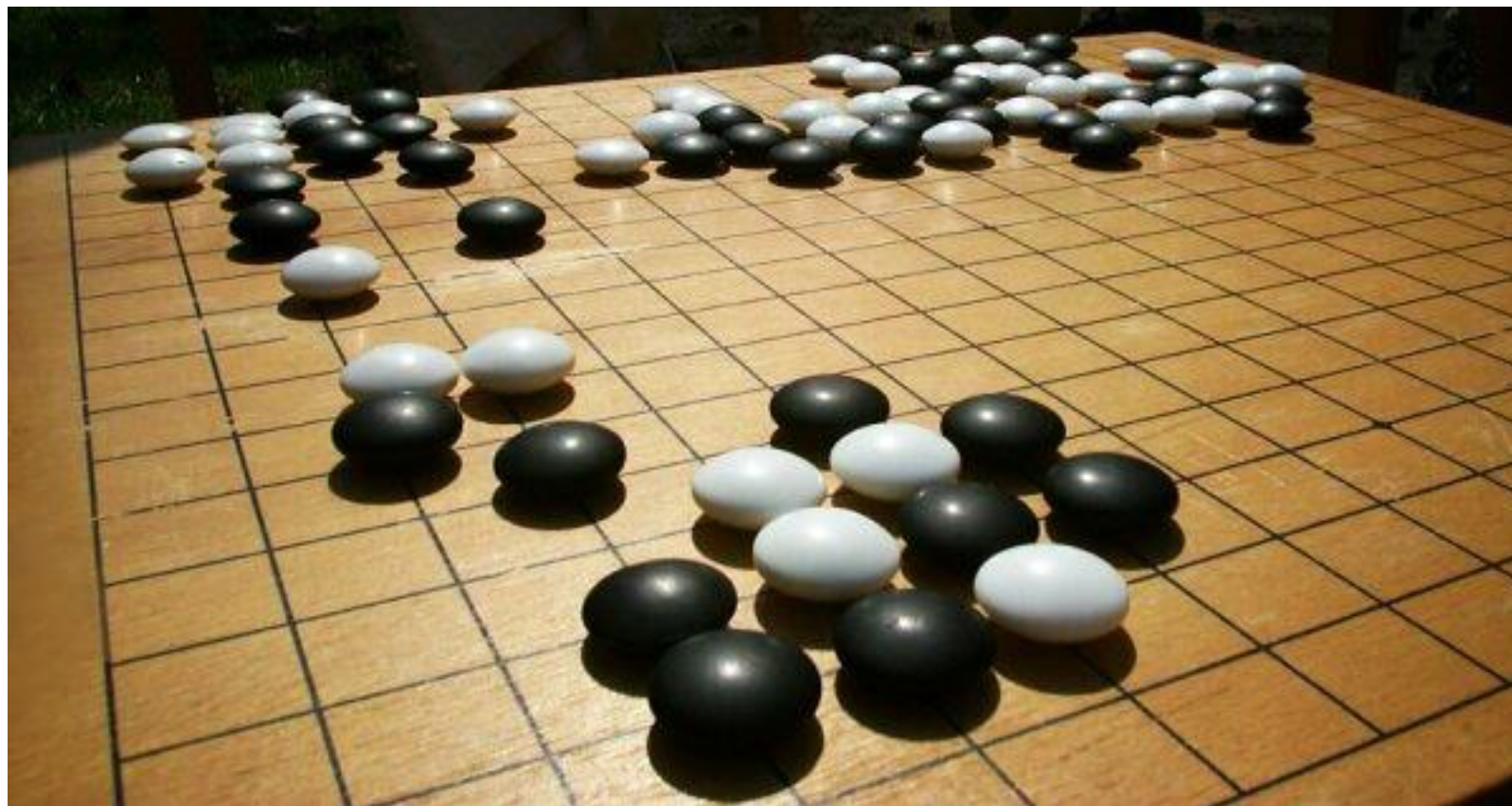
Uczenie ze wzmocnieniem w klasycznych grach: AlphaGo

Literatura dodatkowa

- D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis.
“Mastering the game of Go with deep neural networks and tree search.” Nature 529, pp. 484-489. Jan. 2016.
<https://www.nature.com/articles/nature16961>
<https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf>
- D. Silver. “AlphaGo” (Presentation).
- https://www.davidsilver.uk/wp-content/uploads/2020/03/AlphaGo-tutorial-slides_compressed.pdf

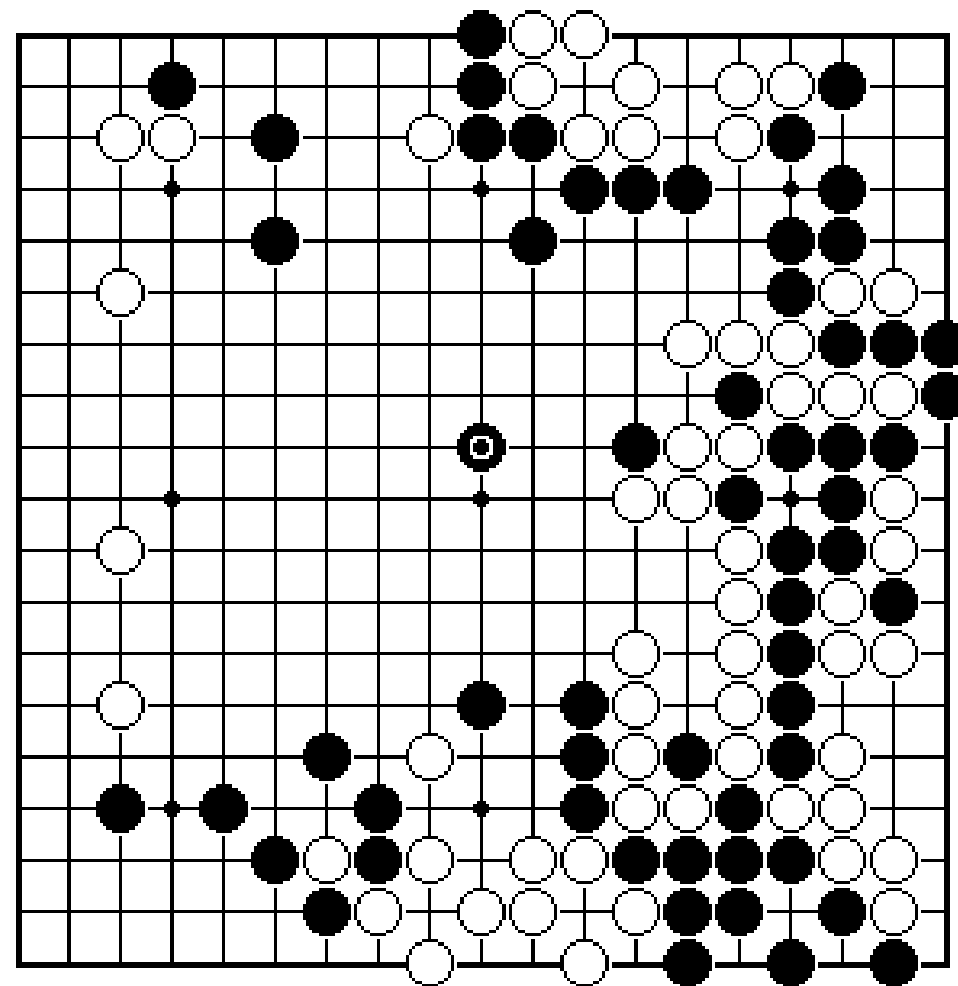
Go - podstawy

- Klasyczna gra, istniejąca od ok. 2 500 lat.
- Gracze kładą na przemian czarne i białe kamienie na przecięciu linii.
- Grę rozpoczyna gracz grający kamieniami czarnymi.
- Celem gry jest otoczenie własnymi kamieniami, na pustej początkowo planszy, terytorium większego niż terytorium przeciwnika.
- Kamieni raz postawionych na planszy nie przesuwa się, mogą natomiast zostać zbite przez przeciwnika.



Go - podstawy

- Gra o doskonałej informacji.
- Z racji swoich rozmiarów i prostoty zasad stanowi wielkie wyzwanie zarówno dla graczy, jak i algorytmów AI.



AlphaGo

- Nagroda:

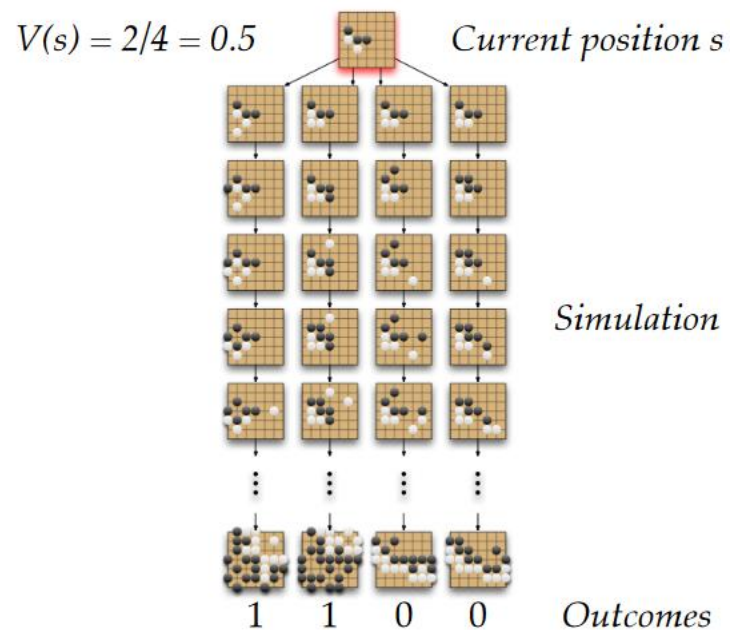
$$R_t = 0 \text{ dla stanów nieterminalnych } t$$
$$R_T = \begin{cases} 1 & \text{gdy wygrywają czarne} \\ 0 & \text{gdy wygrywają białe} \end{cases}$$

- Strategia $\pi = (\pi_B, \pi_W)$ kontroluje zachowanie graczy.
- Optymalna wartość funkcji wartości dla stanu s :

$$v_*(s) = \max_{\pi_B} \min_{\pi_W} v_\pi(s)$$

AlphaGo

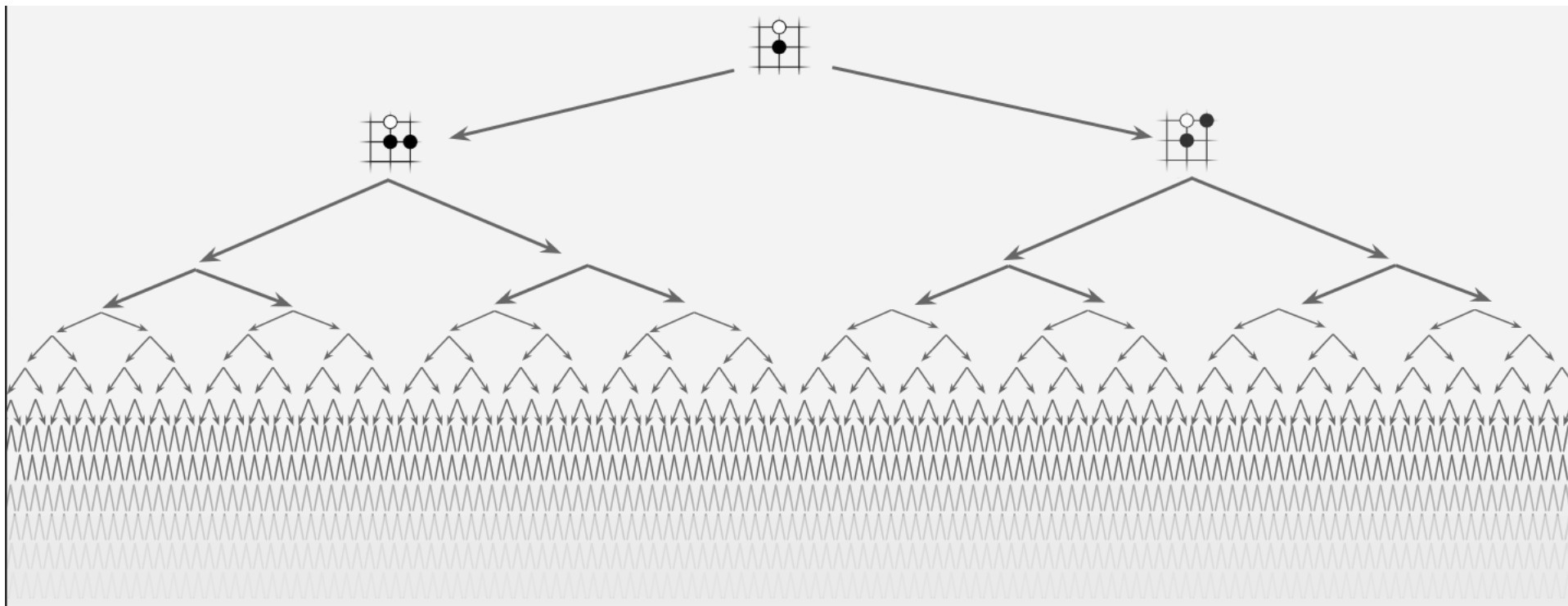
- Drzewo gry w Go:



Źródło: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/dyna.pdf

AlphaGo

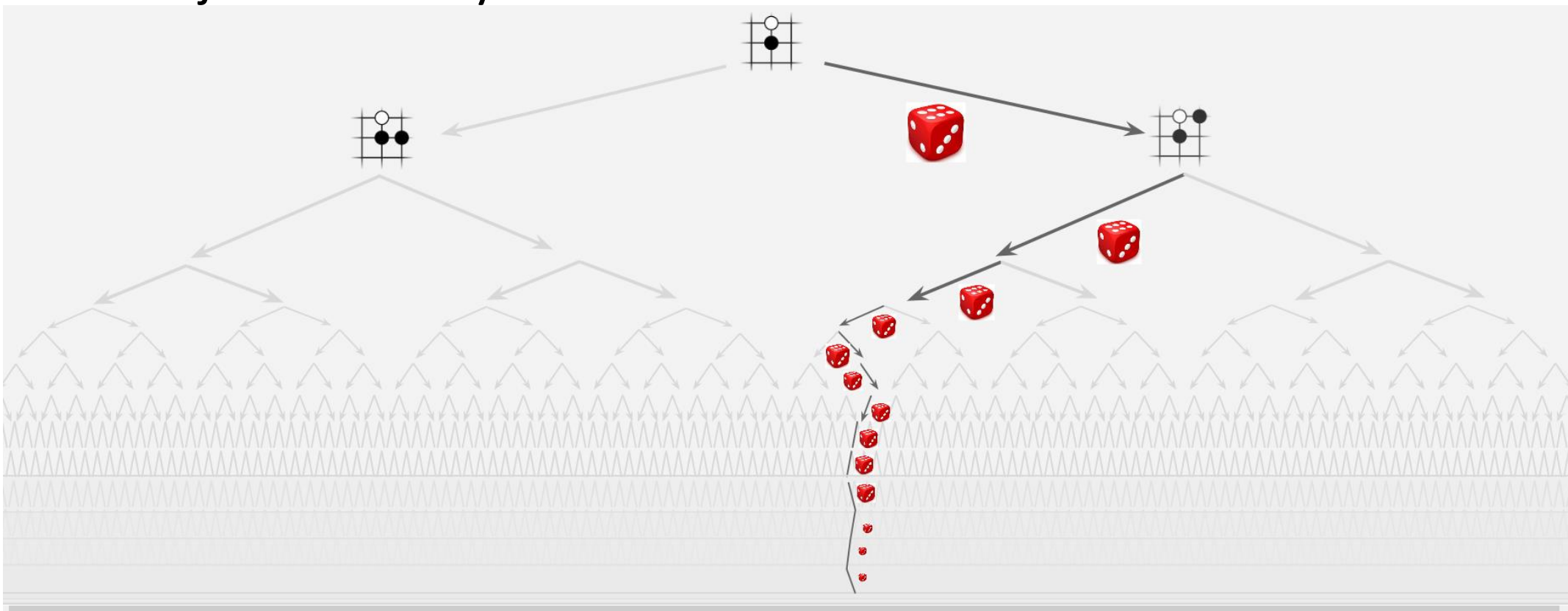
- W rzeczywistości jest ono jednak wyraźnie większe:



http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/AlphaGo_IJCAI.pdf

AlphaGo

- Na tyle duże, że nawet korzystanie z algorytmu Monte-Carlo Tree Search jest nieefektywne:

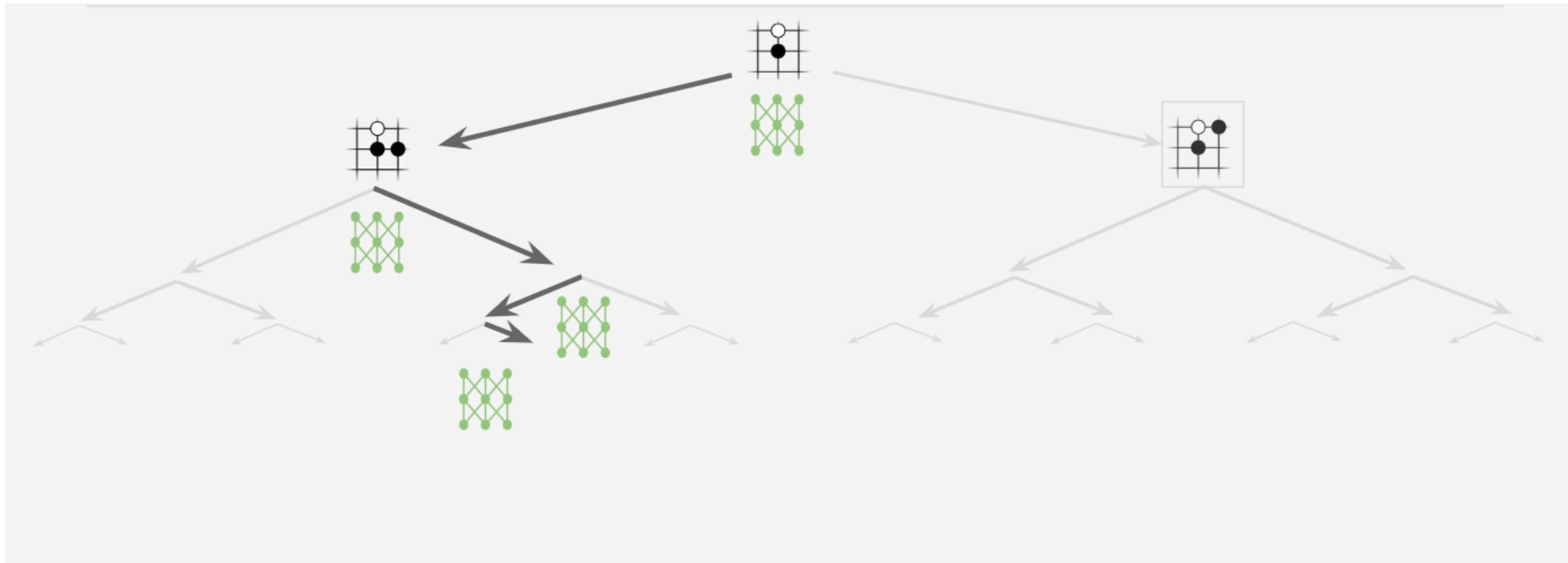


AlphaGo

- Jak rozwiązano ten problem?
- Wykorzystano sieci neuronowe w celu aproksymacji funkcji wartości i aproksymacji funkcji strategii, dzięki czemu udało się zmniejszyć rozmiar przeszukiwanego drzewa zarówno wzdłuż jak i wszerz.

AlphaGo

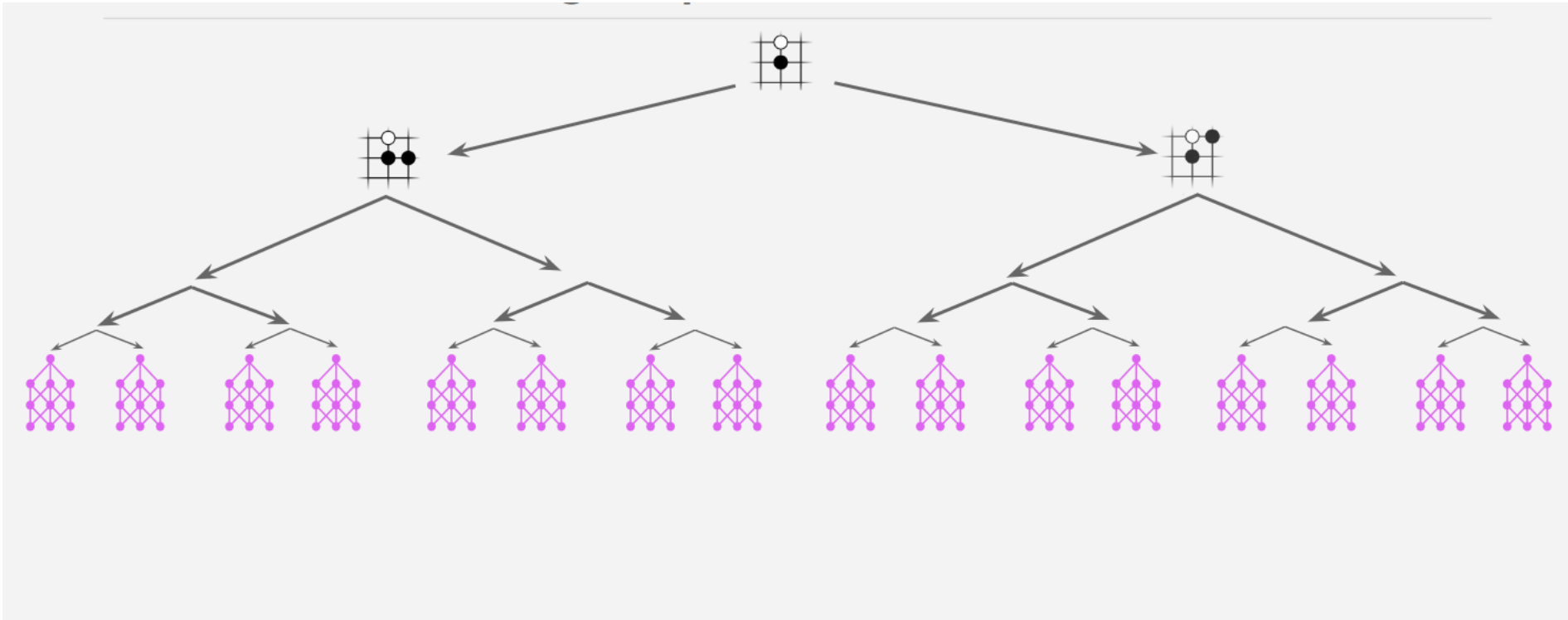
- Dzięki wykorzystaniu sieci neuronowej do wyboru optymalnej akcji udało się zmniejszyć rozmiar drzewa wszerek:



http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/AlphaGo_IJCAI.pdf

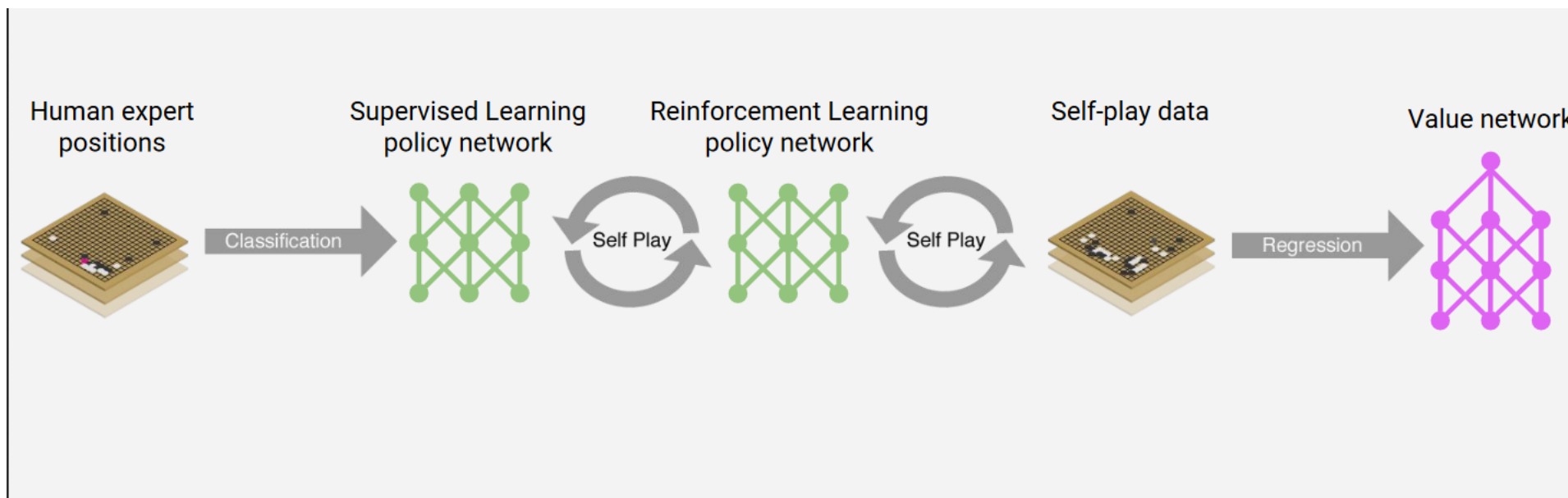
AlphaGo

- A dzięki aproksymacji funkcji wartości udało się zmniejszyć rozmiar drzewa wzdłuż:



AlphaGo

- Proces uczenia był dwustopniowy:



http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/AlphaGo_IJCAI.pdf

AlphaGo

- Proces uczenia był dwustopniowy:
- W pierwszym kroku sieć aproksymująca strategię była uczona na wybranych 30 milionach pozycji z partii rozgrywanych przez arcymistrzów go.
- Na jej podstawie stworzono dwie sieci:
 - **Fast policy network** (p_{π})
 - **Strong policy network** (p_{σ})

AlphaGo

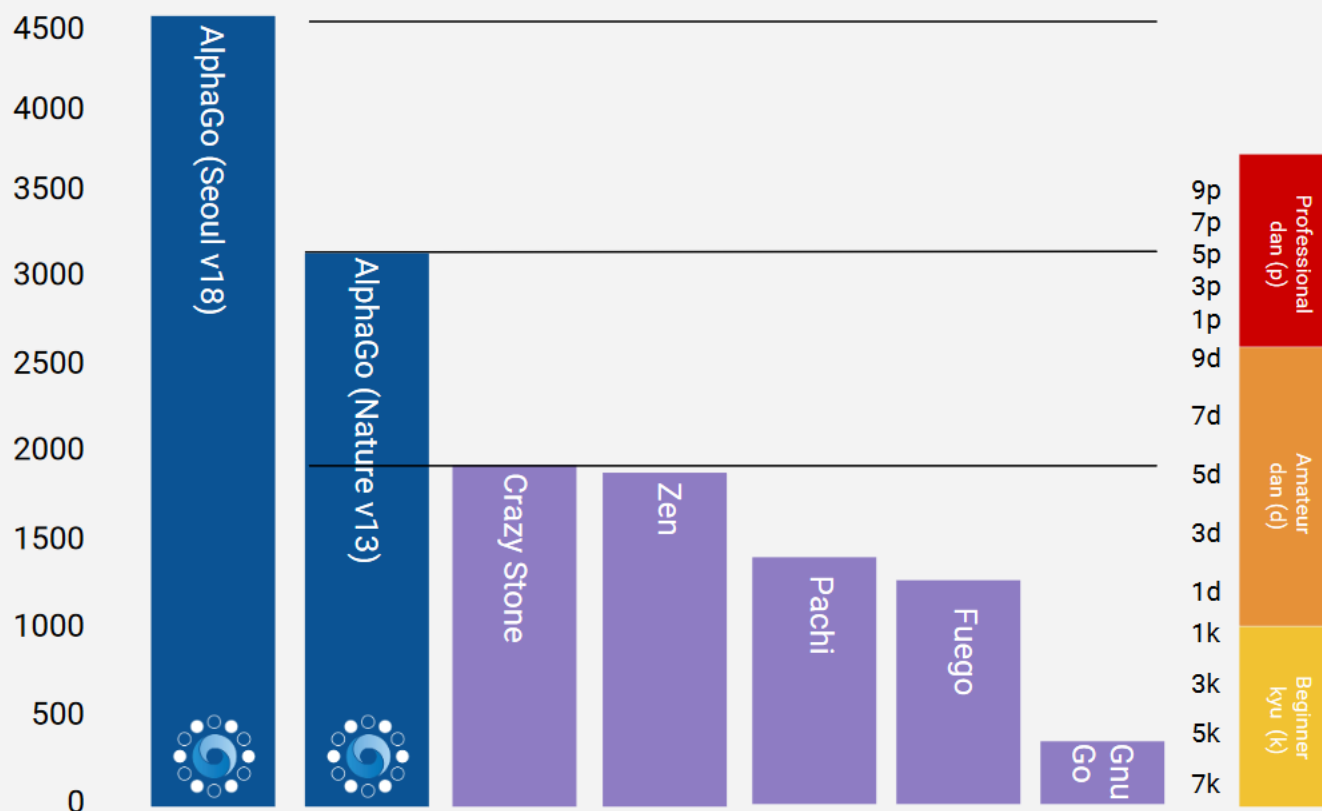
- Proces uczenia był dwustopniowy:
- Model grał sam ze sobą, w międzyczasie dalej trenując *strong policy network* (p_σ) w celu maksymalizacji prawdopodobieństwa zwycięstwa.
- Dodatkowo w tym samym czasie uczona była sieć aproksymująca funkcję wartości $v_\theta(s)$.

AlphaGo

Evaluating Seoul AlphaGo against computers

>50% against
Nature AlphaGo
with 3 to 4 stone
handicap

CAUTION: ratings
based on self-
play results



AlphaGo vs AlphaGo Zero

- AlphaGo Zero:
 - uczy się grać w go od podstaw, nie bazując na przykładach żywych graczy.
 - Zaczyna od pustej planszy i uczy się jedynie grając samemu ze sobą.
 - Ewaluacja Monte-Carlo Tree Search opiera się w pełni na aproksymowanej funkcji wartości.
 - Zamiast sieci konwolucyjnych wykorzystuje architekturę ResNet.

Przykład

- Kod źródłowy:

<https://github.com/tejank10/AlphaGo.jl>

