

---

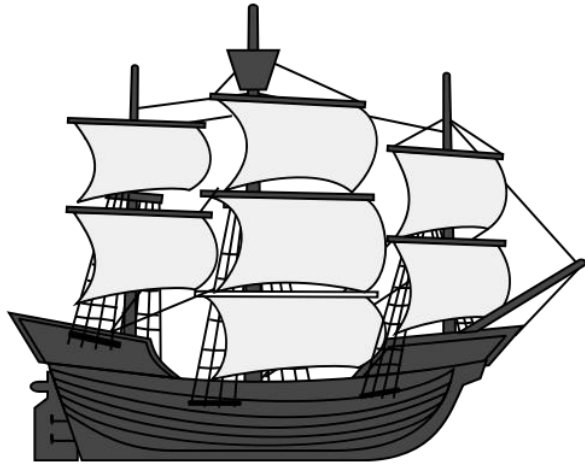
# Wykład 9. Monte Carlo Tree Search

Piotr Morawiecki  
12 maja 2025

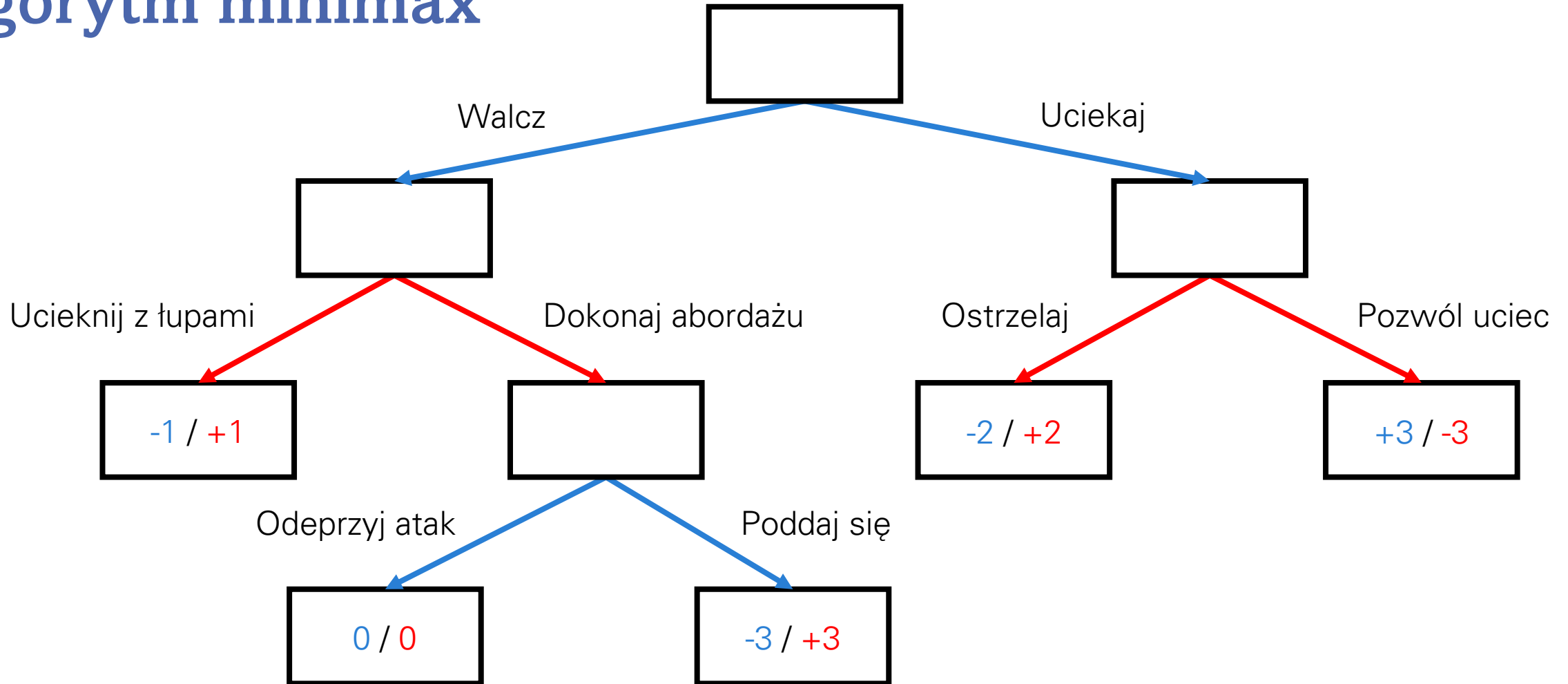


---

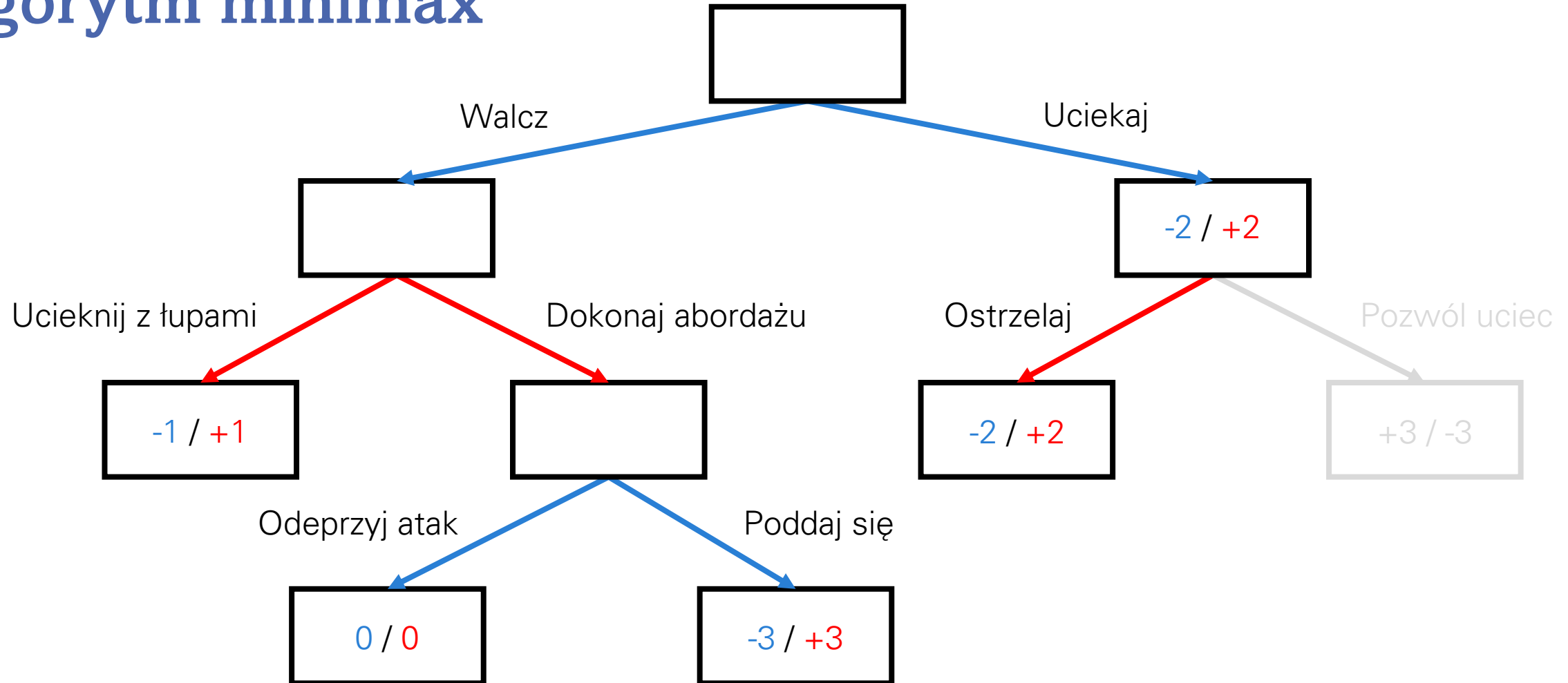
# Powtórka: algorytm minimax



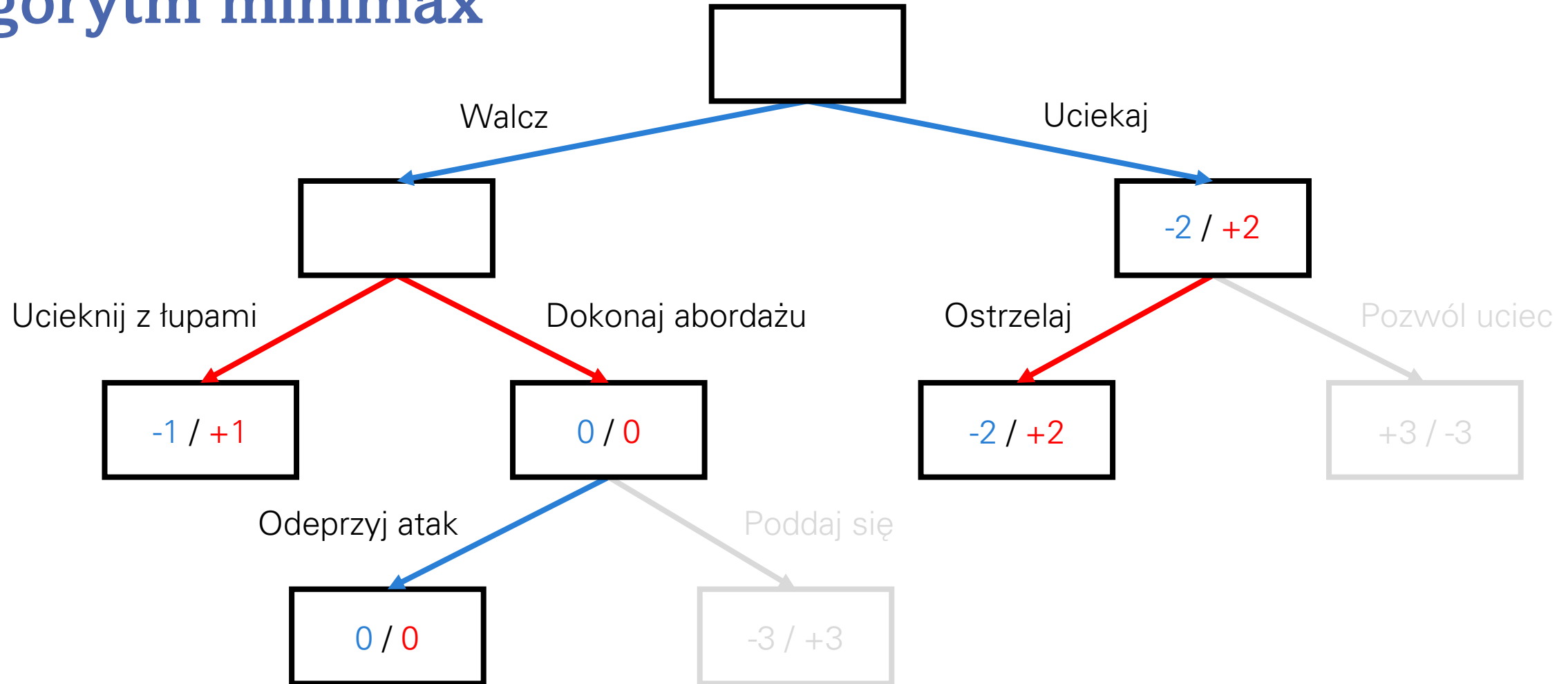
# Algorytm minimax



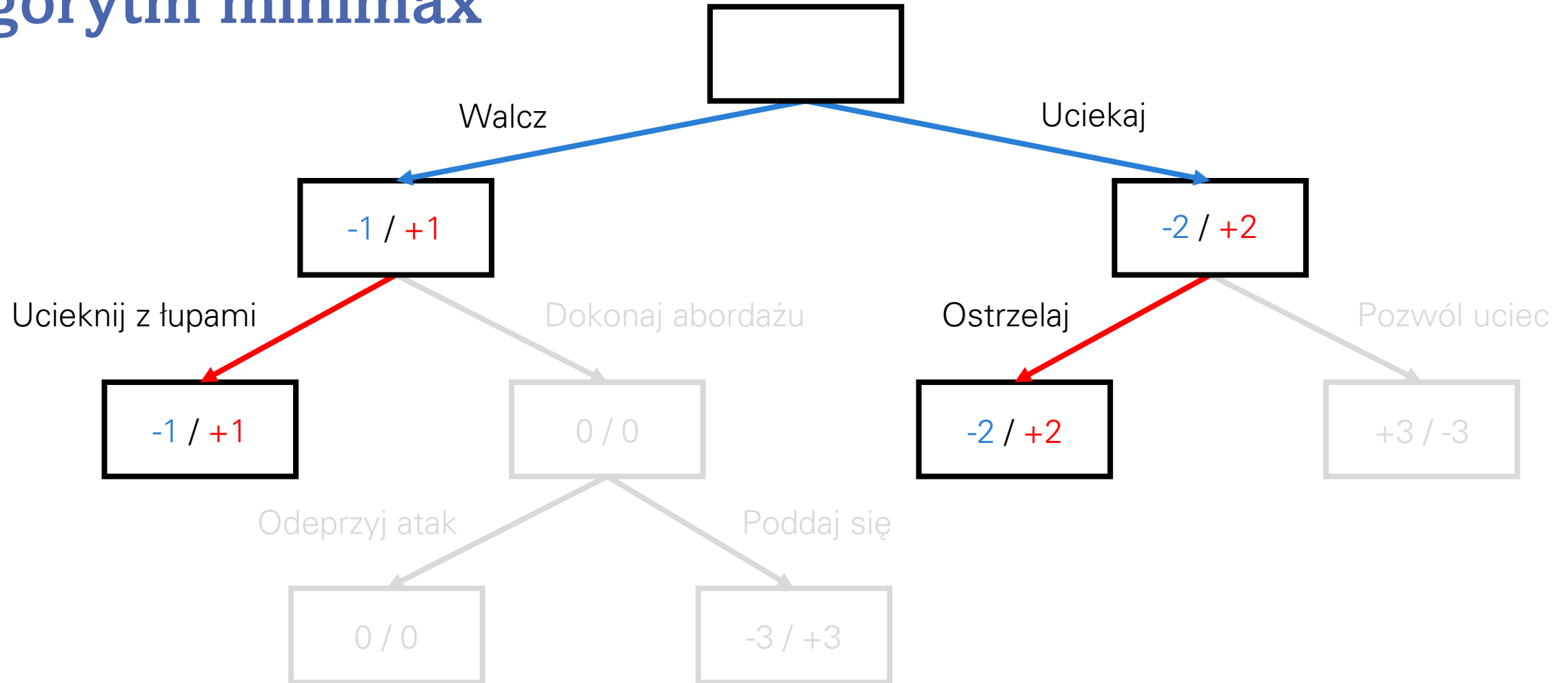
# Algorytm minimax



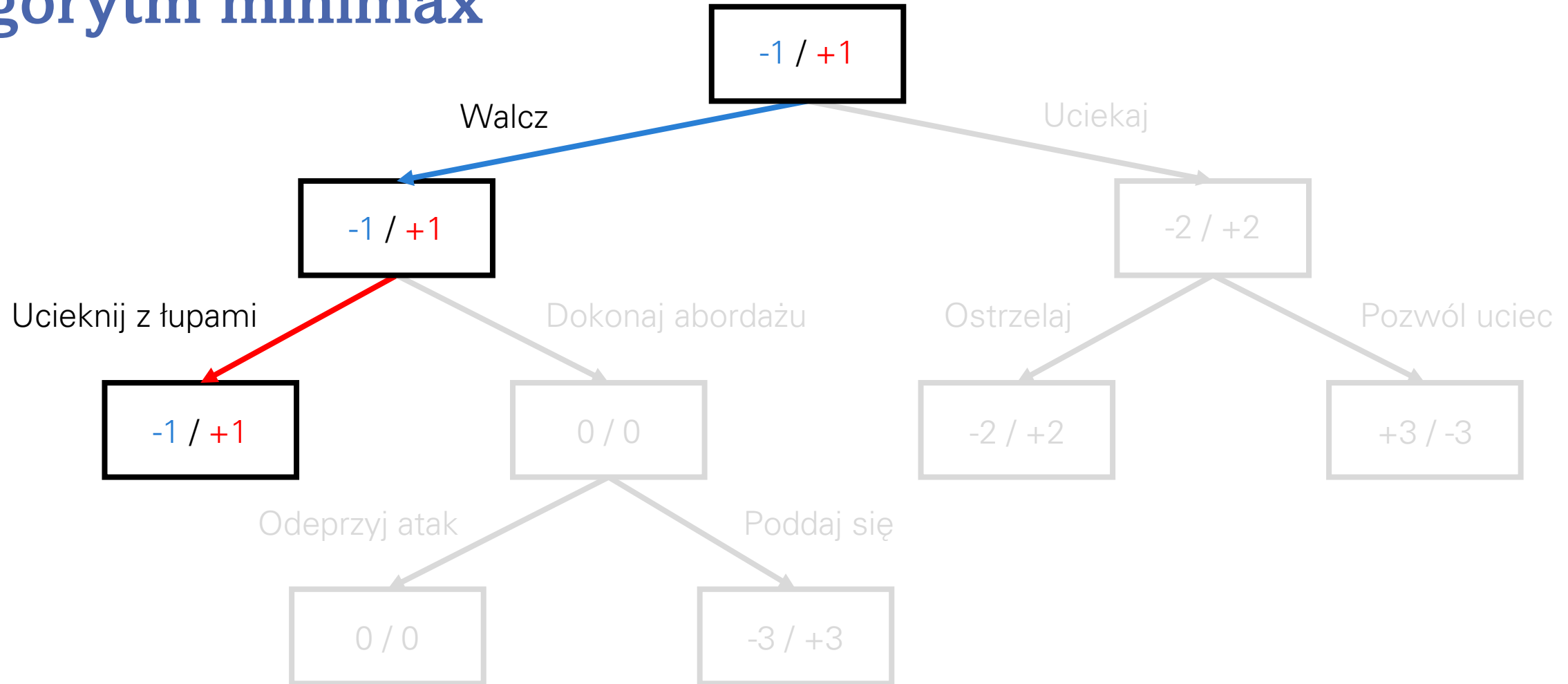
# Algorytm minimax



# Algorytm minimax

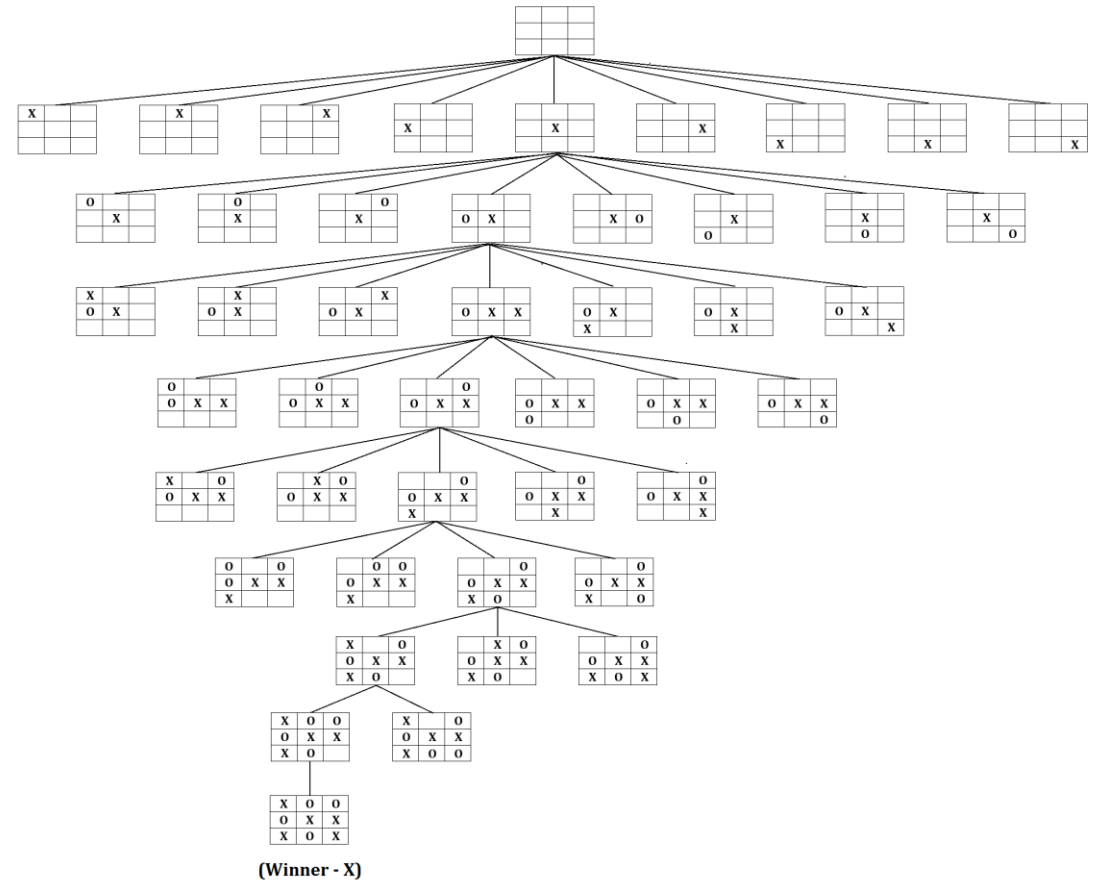


# Algorytm minimax



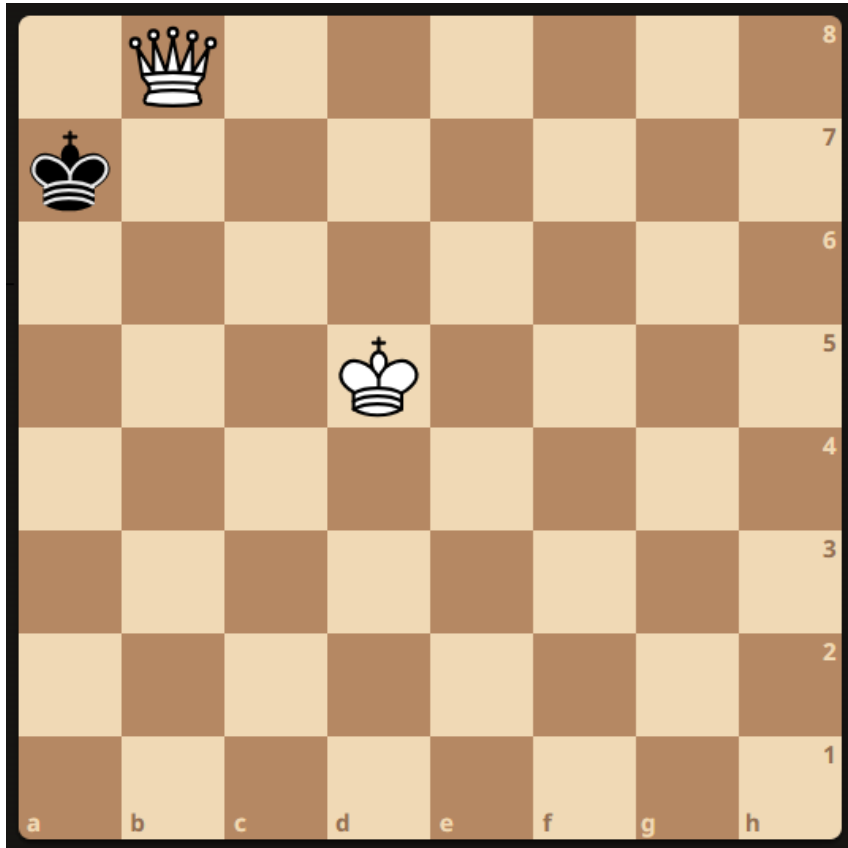
# Główny problem: liczba możliwych scenariuszy

- Głównym problemem jest liczba możliwych scenariuszy, np. grę w kółko i krzyżyk można rozegrać na 255,168 sposobów!
- W szachach można wykonać na 169,518,829,100,544,000,000,000,000,000 sposobów wykonać pierwsze 10 ruchów!
- Możliwych konfiguracji planszy w szachach jest od  $10^{111}$  and  $10^{123}$  (więcej niż atomów we wszechświecie)



---

# Jak my sobie z tym radzimy? (1/2)

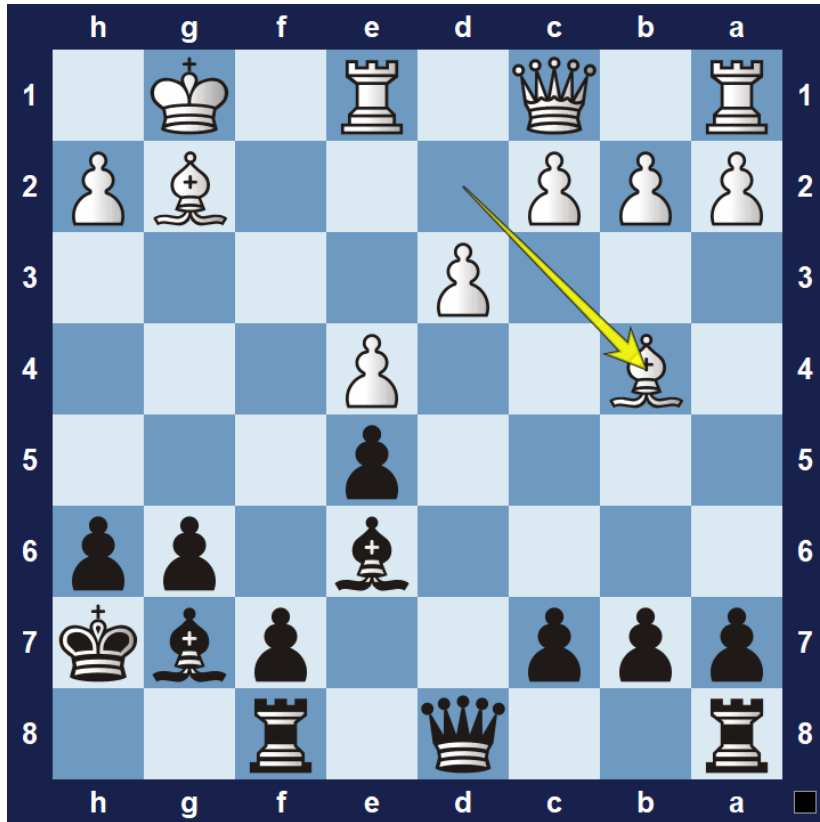


Teraz jest tura białego gracza.

Co powinien zrobić?

---

# Jak my sobie z tym radzimy? (2/2)

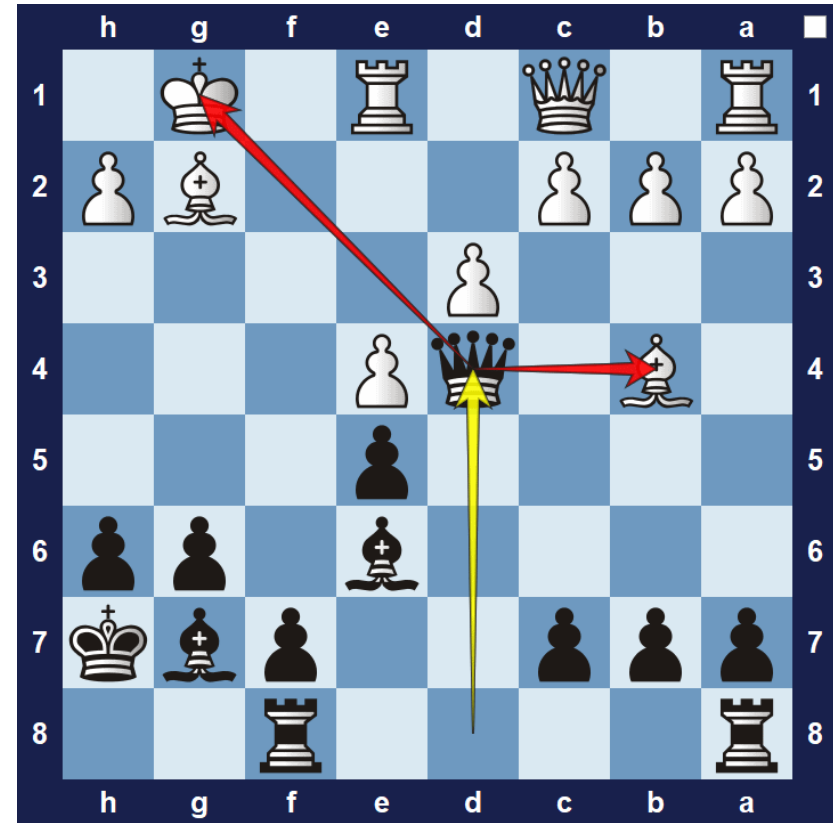
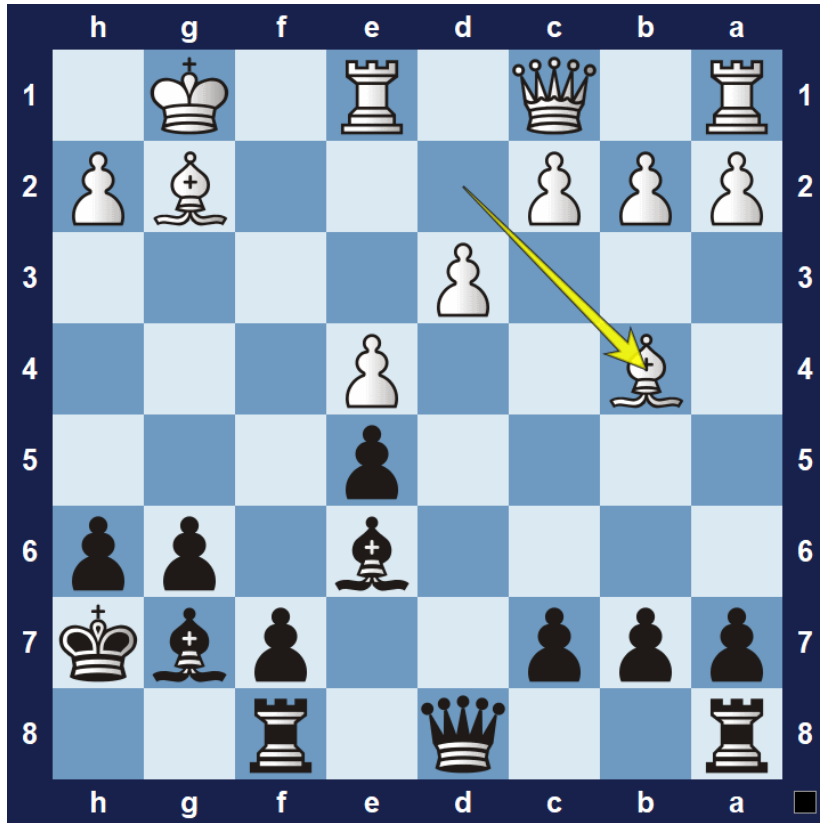


Biały gracz popełnił błąd ruszając się gońcem na pole B4.

Jak czarny gracz może zyskać przewagę?

---

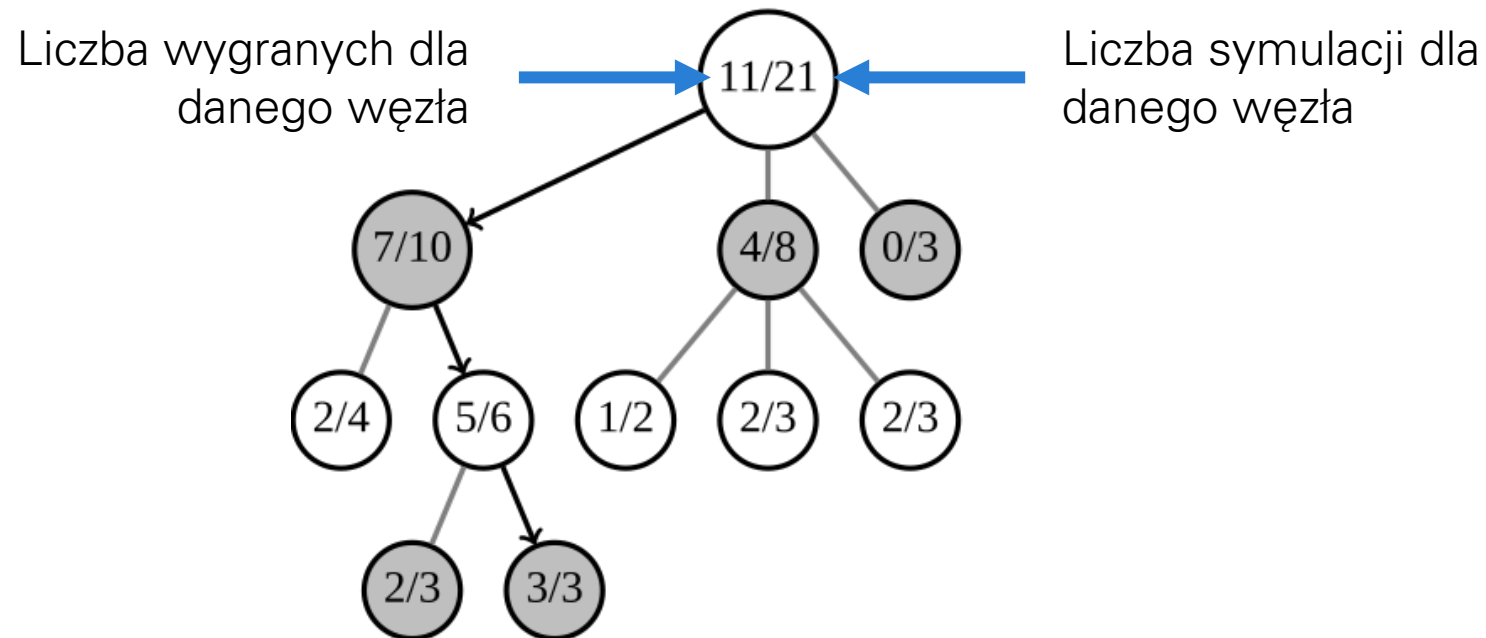
# Jak my sobie z tym radzimy? (2/2)



---

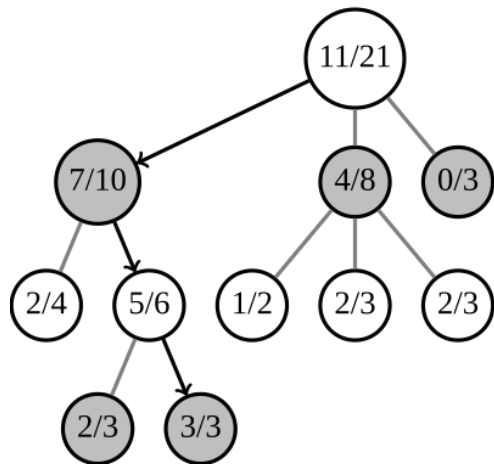
# Algorytm Monte Carlo Tree Search

- W algorytmie Monte Carlo Tree Search (MCTS) iteracyjnie tworzony jest graf przedstawiający możliwe stany gry. W każdym korku algorytm wykonuje jest symulacja gry.
- W każdym węźle zapisywana jest łączna liczba symulacji i liczba wygranych dla każdego węzła.

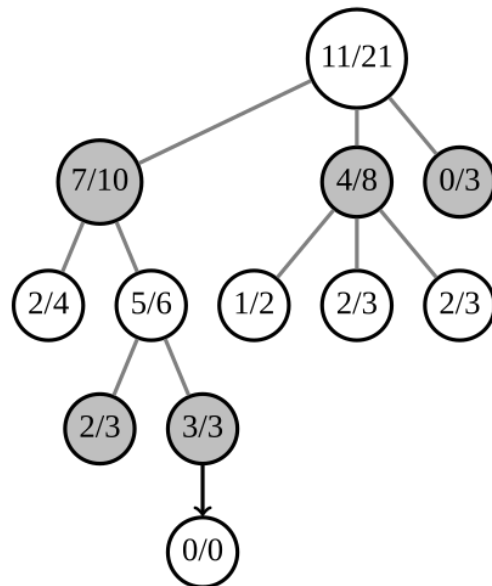


# Algorytm Monte Carlo Tree Search

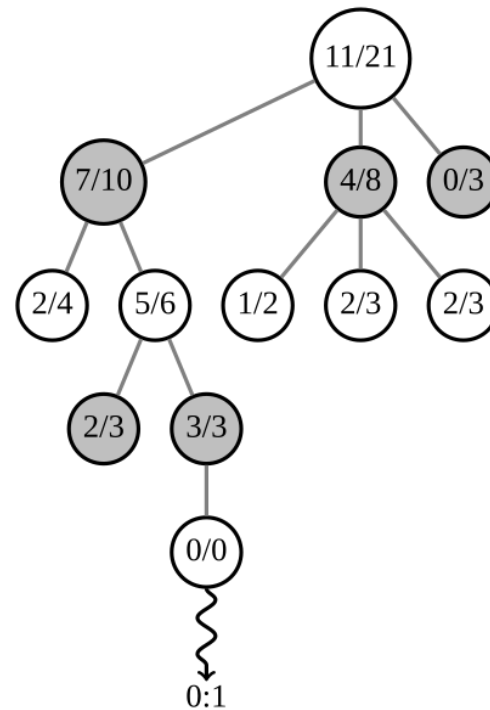
wybór



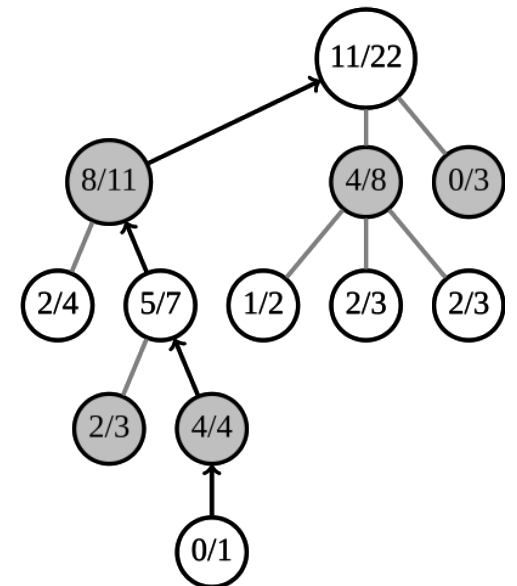
rozrost



symulacja



propagacja wstecz



---

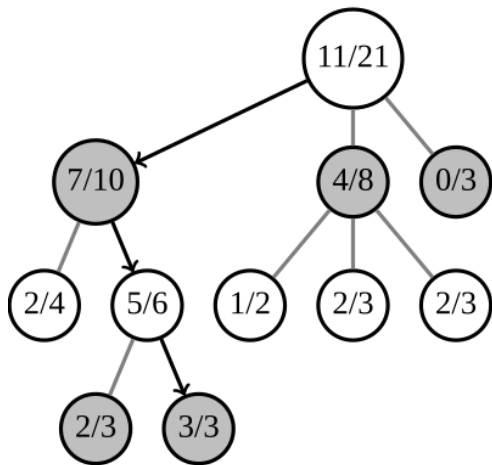
# Algorytm Monte Carlo Tree Search

wybór

rozrost

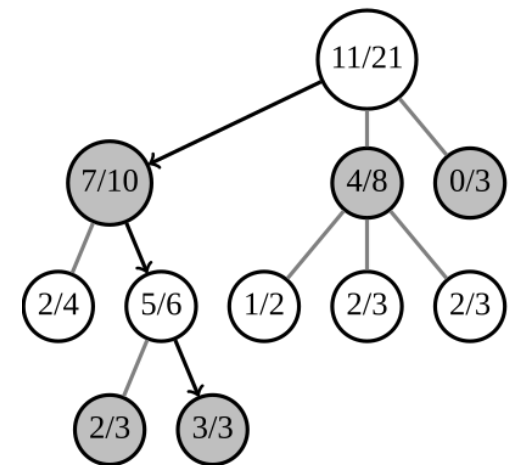
symulacja

propagacja wstecz



# Krok 1. Wybór

- W pierwszym kroku ze stanu początkowego wybieramy kolejne węzły potomne aż dojdziemy do jednego z liści drzewa.
  - Algorytm powinien preferować węzły, dla których do tej pory osiągnięty większy ułamek wygranych.
  - Jednocześnie algorytm powinien badać węzły, dla których wykonano niewiele symulacji, żeby uniknąć pominięcia bardzo dobrych ruchów.



wybór

rozrost

symulacja

propagacja wstecz

# Krok 1. Wybór

- L. Kocsis i C. Szepesvári w swojej pracy *Bandit based Monte-Carlo Planning* (2006) zaproponowali, że można wybrać węzeł potomny, dla którego wyrażenie

Liczba wygranych w danym węźle

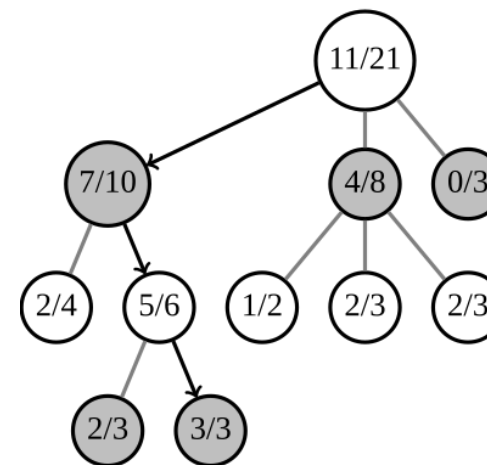
Liczba symulacji w danym węźle

Parametr eksploracji

$$\frac{w_i}{n_i} + c \sqrt{\frac{\ln N_i}{n_i}}$$

Liczba symulacji w rodzicu węzła

osiąga najwyższą wartość.



wybór

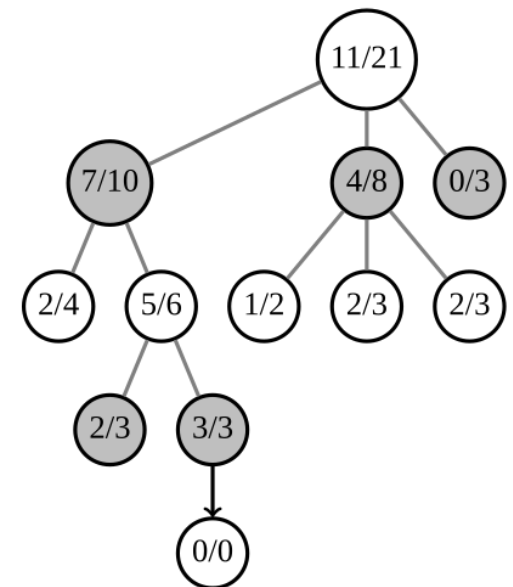
rozrost

symulacja

propagacja wstecz

## Krok 2. Rozrost

- Jeśli wybrany węzeł nie kończy gry, to dodaj do niego węzły potomne, odpowiadające wszystkim możliwym ruchom danego gracza.
- Spośród nich wybierz jeden węzeł, dla którego będzie wykonana symulacja.



wybór

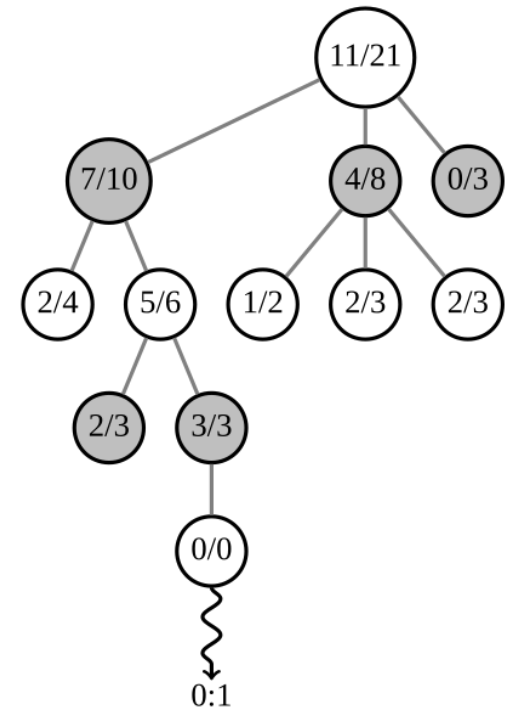
rozrost

symulacja

propagacja wstecz

## Krok 3. Symulacja

- Rozegraj losową symulację gry z wybranego wężła.
- W symulacji gracze wykonują losowe ruchy aż do zakończenia gry.



wybór

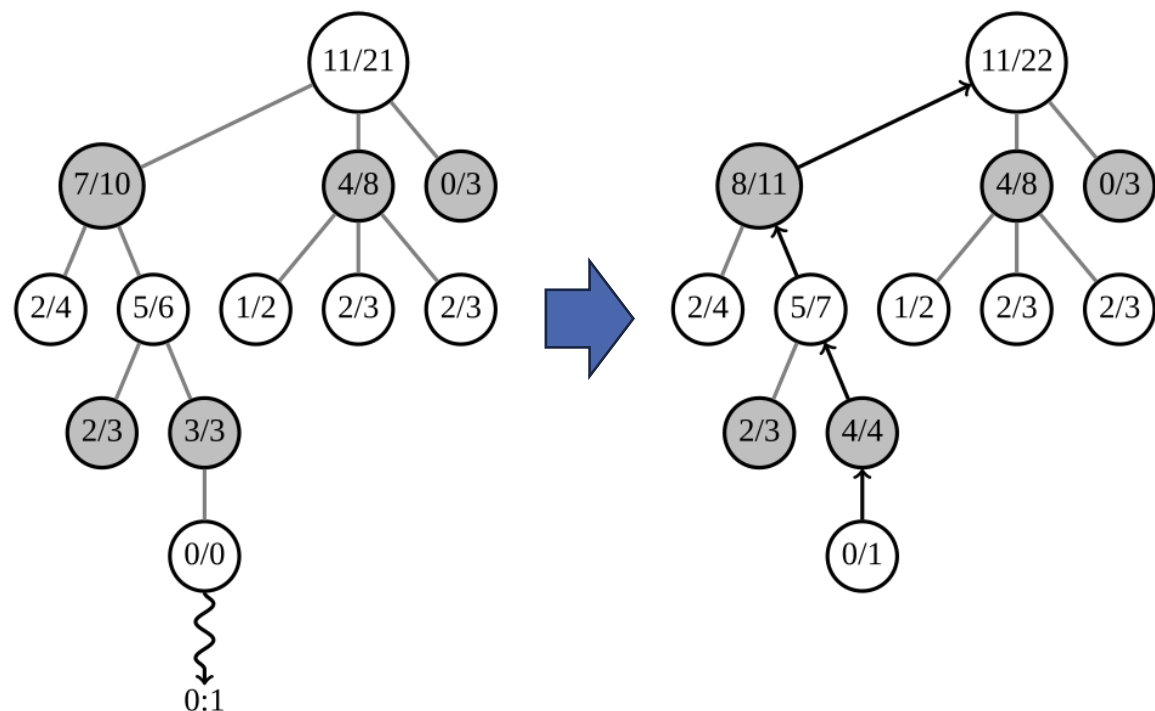
rozrost

symulacja

propagacja wstecz

## Krok 4. Propagacja wstecz

- Po zakończeniu symulacji, stan wszystkich węzłów na ścieżce do wybranego liścia jest aktualizowany:
  - Liczba przeprowadzonych symulacji dla każdego węzła rośnie o 1.
  - Jeśli gracz, który wykonał ruch prowadzący do danego węzła wygrał symulację gry, to liczba wygranych dla danego węzła też rośnie o 1.



wybór

rozrost

symulacja

propagacja wstecz

---

# Case study: wyspy Odyna



---

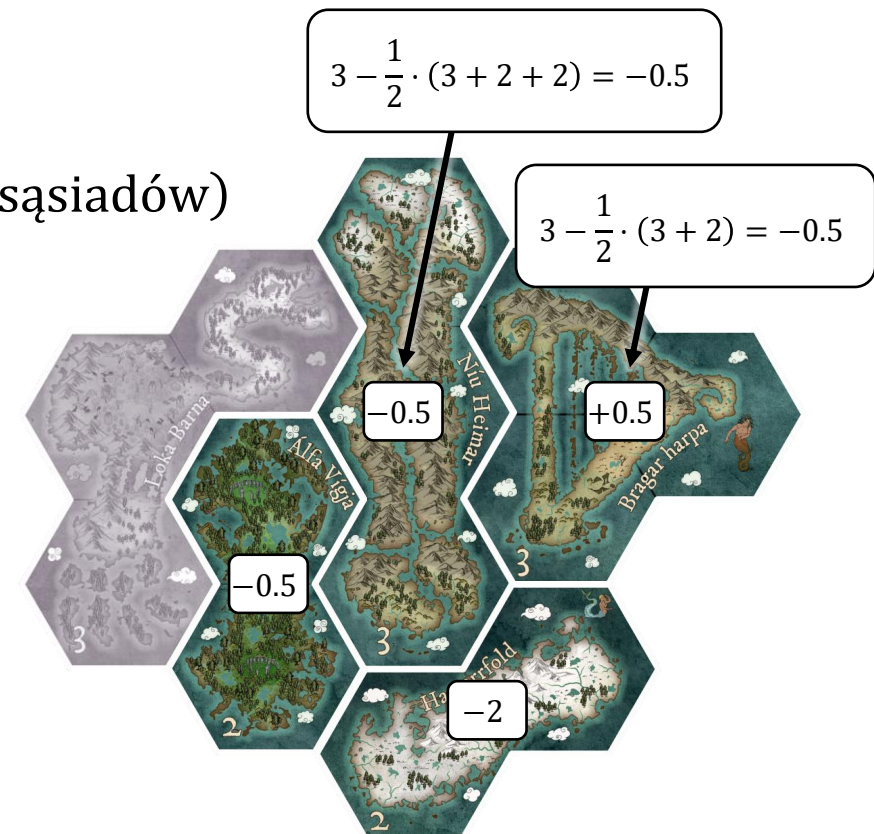
# Case study: wyspy Odyna (reguły gry)



# Case study: wyspy Odyna (heurystyczny algorytm)

- Dobrą strategię jest wybór wysp, które dają względnie dużo punktów, ale blokują jak najmniej innych wysp.
- Dla każdej wyspy możemy ocenić jej wartość następująco:  
(wartość wyspy) = (wielkość wyspy) -  $K \cdot$  (łączna wielkość jej sąsiadów)

gdzie  $K > 0$  to pewna stała, np.  $K = \frac{1}{2}$



---

# Case study: wyspy Odyna (algorytm MCTS)

ToDo: Animowane drzewo gry

---

---

# Case study: wyspy Odyna (porównanie algorytmów 1)

Animacja gry MCTS vs heuristic

Losowa symulacja



Ruchy heurystyczne

---

# Case study: wyspy Odyna (porównanie algorytmów 2)

Animacja gry MCTS vs heuristic

Losowa symulacja



Ruchy heurystyczne

# Case study: wyspy Odyňa



**ISLANDS OF ÓÐINN**

**Overview**

As a Viking leader, you sail north to colonise a newly discovered archipelago. The primary goal of the game is to colonise more land than any other player. There is a catch, though! After colonising any island, the rival tribes will not allow you to colonise the neighbouring lands, unless they have their own share in this part of the archipelago too.

**Mechanics**

Players in turns choose an island to colonise. However, in the base game, you cannot colonise an island if at least one of your opponent's owns fewer neighbouring islands than you. This simple mechanic forces players to strategise between colonising the most valuable islands while avoiding giving the opponent too much advantage in the end-game.

Advanced players can additionally use 28 rule modifications that add new game mechanics and provide new challenges. You can organise ale festivals, tame dragons, invade opponents or do all of it at once!

The number of rules combinations is practically endless, and each one will require players to adapt their strategies or even completely change their play style.

View all by Piotr Morawiecki

Follow Piotr Morawiecki

Add To Collection

Hasło: Odinn

<https://piotrmorawiecki.itch.io/islands-of-odinn>

# Implementacja w Pythonie (Connect 4)



floriangardin / connect4-mcts Public

<> Code Issues Pull requests Actions Projects Security Insights

master 1 Branch 0 Tags Go to file Code

floriangardin change readme 56bc720 · 7 years ago 2 Commits

connect4	initial commit	7 years ago
.gitignore	initial commit	7 years ago
README.md	change readme	7 years ago
main.py	initial commit	7 years ago
requirements.txt	initial commit	7 years ago

README

## Connect 4 AI

- This little project implements a MCTS (Monte-Carlo tree search) for connect 4
- Just launch the `main.py` and you will be able to play against the AI.
- To play you have to input the column number (between 0 and 6).

<https://github.com/floriangardin/connect4-mcts/tree/master>



---

# Podsumowanie wykładu

- W przypadku wielu gier znalezienie optymalnej strategii jest niewykonalne ze względu na wielkość drzewa gry.
- Algorytm *Monte Carlo Tree Search* (MCTS) pozwala podejmować decyzję o ruchu, eksplorując jedynie część drzewa gry.
- Można polepszyć algorytm wykorzystując dodatkowe informacje dot. gry (np. przez wykorzystanie heurystycznych metod oceny stanu gry).
- Za tydzień dowiemy się jak dzięki algorytmowi MCTS udało się pokonać mistrza gry Go.

