Infinity War

Marius Hadry

Florian Pircher

Universität Würzburg

2. Juni 2021

Agenda

1. Aufbau
2. Spielzüge
3. Regeln
4. Beispiele
5. Implementierung
6. Laufzeitanalyse
7. Tipps

Aufbau

- Alice und Bob spielen eine Schach-Variante
- $n \times n$ Brett mit n Zugregeln
- $2 \leq n \leq 100.000$
- Endspiel: jeder ist noch einmal an der Reihe

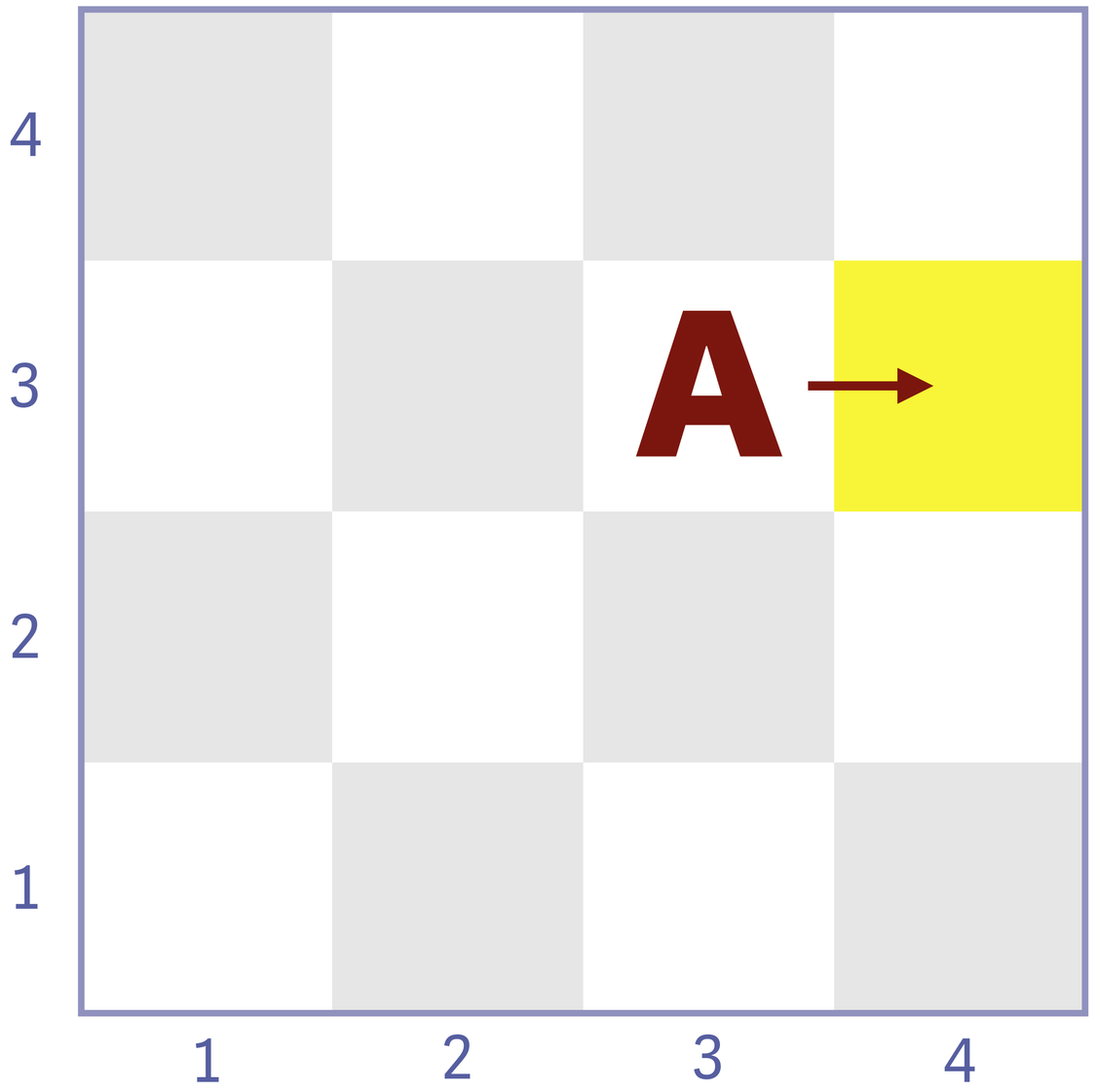
Alice

Bob

Spielzüge

→1 ↑0

Beispiel: $n = 4$

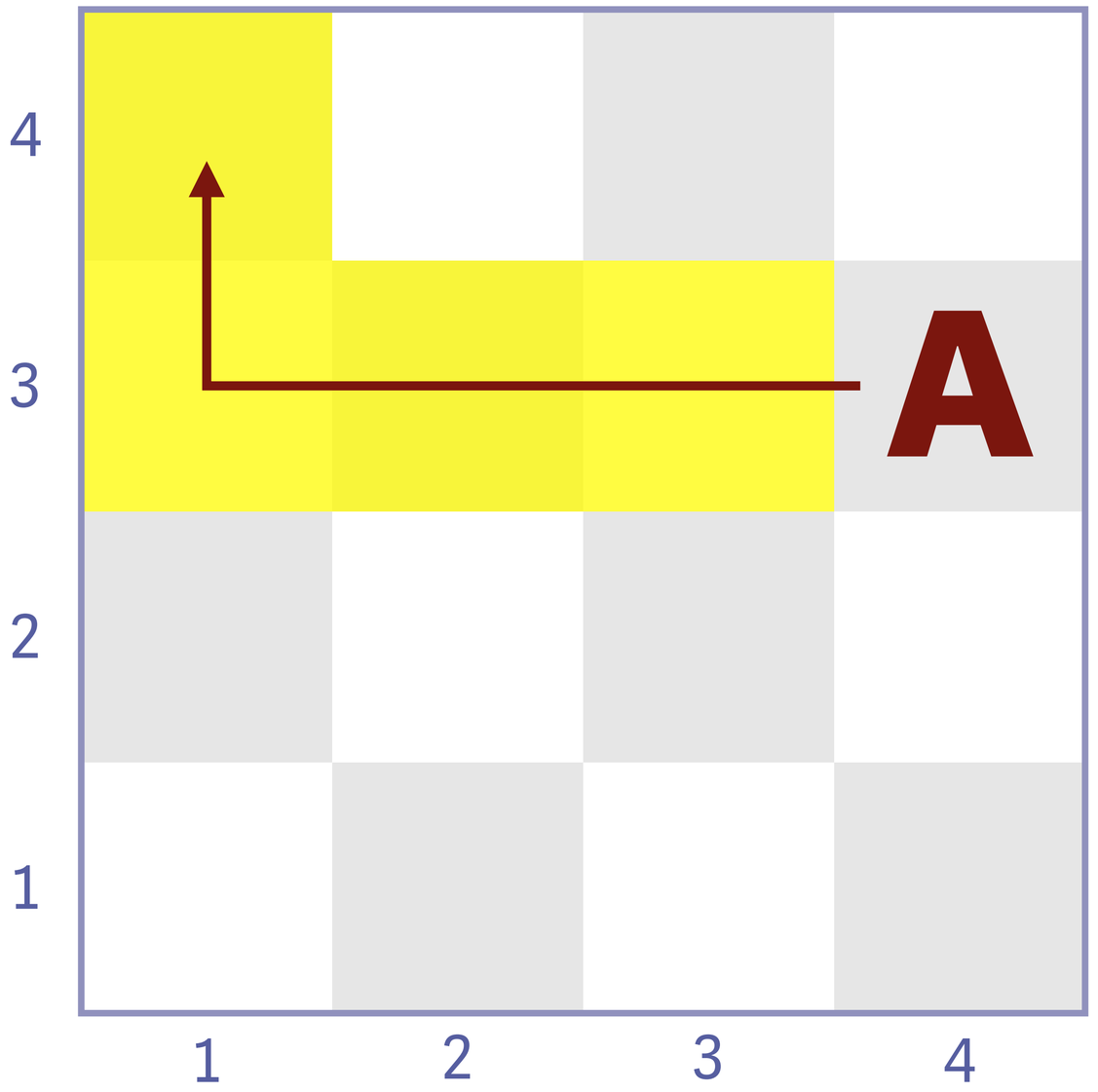


Spielzüge

→1 ↑0

←3 ↑1

Beispiel: $n = 4$



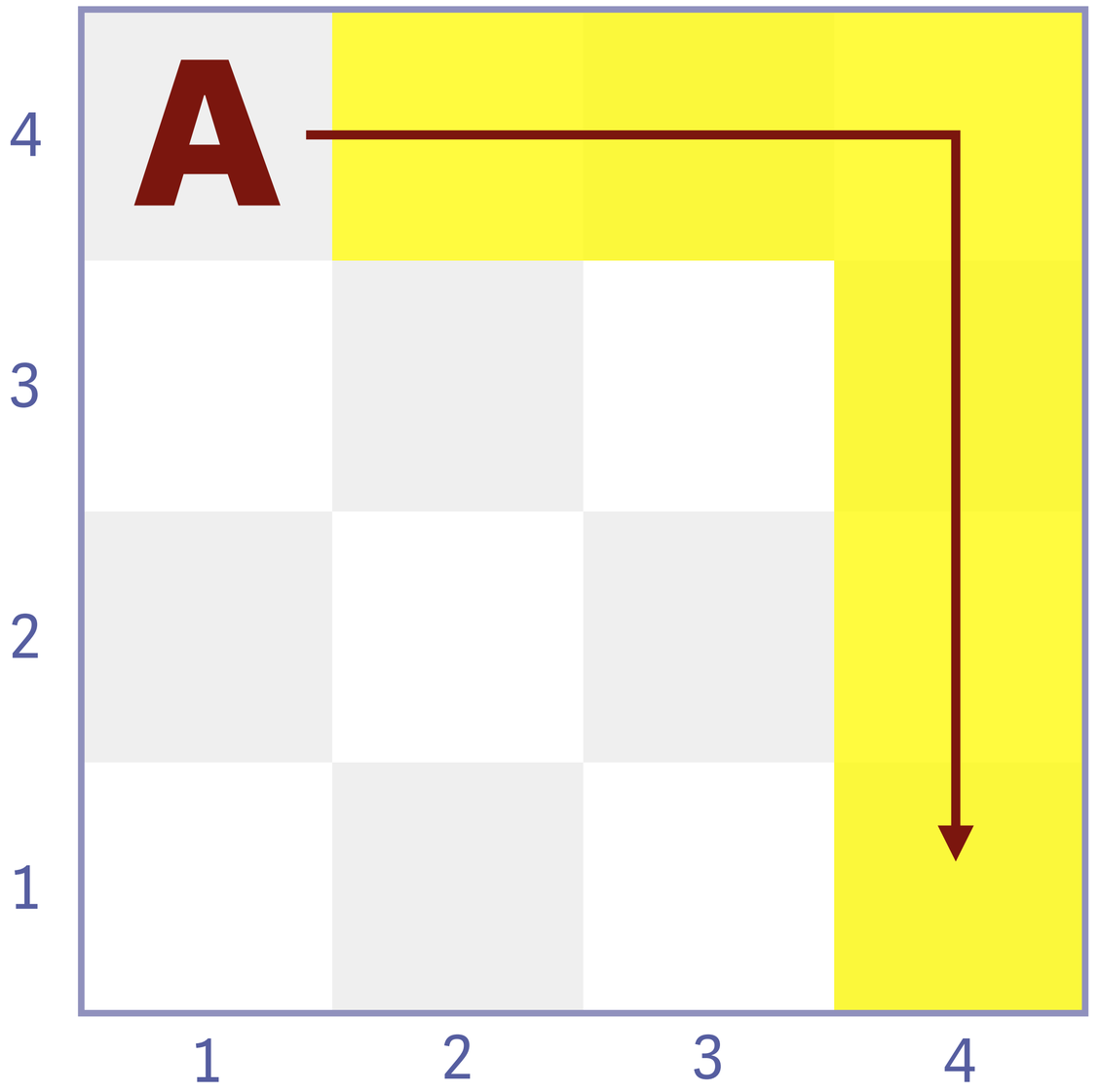
Spielzüge

→1 ↑0

←3 ↑1

→3 ↓3

Beispiel: $n = 4$



Spielzüge

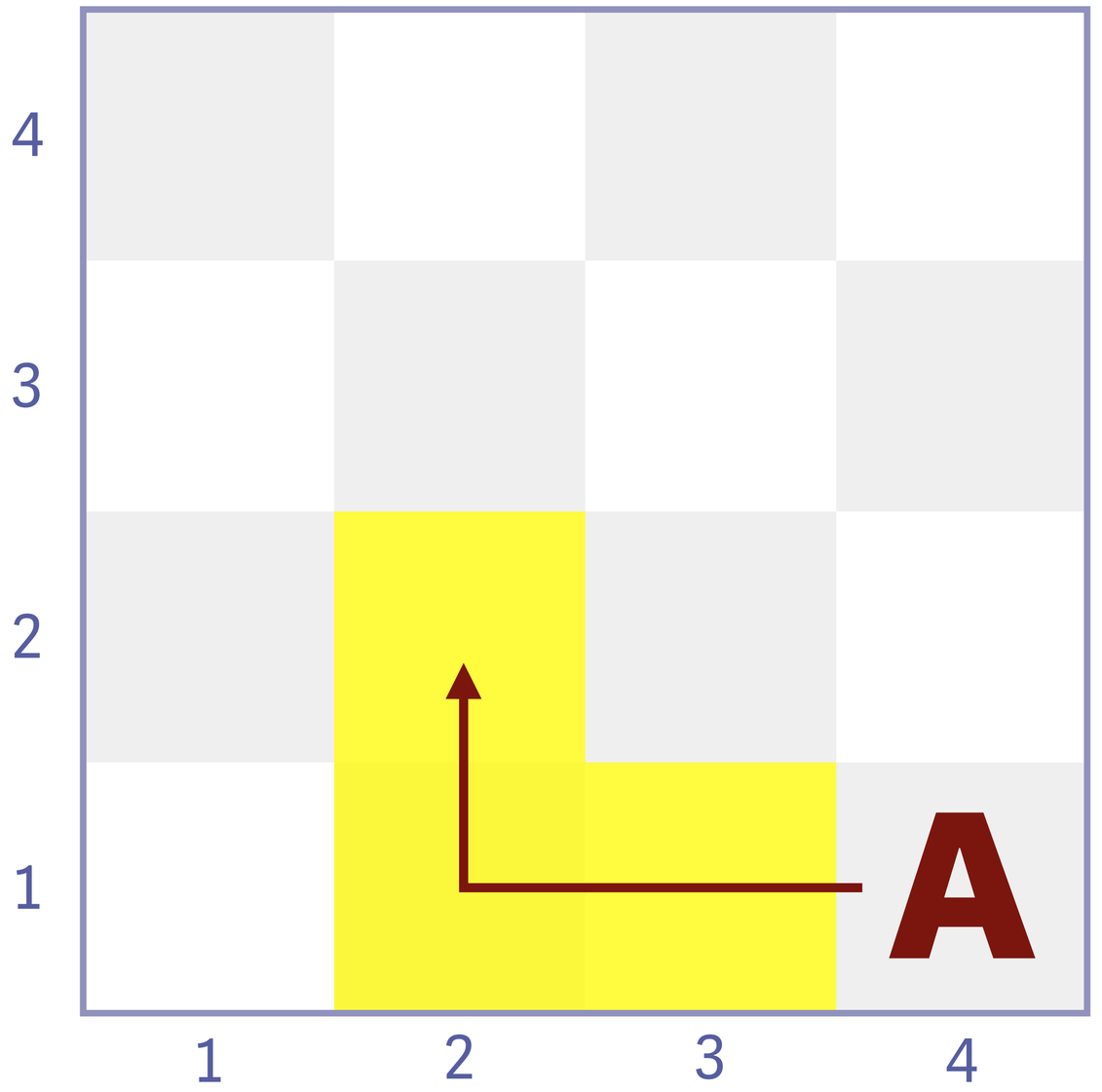
→1 ↑0

←3 ↑1

→3 ↓3

←2 ↑1

Beispiel: $n = 4$



Regeln

→1 ↑0

←3 ↑1

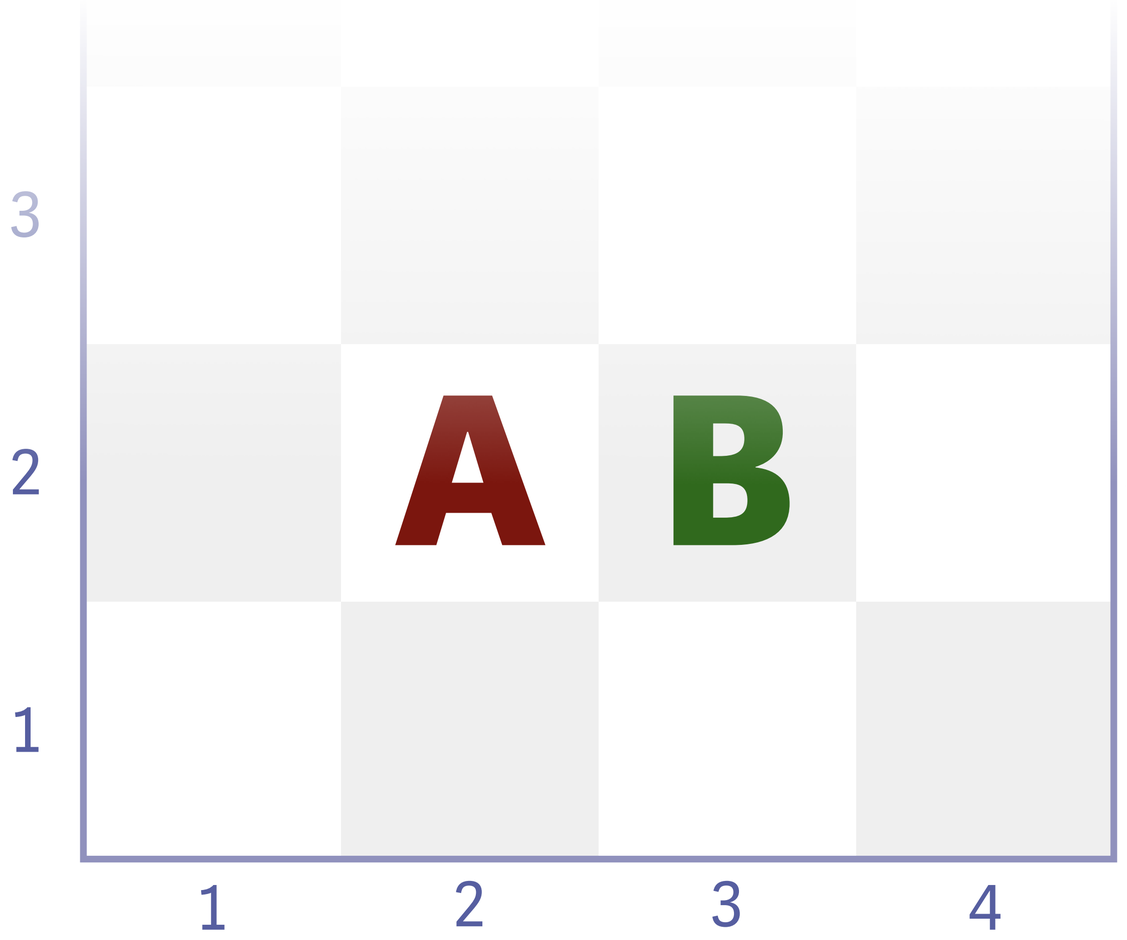
→3 ↓3

←2 ↑1

Eine Figur darf **0**, **1** oder **2** Züge machen …

… oder auf ein freies Feld **teleportiert** werden.

Zunächst ist Alice an der Reihe, dann Bob.



Beispiel 1

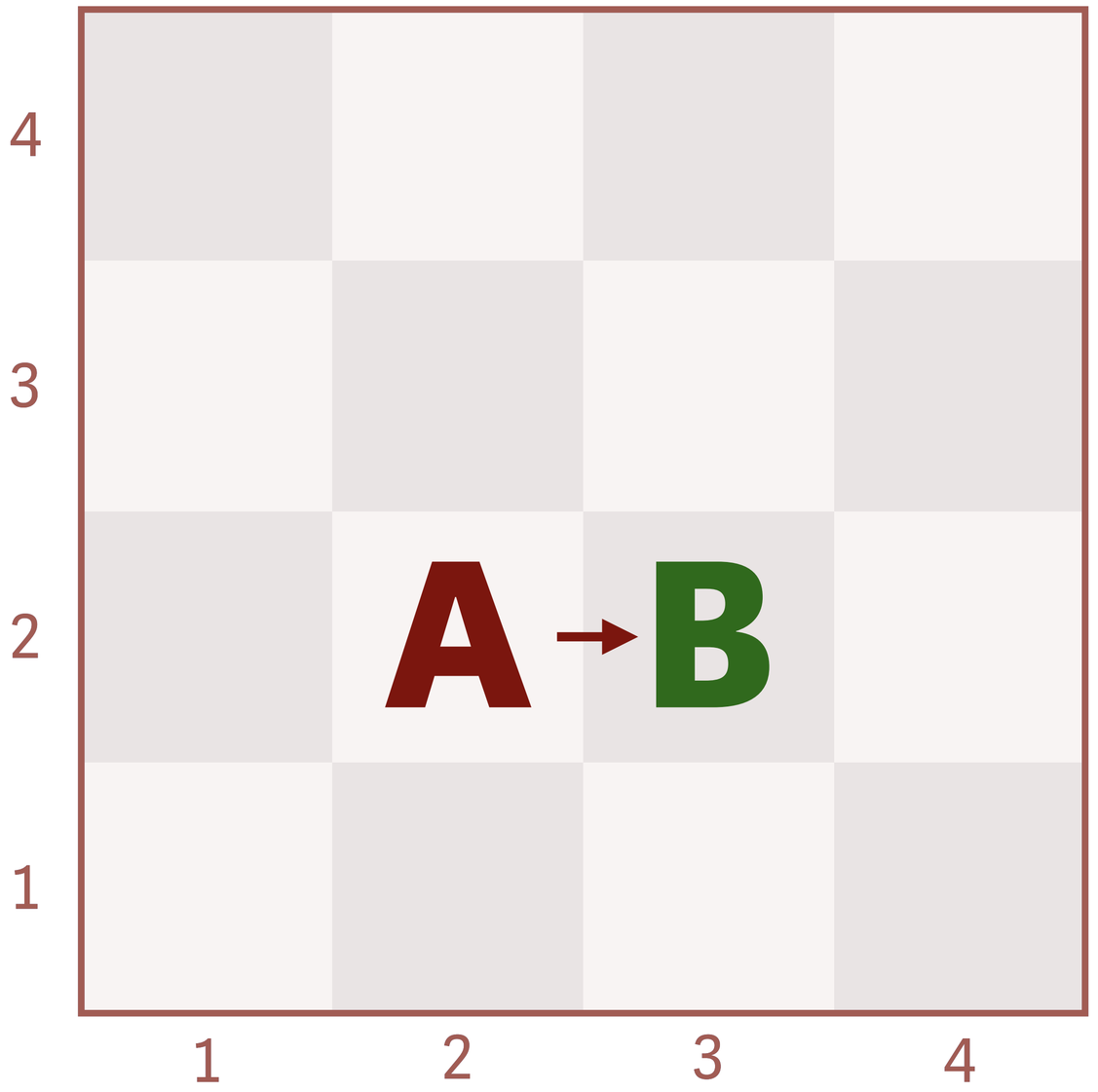
→1 ↑0

←3 ↑1

→3 ↓3

←2 ↑1

Eine Figur darf 0, 1 oder 2 Züge machen.



AUSGABE

```
Alice wins
```

Beispiel 2

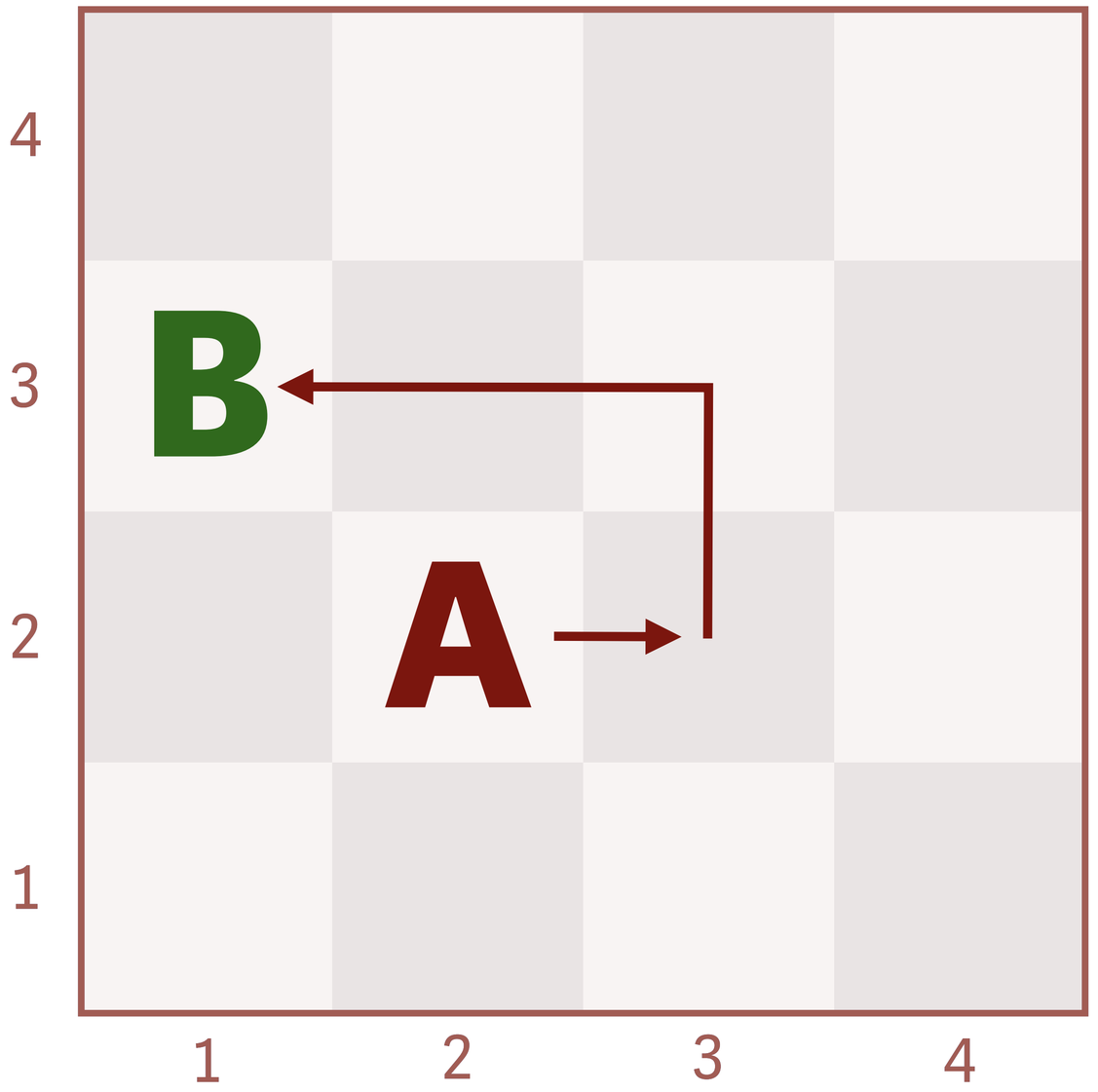
→1 ↑0

←3 ↑1

→3 ↓3

←2 ↑1

Eine Figur darf 0, 1 oder 2 Züge machen.



AUSGABE

```
Alice wins
```

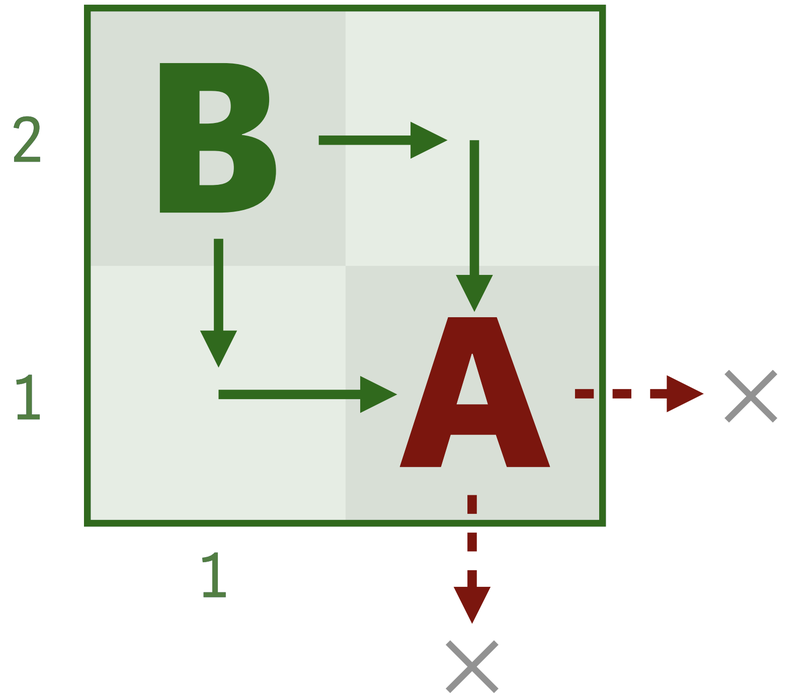
Beispiel 3

Eine Figur darf 0, 1 oder 2 Züge machen …

… oder auf ein freies Feld teleportieren.

→1 ↑0

→0 ↓1



AUSGABE

```
Bob wins
```

Falls **Alice** nicht gewinnen kann:

- Unentschieden
- **Bob** gewinnt

Beispiel 4

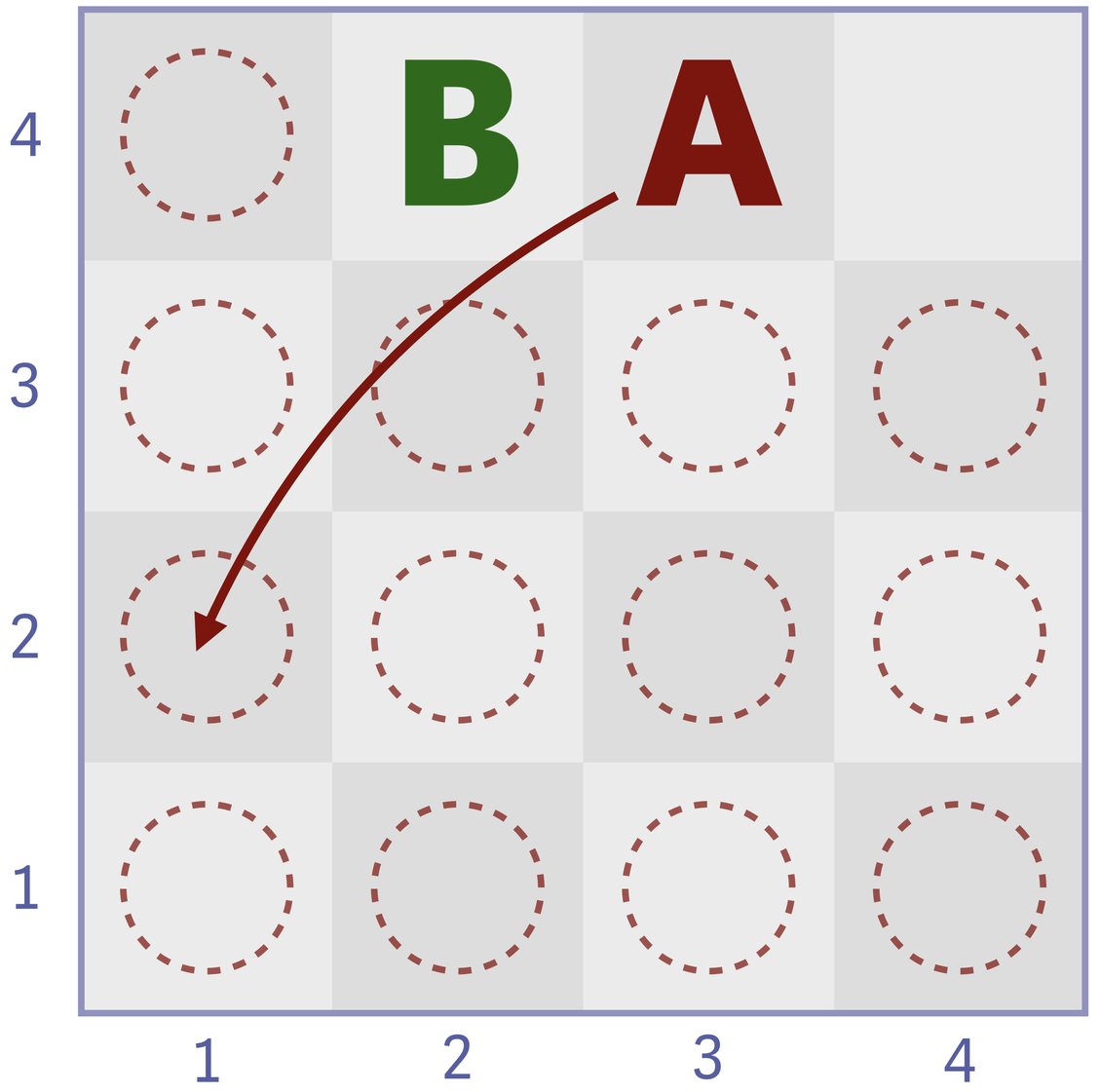
→1 ↑0

←3 ↑1

→3 ↓3

←2 ↑1

Alice kann auf ein sicheres Feld ◌ teleportieren.



AUSGABE

```
tie 1 2
```

Implementierung

1. Alice gewinnt
2. Unentschieden
3. Bob gewinnt

Alice gewinnt

Brute Force

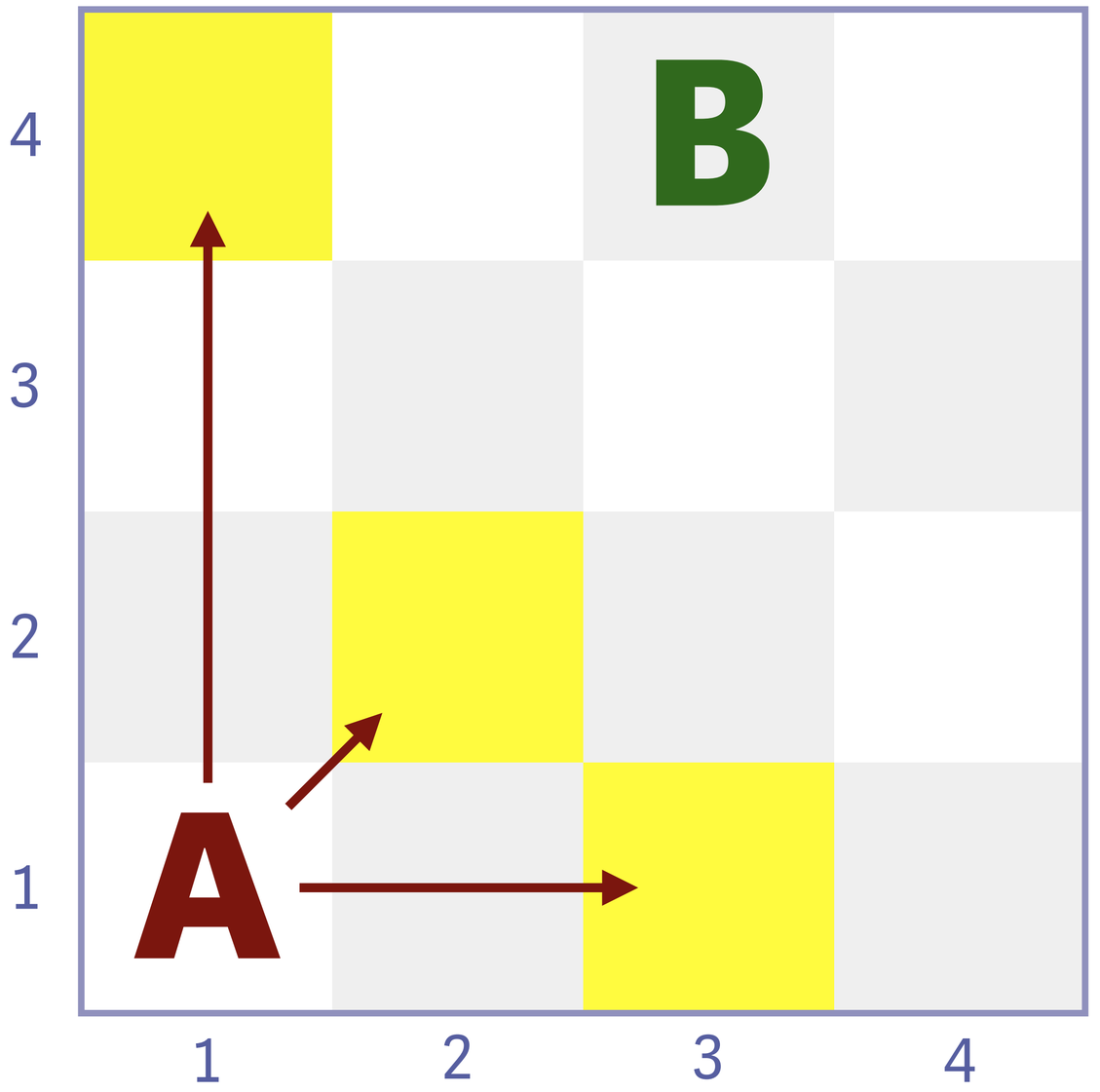
n Felder × n Felder = n^2 Felder bei $2 \leq n \leq 100.000$

→0 ↑3

→2 ↑0

←2 ↓3

→1 ↑1



Alice gewinnt

Brute Force

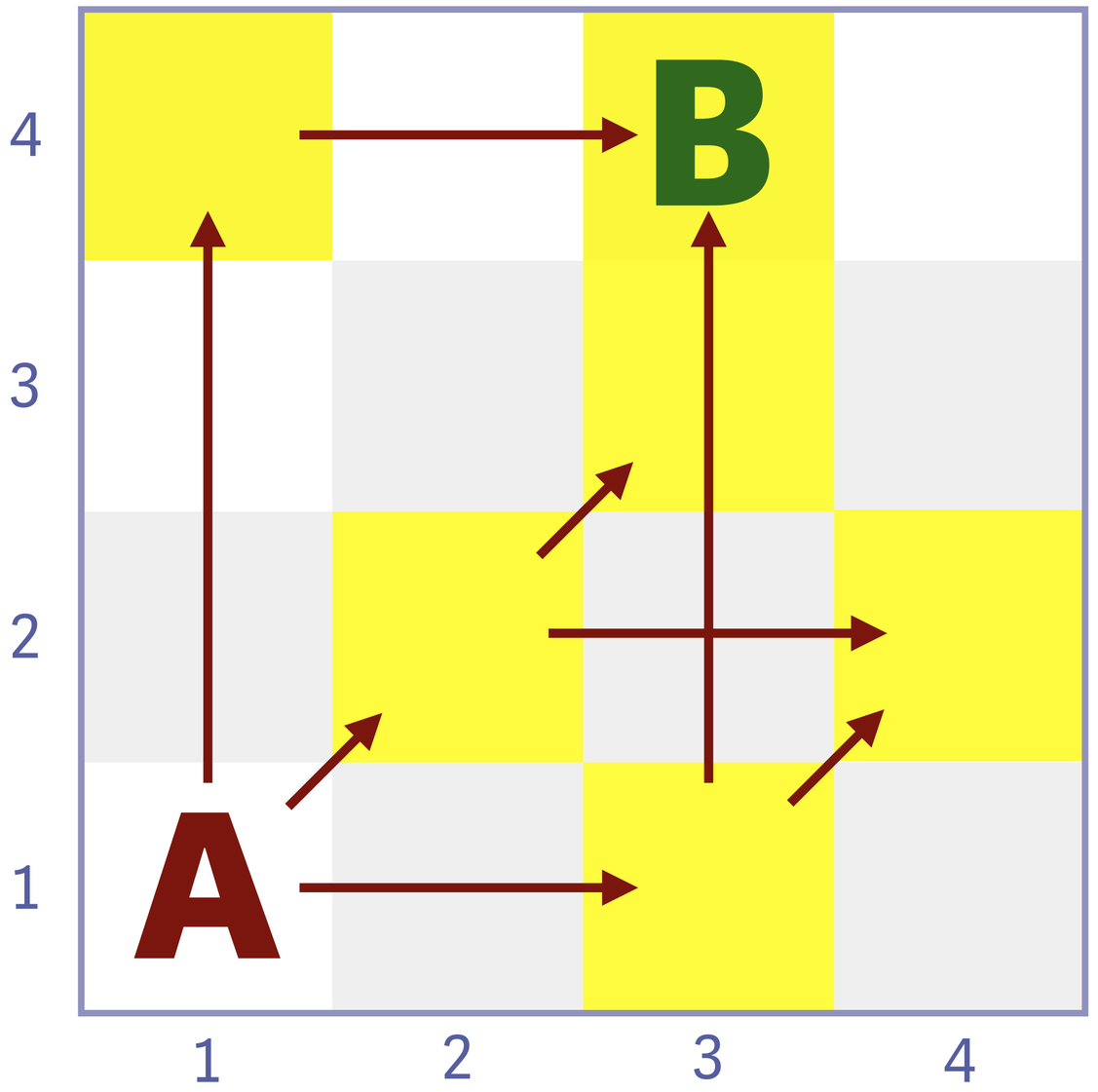
n^2 Felder bei $2 \le n \le 100.000$ → zu viel für große n

→0 ↑3

→2 ↑0

←2 ↓3

→1 ↑1



Alice gewinnt

Bidirektionale Suche

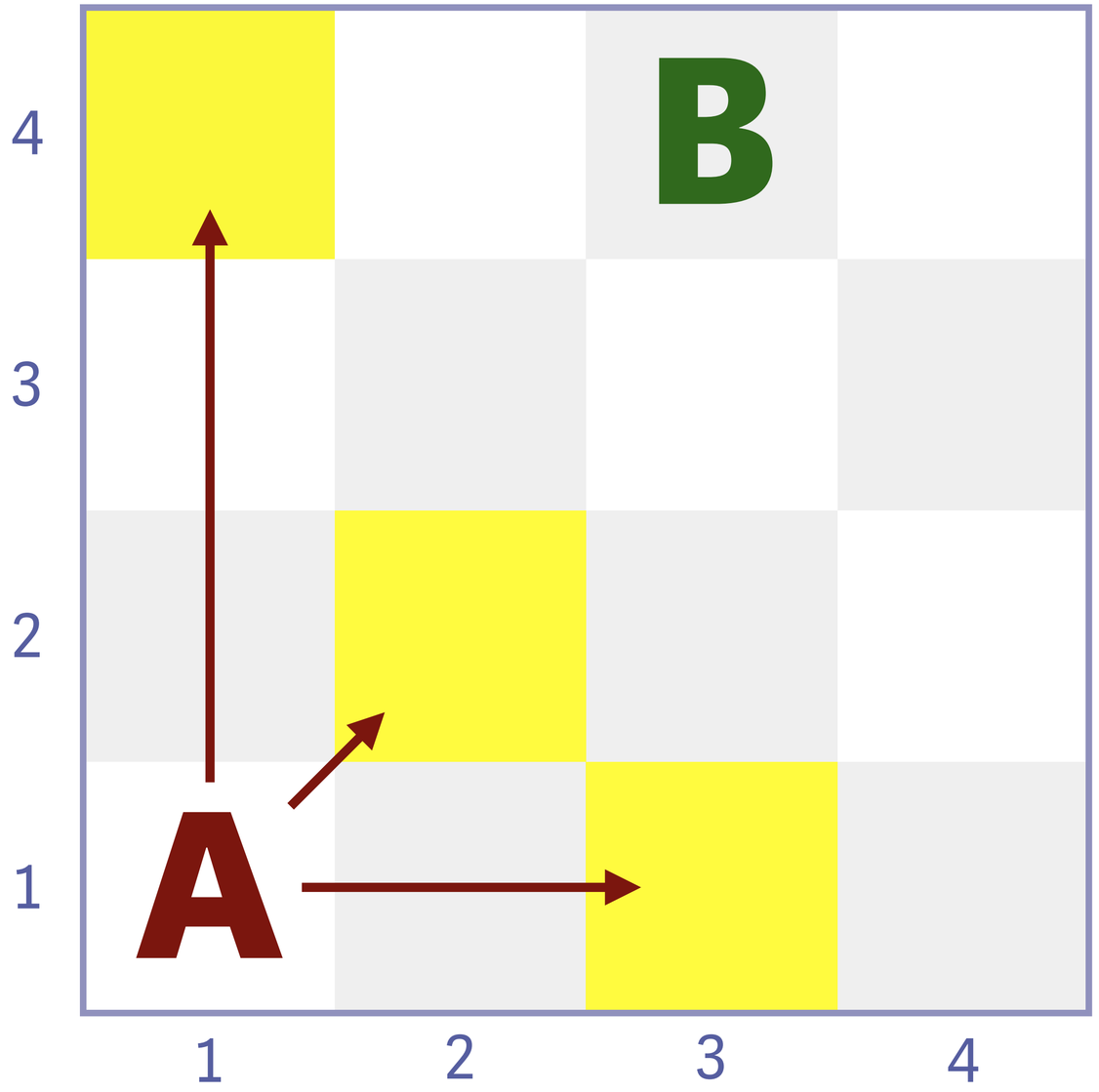
→0 ↑3

→2 ↑0

←2 ↓3

→1 ↑1

Züge kopieren
& invertieren

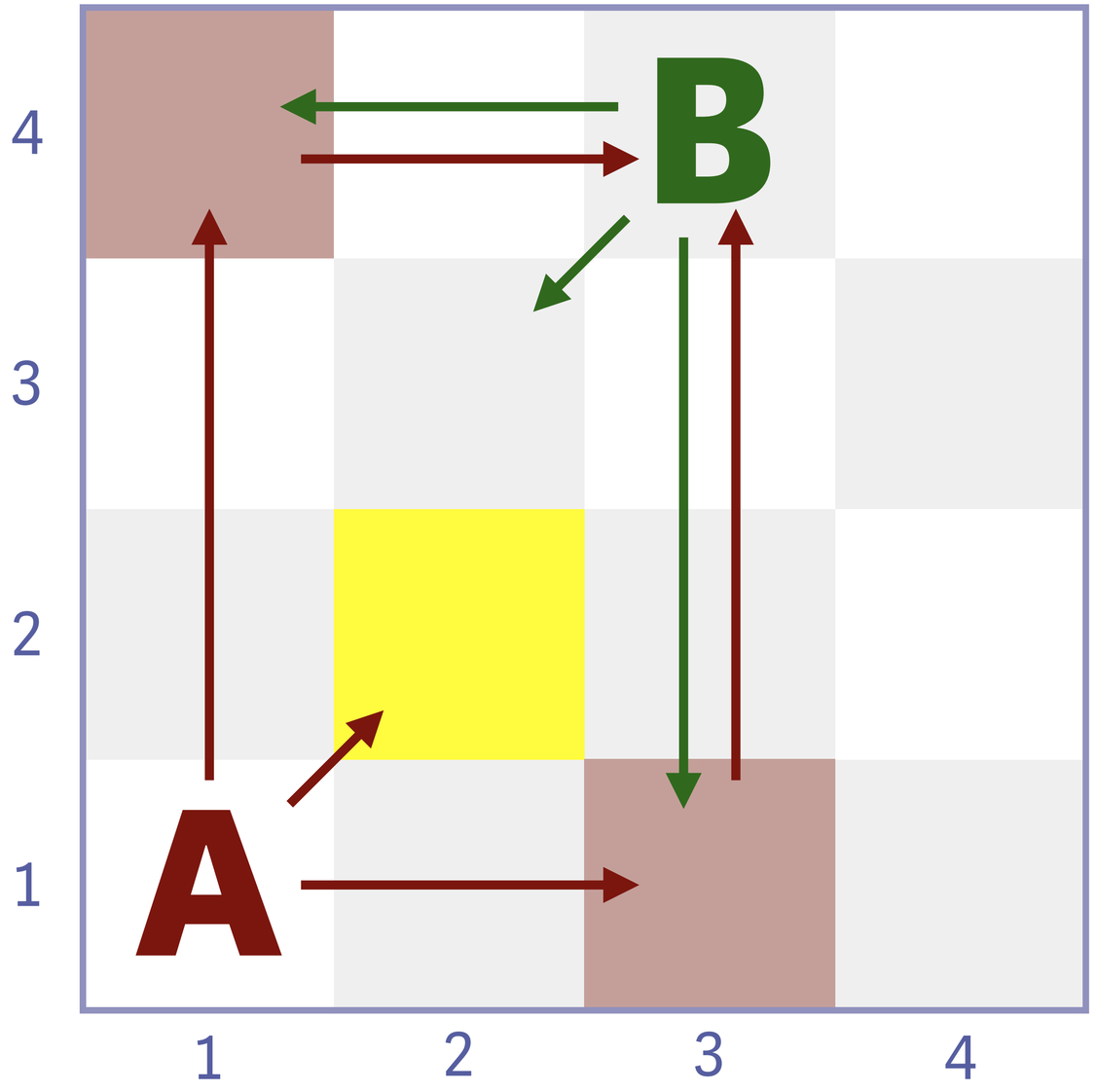


Alice gewinnt

Bidirektionale Suche

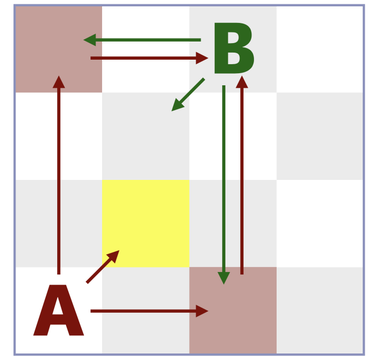
n Felder + n Felder → auch für große n

	INVERTIERTE ZÜGE
→0 ↑3	←0 ↓3
→2 ↑0	←2 ↓0
←2 ↓3	→2 ↑3
→1 ↑1	←1 ↓1



Alice gewinnt

Laufzeitanalyse



function expandPoint(*point* : (*x*, *y*), *moves* : set of (Δ*x*, Δ*y*)) O(*n*)

    return [(*point.x* + Δ*x*, *point.y* + Δ*y*) **for** (Δ*x*, Δ*y*) **in** *moves*] O(*n*)

function bidirectionalSearch(*start*, *end*, *moves*) O(*n*)

    u = expandPoint(*start*, *moves*) O(*n*)

    v = expandPoint(*end*, −*moves*) O(*n*)

    return intersection(*u*, *v*) O(*n*)

if bidirectionalSearch(Alice, Bob, *moves*) **is empty** O(*n*)

    Alice kann Bob nicht schlagen

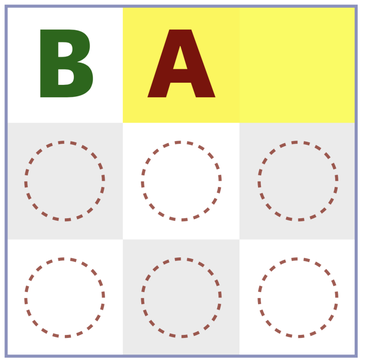
Alice gewinnt

Unentschieden

Bob gewinnt

Unentschieden

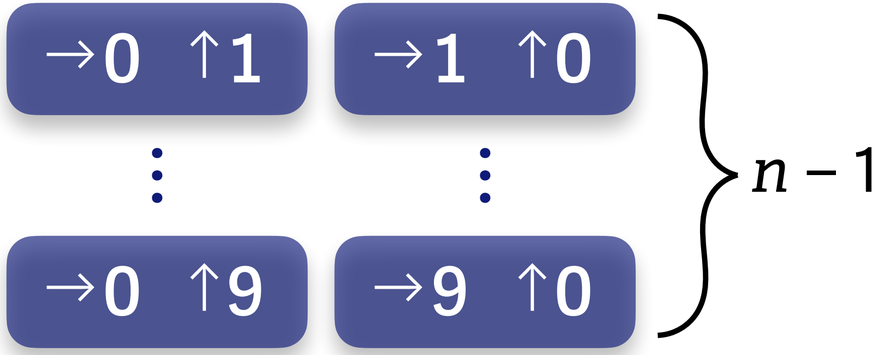
Laufzeitanalyse – *Brute Force*




```
function findTieFieldBruteForce(n, moves)                                  O(n³)
    for (x, y) in n × n                                                    O(n³)
        if (x, y) == Bob
            skip iteration
        matches = bidirectionalSearch(Bob, (x, y), moves)                  O(n)
        if matches is empty — Bob kann das Feld nicht erreichen
            return (x, y)
    return none
```

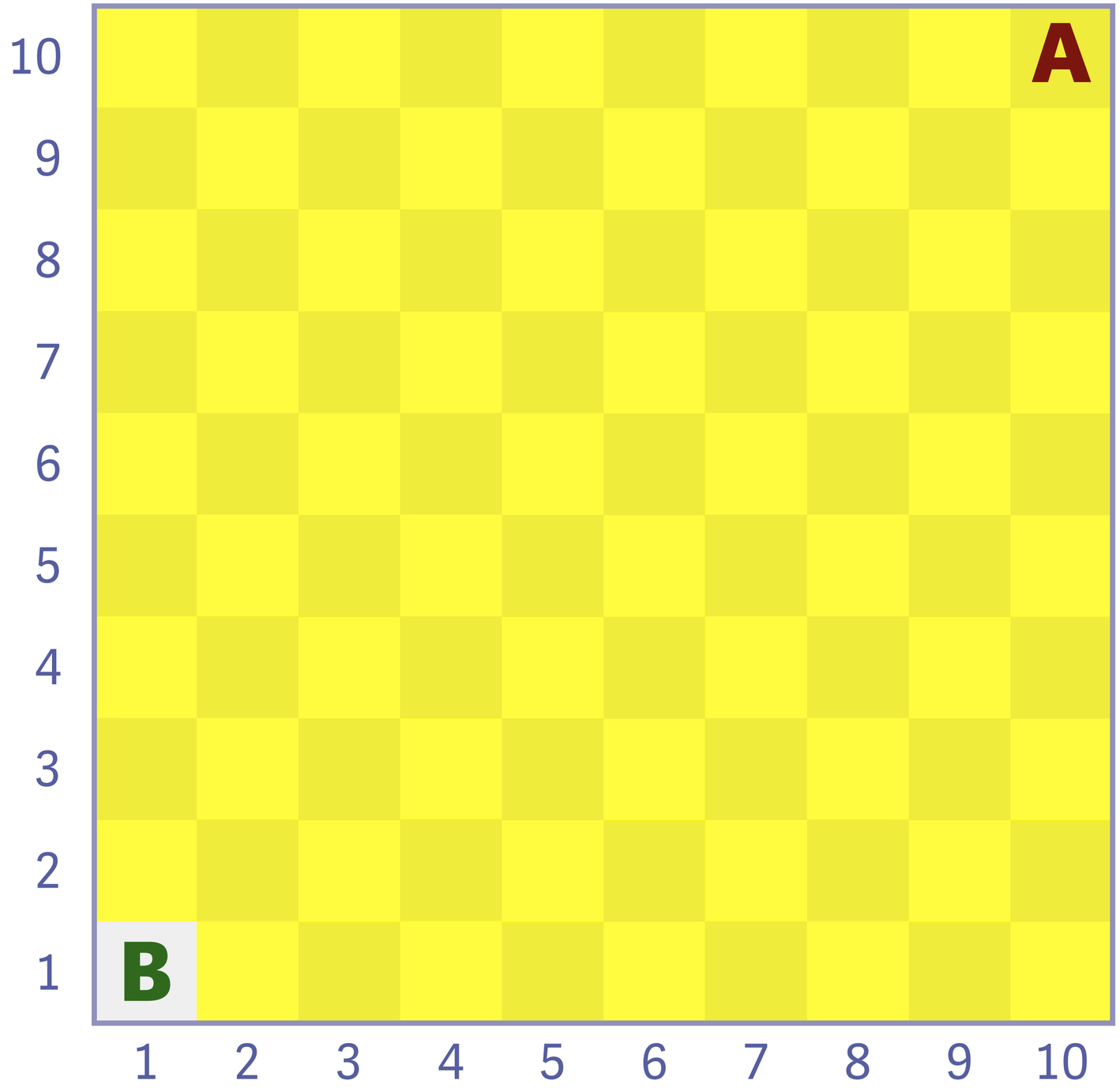
n^2 Felder in $O(n)$ → Laufzeit in $O(n^3)$ → nicht für große Felder geeignet

Unentschieden



- **Bob** benötigt $2(n-1)$ Zugregeln um jedes Feld zu erreichen
- Stets nur n Zugregeln verfügbar
- Wahrscheinlichkeit, dass **Bob** ein zufälliges Feld erreichen kann:

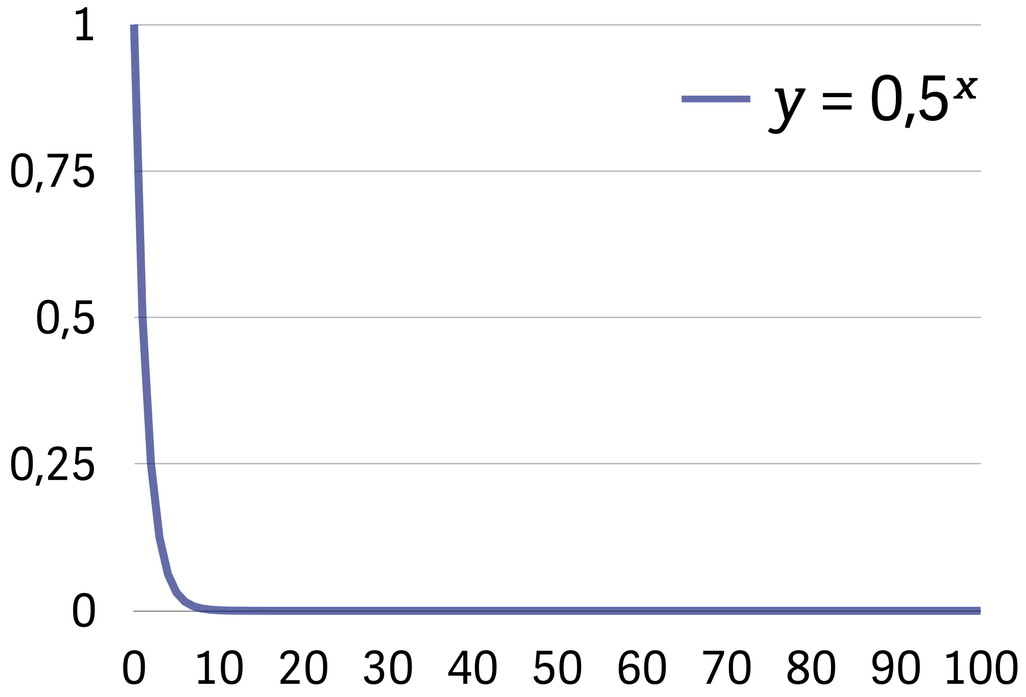
$$\lim_{n \to \infty} \frac{2(n-1)}{n} = \frac{1}{2}$$

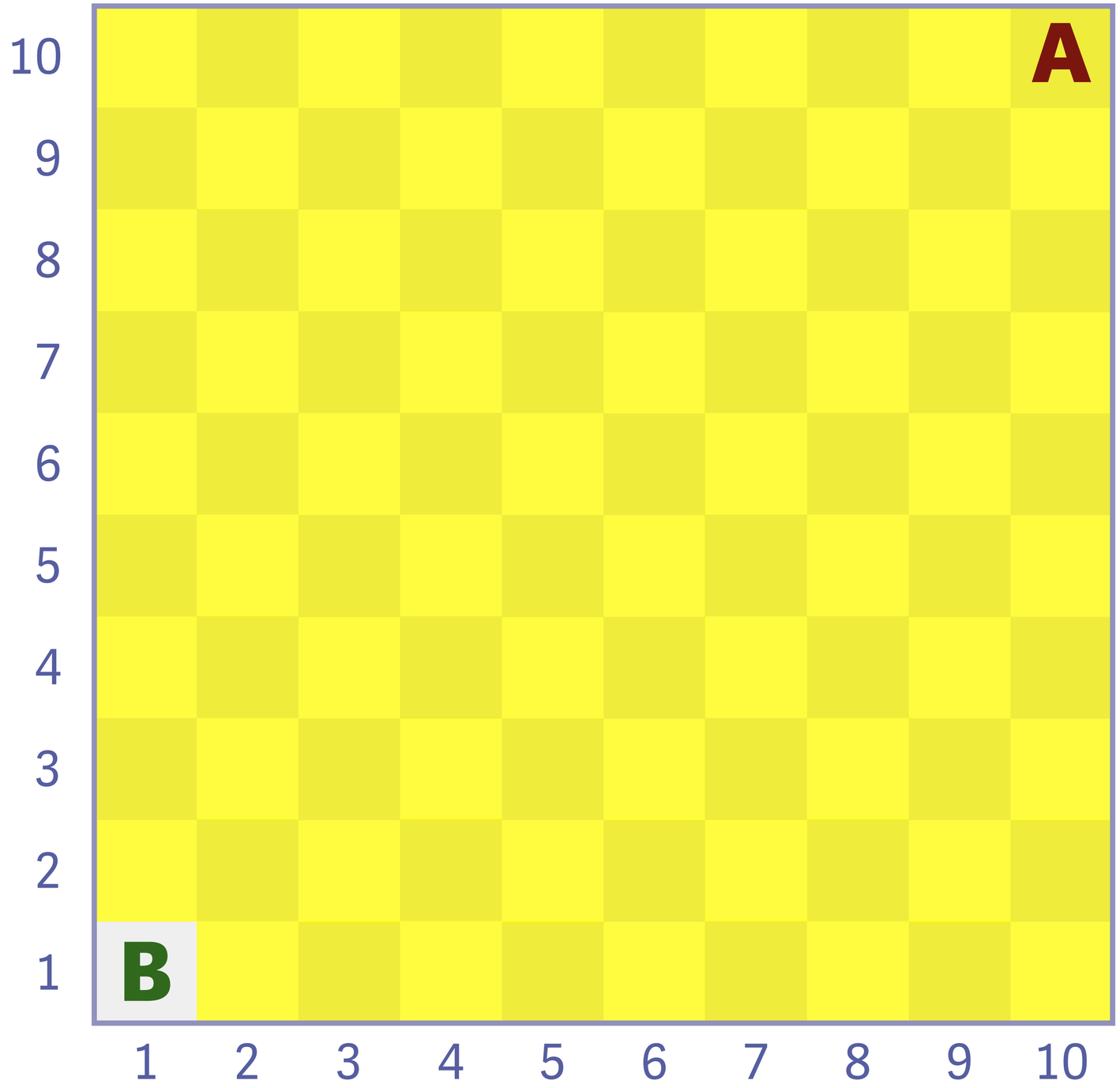


Unentschieden

- Wahrscheinlichkeit, aus 50 Feldern kein Sicheres zu finden:

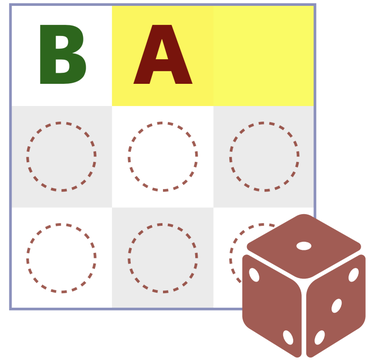
$$0{,}5^{50} = 8{,}88 \cdot 10^{-16}$$





Unentschieden

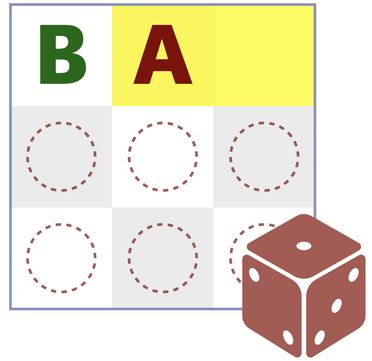
Korrektheit



- Der Unentschieden-Test ist ein Monte-Carlo Algorithmus, da auch ein falsches Ergebnis entstehen kann
- Das Problem gehört zur Komplexitätsklasse RP (randomized polynomial time)
    - Ein Unentschieden-Feld wird mit Wahrscheinlichkeit $0{,}5^{50}$ nicht gefunden
    - Falls kein Unentschieden-Feld existiert, wird dies mit Wahrscheinlichkeit 1 erkannt

Unentschieden

Laufzeitanalyse – *randomisiert*




```
function findTieFieldRandomized(n, moves)                          O(n)
    repeat 50 times                                                O(n)
        (x, y) = random(n × n)
        if (x, y) == Bob
            skip iteration
        matches = bidirectionalSearch(Bob, (x, y), moves)          O(n)
        if matches is empty — Bob kann das Feld nicht erreichen
            return (x, y)
    return none
```

50 zufällige Felder in O(n) → Laufzeit in O(n) → gut geeignet für große Felder

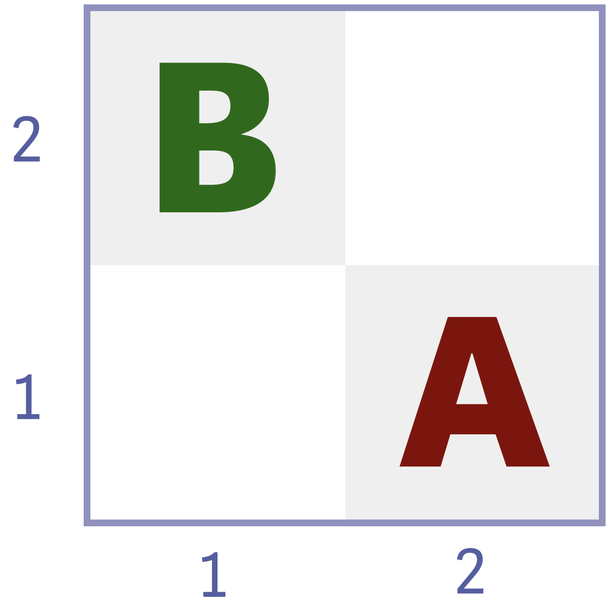
Alice gewinnt

Unentschieden

Bob gewinnt

Bob gewinnt

- Gegeben:
  - Alice kann Bob nicht schlagen
  - Alice kann kein Unentschieden provozieren
- Schlussfolgerung:
  - Bob hat gewonnen



Laufzeitanalyse

- Einlesen der Daten $O(n)$
- Alice gewinnt prüfen mit bidirektionaler Suche $O(n)$
- Unentschieden prüfen
  - Brute Force, falls Feldgröße < 15 $O(n^3)$
  - Randomisiert, sonst $O(n)$
- Sonst gewinnt Bob $O(1)$

Gesamtlaufzeit in $O(n)$

Tipps

- Null-Zug zu Menge von Zugregeln aufnehmen
  - *Null* → *Null* kein Zug
  - *Null* → Zug ein Zug
  - Zug → Zug zwei Züge
  - beeinflusst nicht die asymptotische Laufzeit
- Figuren dürfen die Grenzen des Spielfeldes nicht verlassen
- Unentschieden-Test: expandPoint(Bob, *moves*) nur einmal berechnen