Chapter 3: Nash Equilibrium

Algorithmic Game Theory

Formal definition · Best response · Dominated strategies · Mixed strategies · Existence theorem · Cournot and Bertrand · Equilibrium selection · Correlated equilibrium · Minimax · Security games · PPAD

Sercan KÜLCÜ · Giresun University · Computer Engineering

© Sercan Külcü. All rights reserved. · Interactive HTML version of these slides: https://sercankulcu.github.io/files/agt/slides/Chapter_03_Nash_Equilibrium.html

Warm-up Game: The Prisoner's Dilemma

Rules: Sam and John are questioned in separate rooms. Without knowing the other's choice, each either stays *Silent* or *Betrays*, and serves the sentence shown in the table. Payoff = −(years in prison), so bigger is better.

Sam \ John	Silent	Betray
Silent	(−1 , −1)	(−3 , 0)
Betray	(0 , −3)	(−2 , −2)



choices are hidden, then revealed together

Cell (Sam, John): payoff = −(years in prison)

Question 1

In Sam's place, not knowing what John will do, would you stay silent or betray?

Question 2

Against a computer playing Tit-for-Tat (silent in round one, then copies your previous move), what is the fewest total years you can serve over ten rounds?

Learning Outcomes

Definition

State the formal definition of Nash equilibrium and relate it to best responses.

Pure equilibria

Mark best responses in a payoff matrix, apply iterated elimination of dominated strategies, and find equilibria of continuous games such as Cournot and Bertrand.

Mixed equilibria

Compute mixed equilibria by hand and in code using the indifference principle; find the minimax value of zero-sum and security games.

Existence and complexity

Interpret Nash's existence theorem, the problem of multiple equilibria, and the PPAD-completeness of computing an equilibrium.

FOUNDATIONS

Who Was John Nash?

- American mathematician **John Forbes Nash Jr.** (1928–2015).
- 1950: a short Princeton PhD thesis, *Non-Cooperative Games*, and a two-page note in *PNAS* proving existence.
- 1994 Nobel Prize in Economics (shared with Harsanyi and Selten).
- 2015 Abel Prize for his work on partial differential equations.
- His life inspired the film *A Beautiful Mind* (2001).



John F. Nash Jr.

His contribution: he extended von Neumann's solution for **two-player zero-sum** games to **general n-player** (non-zero-sum) games.

Strategic Behavior and Applications

- Each player tries to maximize their own payoff.
- But each payoff also depends on **what the others choose**: “What should I do?” cannot be separated from “What will they do?”

Price competition

Two firms cut prices; both lose profit.

`Prisoner's Dilemma structure`

Advertising / arms races

Everyone spends; relative positions stay the same.

`dominant strategy: spend`

Penalty kicks

Kicker and goalkeeper must be unpredictable.

`mixed strategy, zero-sum`

Choosing a lane

If everyone moves to the fast lane, it slows down.

`congestion game`

Choosing a standard

Everyone gains by adopting the same technology.

`coordination, multiple equilibria`

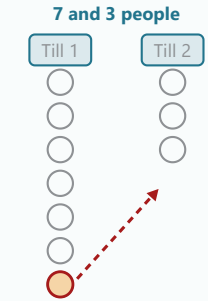
Sharing resources

Shared bandwidth, servers, roads.

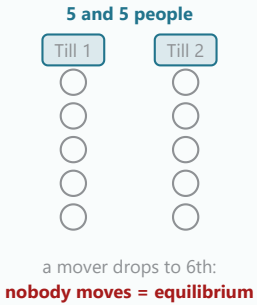
`Price of Anarchy`

FOUNDATIONS

A State Nobody Regrets



last one moves: 4th, not 7th



a mover drops to 6th:
nobody moves = equilibrium

- The cafeteria has two tills; everyone wants a shorter wait and may switch queues freely.
- **7 – 3**: the student at the back of the long queue gains by switching → this state will not last.
- **5 – 5**: whoever switches ends up further back → nobody moves.
- **Regret test:** "Had I seen what everyone else did, would I change my choice?" If everyone answers *no*, the state is a **Nash equilibrium**.

An equilibrium is self-enforcing: everyone correctly anticipates the others' behavior and does the best they can in response. It does not mean "the best outcome" or "a fair outcome".

FOUNDATIONS

Nash Equilibrium: Formal Definition

Normal-form game $G = (N, (S_i), (u_i))$:

- $N = \{1, \ldots, n\}$ players; S_i is player i's strategy set.
- Strategy profile $s = (s_1, \ldots, s_n)$; s_{-i}: the strategies of everyone except i.
- $u_i(s)$: player i's payoff (u_i is the payoff function).

Nash equilibrium: a profile s^* such that for every player i and every $s_i \in S_i$

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*)$$

Unilateral deviation

With the others fixed, no player can raise their payoff by switching strategy.

Self-enforcing agreement

If you believe the others will stick to it, you stick to it too.

Stability ≠ efficiency

An equilibrium guarantees neither fairness nor efficiency (Pareto optimality).

FOUNDATIONS

Common Mistakes: Defining Nash Equilibrium

"The equilibrium is the best outcome."

No. In the Prisoner's Dilemma the equilibrium (B, B) is worse for both players than (S, S). Equilibrium means stability, not efficiency.

"They gain if both switch together."

The definition only checks *unilateral* deviations. A joint deviation (a coalition) does not break a Nash equilibrium.

"One satisfied player is enough."

The test is run for *every* player; if even one player wants to deviate, the cell is not an equilibrium.

"The row player also picks the column."

Each player changes only their own strategy: the row player moves up and down within a column, the column player left and right within a row.

"Every game has exactly one pure equilibrium."

Matching Pennies has none; Stag Hunt has two pure equilibria and one mixed.

Cost or payoff?

For costs such as prison years, *smaller* is better. Negate them to get payoffs: 2 years → −2.

Best Response

- Player i's **best response** to the others' choice s_{-i} is the set of strategies that maximize i's payoff:

$$BR_i(s_{-i}) = \text{argmax}_{s_i \in S_i} u_i(s_i, s_{-i})$$

- Equivalently: s* is a Nash equilibrium $\Leftrightarrow$ $s_i^* \in BR_i(s_{-i}^*)$ for every i. Equilibrium = **mutual best responses**.

1. Row player

In each *column*, mark the row player's largest payoff (u_1).

2. Column player

In each *row*, mark the column player's largest payoff (u_2).

3. Intersection

Cells with both payoffs marked are the **pure-strategy Nash equilibria.**

Cost on an m × n matrix: O(mn), since each cell is compared with one column maximum and one row maximum.

Finding the Equilibrium by Marking Best Responses

Sam \ John	Silent	Betray
Silent	a (−1 , −1)	b (−3 , 0)
Betray	c (0 , −3)	d (−2 , −2)

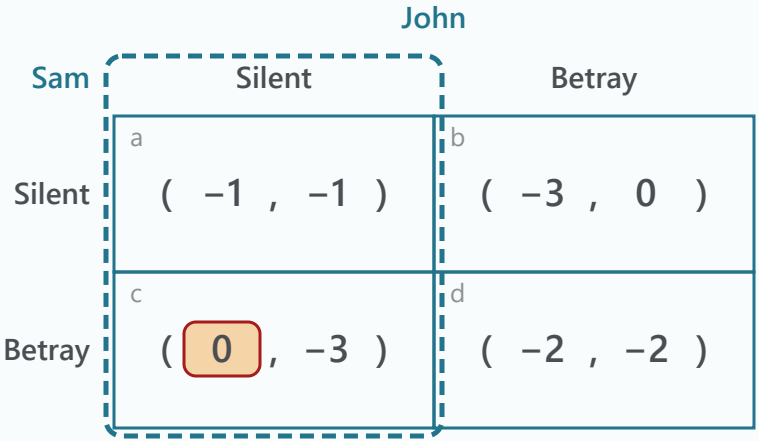
Payoff = −(years in prison)

Sam is row, John is column player

Cell: (Sam's payoff, John's payoff)

Payoffs are prison years with a minus sign (−1 = 1 year). Each player wants their own number to be larger.

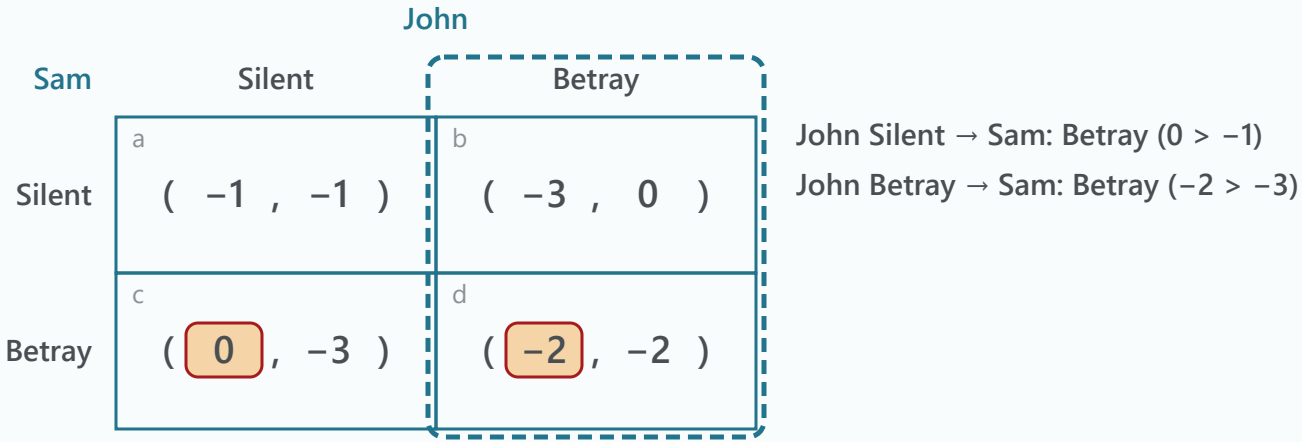
Finding the Equilibrium by Marking Best Responses



John Silent → Sam: Betray (0 > −1)

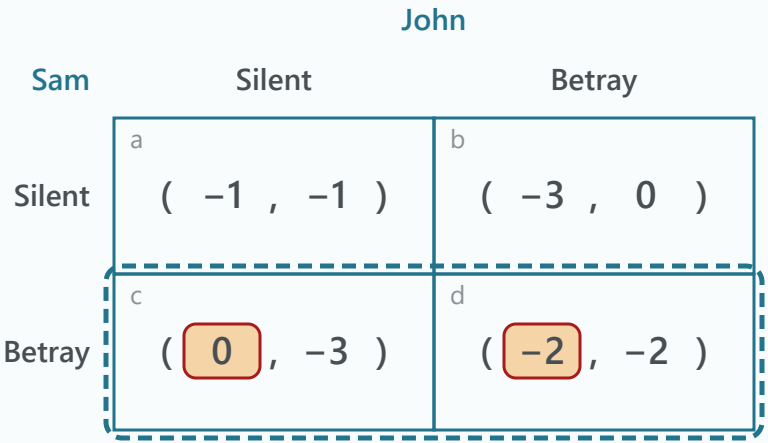
Suppose John stays **Silent**. Sam compares only the first numbers in that column: 0 > −1, so his best response is Betray.

Finding the Equilibrium by Marking Best Responses



If John **Betrays**: −2 > −3, so Sam's best response is again Betray.

Finding the Equilibrium by Marking Best Responses



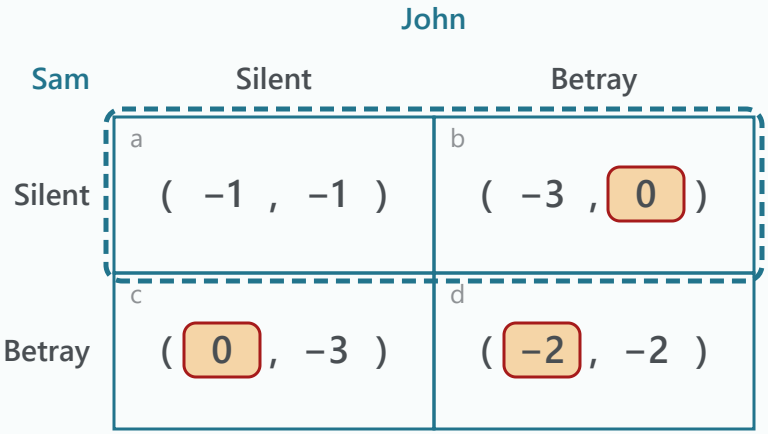
John Silent → Sam: Betray (0 > −1)

John Betray → Sam: Betray (−2 > −3)

⇒ Betray dominant for Sam

Both marks sit in the same row: whatever John does, Betray is better. It is a **dominant strategy**.

Finding the Equilibrium by Marking Best Responses



John Silent → Sam: Betray (0 > −1)

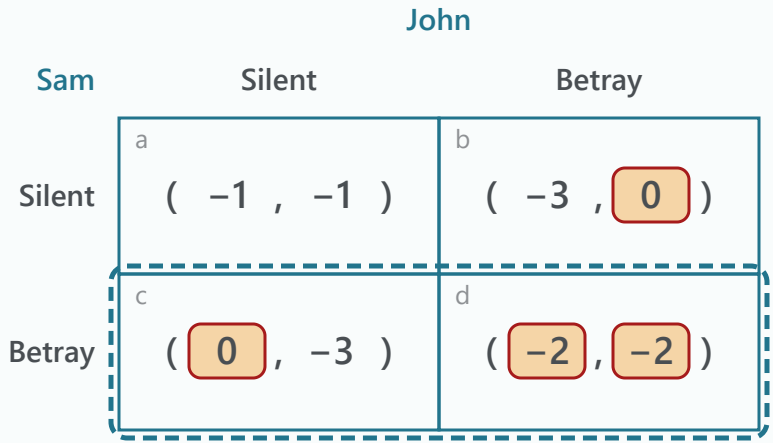
John Betray → Sam: Betray (−2 > −3)

⇒ Betray dominant for Sam

Sam Silent → John: Betray (0 > −1)

Now John. If Sam stays Silent, John compares the second numbers: 0 > −1, so Betray.

Finding the Equilibrium by Marking Best Responses



John Silent → Sam: Betray (0 > −1)

John Betray → Sam: Betray (−2 > −3)

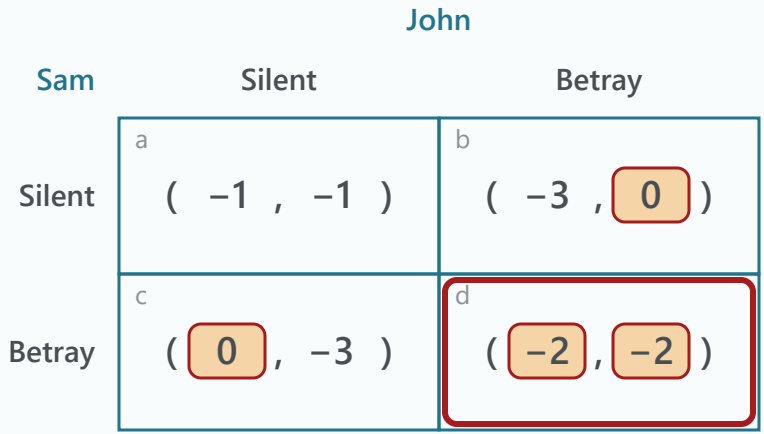
⇒ Betray dominant for Sam

Sam Silent → John: Betray (0 > −1)

Sam Betray → John: Betray (−2 > −3)

If Sam Betrays: −2 > −3, so Betray for John too. The game is symmetric, so this is no surprise.

Finding the Equilibrium by Marking Best Responses



John Silent → Sam: Betray (0 > −1)

John Betray → Sam: Betray (−2 > −3)

⇒ Betray dominant for Sam

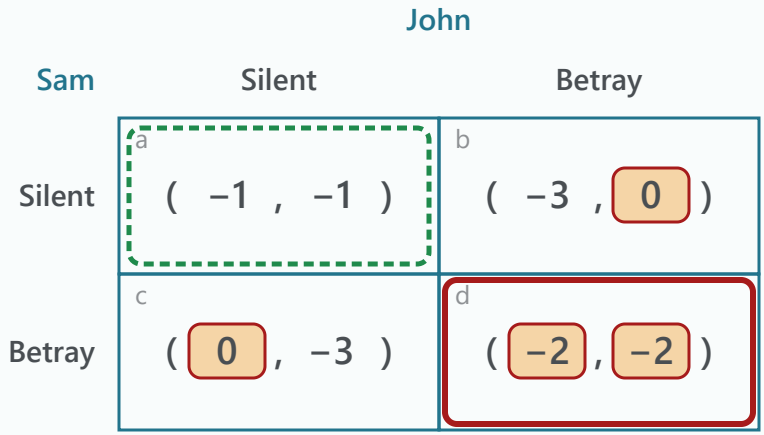
Sam Silent → John: Betray (0 > −1)

Sam Betray → John: Betray (−2 > −3)

⇒ Betray dominant for John too

The only cell with both marks is d = (Betray, Betray): nobody wants to deviate alone. **Nash equilibrium** (−2, −2).

Finding the Equilibrium by Marking Best Responses



John Silent → Sam: Betray (0 > −1)

John Betray → Sam: Betray (−2 > −3)

⇒ Betray dominant for Sam

Sam Silent → John: Betray (0 > −1)

Sam Betray → John: Betray (−2 > −3)

⇒ Betray dominant for John too

a = (−1, −1) is better for both!

Yet a = (Silent, Silent) would give each only 1 year. The equilibrium is **not Pareto optimal**: that is the dilemma.

Simple Example: Mark the Best Responses

Row player \ Column player	X	Y	Z
A	(4 , 3)	(0 , 2)	(1 , 1)
B	(2 , 1)	(3 , 3)	(0 , 2)
C	(1 , 4)	(2 , 0)	(3 , 2)

Each cell holds (u_1, u_2): the row player's payoff, then the column player's. Nothing marked yet.

Simple Example: Mark the Best Responses

Row player \ Column player	X	Y	Z
A	(4, 3)	(0, 2)	(1, 1)
B	(2, 1)	(3, 3)	(0, 2)
C	(1, 4)	(2, 0)	(3, 2)

If the column player plays **X**, the row player compares u_1: A:4, B:2, C:1 → best is **A** (4).

Simple Example: Mark the Best Responses

Row player \ Column player	X	Y	Z
A	(4, 3)	(0, 2)	(1, 1)
B	(2, 1)	(3, 3)	(0, 2)
C	(1, 4)	(2, 0)	(3, 2)

If the column player plays **Y**, the row player compares u_1: A:0, B:3, C:2 → best is **B** (3).

Simple Example: Mark the Best Responses

Row player \ Column player	X	Y	Z
A	(**4** , 3)	(0 , 2)	(1 , 1)
B	(2 , 1)	(**3** , 3)	(0 , 2)
C	(1 , 4)	(2 , 0)	(**3** , 2)

If the column player plays **Z**, the row player compares u_1: A:1, B:0, C:3 → best is **C** (3).

Simple Example: Mark the Best Responses

Row player \ Column player	X	Y	Z
A	(**4**, **3**)	(0, 2)	(1, 1)
B	(2, 1)	(**3**, 3)	(0, 2)
C	(1, 4)	(2, 0)	(**3**, 2)

If the row player plays **A**, the column player compares u_2: X:3, Y:2, Z:1 → best is **X** (3).

Simple Example: Mark the Best Responses

Row player \ Column player	X	Y	Z
A	(**4**, **3**)	(0, 2)	(1, 1)
B	(2, 1)	(**3**, **3**)	(0, 2)
C	(1, 4)	(2, 0)	(**3**, 2)

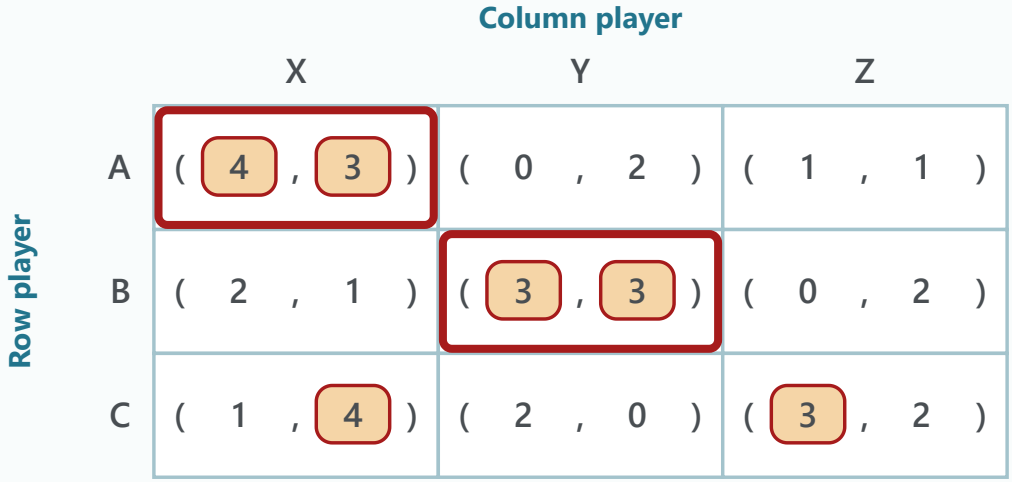
If the row player plays **B**, the column player compares u_2: X:1, Y:3, Z:2 → best is **Y** (3).

Simple Example: Mark the Best Responses

Row player \ Column player	X	Y	Z
A	(**4**, **3**)	(0, 2)	(1, 1)
B	(2, 1)	(**3**, **3**)	(0, 2)
C	(1, **4**)	(2, 0)	(**3**, 2)

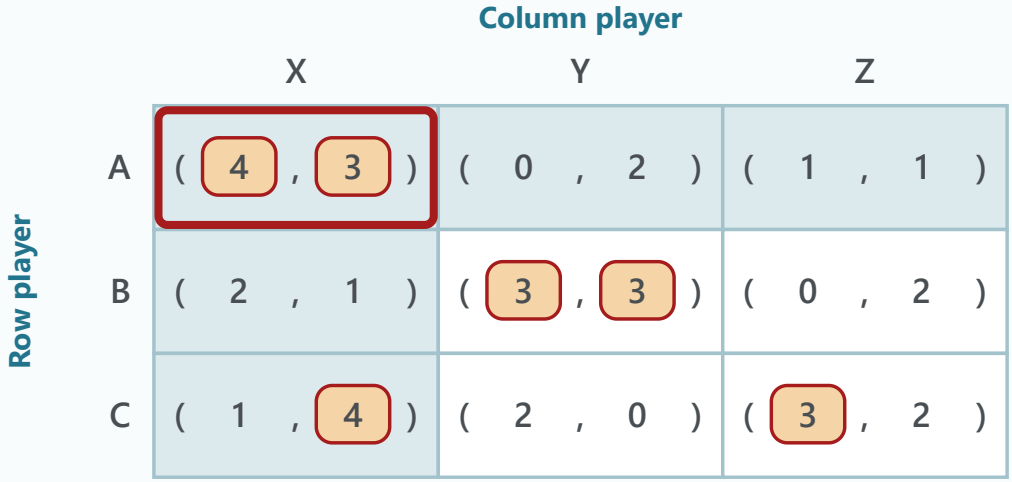
If the row player plays **C**, the column player compares u_2: X:4, Y:0, Z:2 → best is **X** (4).

Simple Example: Mark the Best Responses



Cells with both payoffs marked: **(A, X) = (4, 3)** and **(B, Y) = (3, 3)** → two pure Nash equilibria. In cells such as (C, Z), at least one player wants to deviate.

Simple Example: Mark the Best Responses



Deviation test at (A, X): if the row player switches to B or C, they get 2 or 1 (< 4); if the column player switches to Y or Z, they get 2 or 1 (< 3). Nobody gains ✓

BEST RESPONSE

Step by Step: The Underlining Method (with Ties)

Ayşe \ Bora	Python	Java	C
Python	(4, 2)	(1, 1)	(1, 0)
Java	(0, 1)	(2, 4)	(1, 2)
C	(1, 0)	(0, 2)	(1, 1)

Ayşe and Bora pick a language for a joint project; cells are (Ayşe, Bora). Rule: for Ayşe, underline the largest first number in each **column**; for Bora, the largest second number in each **row**.

BEST RESPONSE

Step by Step: The Underlining Method (with Ties)

Ayşe \ Bora	Python	Java	C
Python	(4, 2)	(1, 1)	(1, 0)
Java	(0, 1)	(2, 4)	(1, 2)
C	(1, 0)	(0, 2)	(1, 1)

If Bora picks **Python**, Ayşe looks only at the first numbers: 4, 0, 1 → underline the largest, **4** (Python).

BEST RESPONSE

Step by Step: The Underlining Method (with Ties)

Ayşe \ Bora	Python	Java	C
Python	(4, 2)	(1, 1)	(1, 0)
Java	(0, 1)	(2, 4)	(1, 2)
C	(1, 0)	(0, 2)	(1, 1)

If Bora picks **Java**: 1, 2, 0 → the largest is **2** (Java).

BEST RESPONSE

Step by Step: The Underlining Method (with Ties)

Ayşe \ Bora	Python	Java	C
Python	(4, 2)	(1, 1)	(1, 0)
Java	(0, 1)	(2, 4)	(1, 2)
C	(1, 0)	(0, 2)	(1, 1)

If Bora picks **C**: 1, 1, 1 → a three-way **tie**, so underline all three. Every tied option is a best response; marking only one is a common mistake.

BEST RESPONSE

Step by Step: The Underlining Method (with Ties)

Ayşe \ Bora	Python	Java	C
Python	(**4**, **2**)	(1, 1)	(**1**, 0)
Java	(0, 1)	(**2**, 4)	(**1**, 2)
C	(1, 0)	(0, 2)	(**1**, 1)

Now Bora. If Ayşe picks **Python**, Bora looks at the second numbers: 2, 1, 0 → **2** (Python).

BEST RESPONSE

Step by Step: The Underlining Method (with Ties)

Ayşe \ Bora	Python	Java	C
Python	(**4**, **2**)	(1, 1)	(**1**, 0)
Java	(0, 1)	(**2**, **4**)	(**1**, 2)
C	(1, 0)	(0, 2)	(**1**, 1)

If Ayşe picks **Java**: 1, 4, 2 → **4** (Java).

BEST RESPONSE

Step by Step: The Underlining Method (with Ties)

Ayşe \ Bora	Python	Java	C
Python	(4, 2)	(1, 1)	(1, 0)
Java	(0, 1)	(2, 4)	(1, 2)
C	(1, 0)	(0, 2)	(1, 1)

If Ayşe picks **C**: 0, 2, 1 → **2** (Java).

BEST RESPONSE

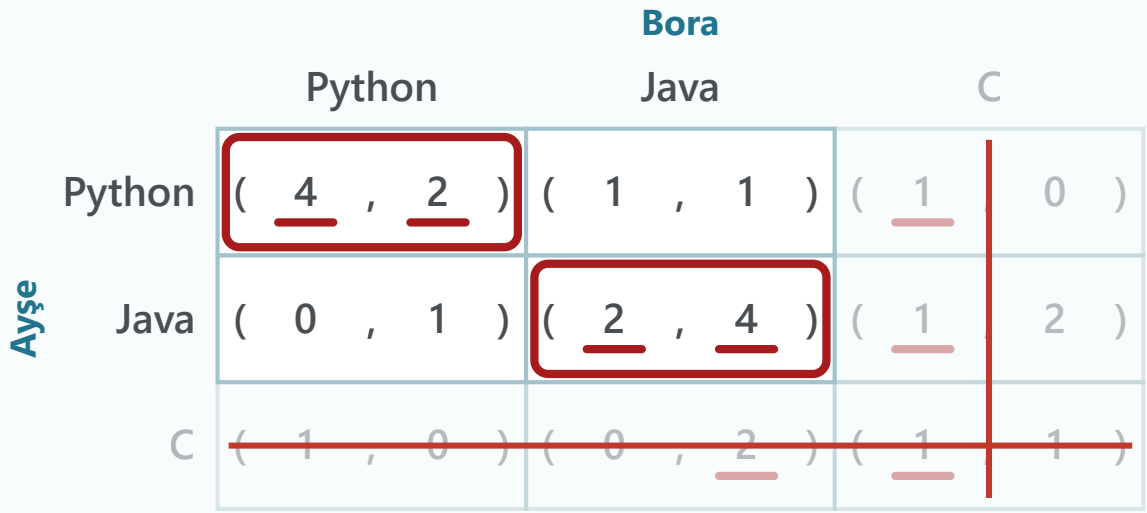
Step by Step: The Underlining Method (with Ties)

Ayşe \ Bora	Python	Java	C
Python	(4, 2)	(1, 1)	(1, 0)
Java	(0, 1)	(2, 4)	(1, 2)
C	(1, 0)	(0, 2)	(1, 1)

Cells with both numbers underlined: **(Python, Python) = (4, 2)** and **(Java, Java) = (2, 4)**. In (C, C) only the first number is underlined: Bora switches to Java and gets 2 instead of 1.

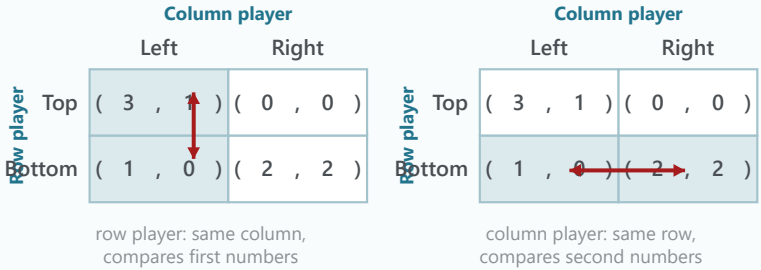
BEST RESPONSE

Step by Step: The Underlining Method (with Ties)



Strict elimination is not enough here: Java removes column C, then Python removes row C. The remaining 2×2 has no dominated strategy, so two equilibria survive. Underlining found both directly.

Common Mistakes: Marking Best Responses



Wrong direction

Searching along the row for the row player's best. They only change rows, so compare within the same column.

Wrong number

Using the first numbers for the column player. The column player wants the second number to be large.

One mark for a tie

If the maximum appears twice, both are best responses; mark both.

One mark is enough

A cell marked for only one player is not an equilibrium; it needs *two* marks.

Hunting the biggest cell

An equilibrium is not the cell with the largest numbers but the cell of mutual best responses.

Skipping the check

Run the deviation test on every candidate: does any player get more by switching?

Three Players: The Group Project Game

Three students choose **Join** or **Skip** at once. If at least two join, the project succeeds and *everyone* gets 3; joining costs 1. k: number of joiners.

Joiners	Payoffs	Unilateral deviation	Equilibrium?
k = 3 (everyone)	3 − 1 = 2 each	A quitter still gets the project: 3 > 2	No
k = 2	joiners 2, the other 3	A quitting joiner sinks it: 0 < 2; the skipper joining gets 2 < 3	**Yes** (3 profiles)
k = 1	joiner −1, others 0	The joiner quits: 0 > −1	No
k = 0 (nobody)	0 each	A lone joiner gets −1: −1 < 0	**Yes**

Four pure equilibria

The three profiles where exactly two join, plus nobody joining. With n players the test is the same: each player, others fixed.

Coordination failure

“Nobody joins” gives everyone 0, yet no one can break it alone; joint deviations are outside the Nash definition.

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria

	X	Y	Z
A	(4, 3)	(0, 2)	(1, 1)
B	(2, 1)	(3, 3)	(0, 2)
C	(1, 4)	(2, 0)	(3, 2)

U1 is the row player's payoff matrix, U2 the column player's. m = n = 3, **eq = []**.

Step 1 / 13

```
1 def pure_nash(U1, U2):
▶     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
5             br1 = max(U1[k][j] for k in range(m))
6             br2 = max(U2[i][l] for l in range(n))
7             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
9     return eq
```

VARIABLES

`m = 3` `n = 3` `eq = []`

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria

	X	Y	Z
A	(4 , 3)	(0 , 2)	(1 , 1)
B	(2 , 1)	(3 , 3)	(0 , 2)
C	(1 , 4)	(2 , 0)	(3 , 2)

Cell **(A, X)**: U1 = 4 is the column maximum and U2 = 3 the row maximum → condition **true**.

Step 2 / 13

```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
▶             br1 = max(U1[k][j] for k in range(m))
▶             br2 = max(U2[i][l] for l in range(n))
▶             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
9     return eq
```

VARIABLES

`i = 0` `j = 0` `U1[i][j] = 4` `br1 = 4` `U2[i][j] = 3` `br2 = 3` `eq = []`

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria

	X	Y	Z
A	(4, 3)	(0, 2)	(1, 1)
B	(2, 1)	(3, 3)	(0, 2)
C	(1, 4)	(2, 0)	(3, 2)

Append (0, 0): **eq = [(0, 0)]**.

Step 3 / 13

```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
5             br1 = max(U1[k][j] for k in range(m))
6             br2 = max(U2[i][l] for l in range(n))
7             if U1[i][j] == br1 and U2[i][j] == br2:
▶                 eq.append((i, j))
9     return eq
```

VARIABLES

i = 0 | j = 0 | eq = [(0, 0)]

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria

	X	Y	Z
A	(4, 3)	(0, 2)	(1, 1)
B	(2, 1)	(3, 3)	(0, 2)
C	(1, 4)	(2, 0)	(3, 2)

Cell **(A, Y)**: U1 = 0 < 3 and U2 = 2 < 3 → false, not an equilibrium.

Step 4 / 13

```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
▶             br1 = max(U1[k][j] for k in range(m))
▶             br2 = max(U2[i][l] for l in range(n))
▶             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
9     return eq
```

VARIABLES

`i = 0` `j = 1` `U1[i][j] = 0` `br1 = 3` `U2[i][j] = 2` `br2 = 3` `eq = [(0, 0)]`

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria

	X	Y	Z
A	(4, 3)	(0, 2)	(1, 1)
B	(2, 1)	(3, 3)	(0, 2)
C	(1, 4)	(2, 0)	(3, 2)

Cell **(A, Z)**: U1 = 1 < 3 and U2 = 1 < 3 → false, not an equilibrium.

Step 5 / 13

```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
▶             br1 = max(U1[k][j] for k in range(m))
▶             br2 = max(U2[i][l] for l in range(n))
▶             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
9     return eq
```

VARIABLES

`i = 0` `j = 2` `U1[i][j] = 1` `br1 = 3` `U2[i][j] = 1` `br2 = 3` `eq = [(0, 0)]`

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria

	X	Y	Z
A	(4 , 3)	(0 , 2)	(1 , 1)
B	(2 , 1)	(3 , 3)	(0 , 2)
C	(1 , 4)	(2 , 0)	(3 , 2)

Cell **(B, X)**: U1 = 2 < 4 and U2 = 1 < 3 → false, not an equilibrium.

Step 6 / 13

```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
▶             br1 = max(U1[k][j] for k in range(m))
▶             br2 = max(U2[i][l] for l in range(n))
▶             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
9     return eq
```

VARIABLES

i = 1 | j = 0 | U1[i][j] = 2 | br1 = 4 | U2[i][j] = 1 | br2 = 3 | eq = [(0, 0)]

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria

	X	Y	Z
A	(4 , 3)	(0 , 2)	(1 , 1)
B	(2 , 1)	(3 , 3)	(0 , 2)
C	(1 , 4)	(2 , 0)	(3 , 2)

Cell **(B, Y)**: U1 = 3 is the column maximum and U2 = 3 the row maximum → condition **true**.

Step 7 / 13

```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
▶             br1 = max(U1[k][j] for k in range(m))
▶             br2 = max(U2[i][l] for l in range(n))
▶             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
9     return eq
```

VARIABLES

`i = 1` `j = 1` `U1[i][j] = 3` `br1 = 3` `U2[i][j] = 3` `br2 = 3` `eq = [(0, 0)]`

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria

	X	Y	Z
A	(4, 3)	(0, 2)	(1, 1)
B	(2, 1)	(3, 3)	(0, 2)
C	(1, 4)	(2, 0)	(3, 2)

Append (1, 1): **eq = [(0, 0), (1, 1)]**.

Step 8 / 13

```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
5             br1 = max(U1[k][j] for k in range(m))
6             br2 = max(U2[i][l] for l in range(n))
7             if U1[i][j] == br1 and U2[i][j] == br2:
▶                 eq.append((i, j))
9     return eq
```

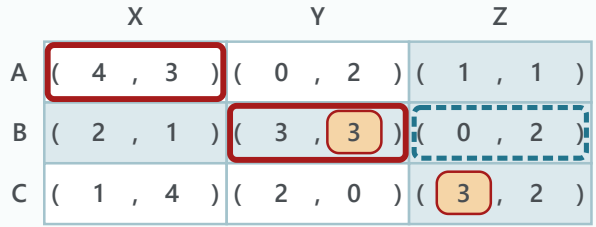
VARIABLES

i = 1 j = 1

eq = [(0, 0), (1, 1)]

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria



Cell **(B, Z)**: U1 = 0 < 3 and U2 = 2 < 3 → false, not an equilibrium.

Step 9 / 13

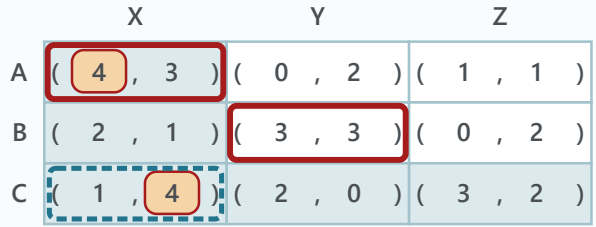
```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
▶             br1 = max(U1[k][j] for k in range(m))
▶             br2 = max(U2[i][l] for l in range(n))
▶             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
9     return eq
```

VARIABLES

`i = 1` `j = 2` `U1[i][j] = 0` `br1 = 3` `U2[i][j] = 2` `br2 = 3` `eq = [(0, 0), (1, 1)]`

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria



Cell **(C, X)**: U1 = 1 < 4 → false, not an equilibrium.

Step 10 / 13

```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
▶             br1 = max(U1[k][j] for k in range(m))
▶             br2 = max(U2[i][l] for l in range(n))
▶             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
9     return eq
```

VARIABLES

`i = 2` `j = 0` `U1[i][j] = 1` `br1 = 4` `U2[i][j] = 4` `br2 = 4` `eq = [(0, 0), (1, 1)]`

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria

	X	Y	Z
A	(4 , 3)	(0 , 2)	(1 , 1)
B	(2 , 1)	(3 , 3)	(0 , 2)
C	(1 , 4)	(2 , 0)	(3 , 2)

Cell **(C, Y)**: U1 = 2 < 3 and U2 = 0 < 4 → false, not an equilibrium.

Step 11 / 13

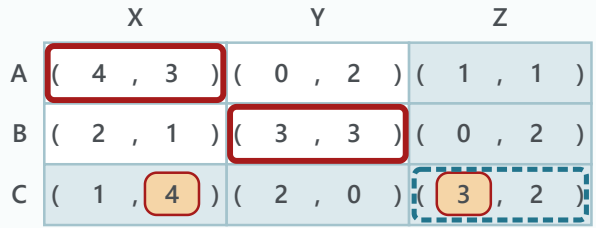
```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
▶             br1 = max(U1[k][j] for k in range(m))
▶             br2 = max(U2[i][l] for l in range(n))
▶             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
9     return eq
```

VARIABLES

`i = 2` `j = 1` `U1[i][j] = 2` `br1 = 3` `U2[i][j] = 0` `br2 = 4` `eq = [(0, 0), (1, 1)]`

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria



Cell **(C, Z)**: U2 = 2 < 4 → false, not an equilibrium.

Step 12 / 13

```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
▶             br1 = max(U1[k][j] for k in range(m))
▶             br2 = max(U2[i][l] for l in range(n))
▶             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
9     return eq
```

VARIABLES

`i = 2` `j = 2` `U1[i][j] = 3`

`br1 = 3` `U2[i][j] = 2` `br2 = 4`

`eq = [(0, 0), (1, 1)]`

BEST RESPONSE

Code Trace: Finding Pure Nash Equilibria

	X	Y	Z
A	(4, 3)	(0, 2)	(1, 1)
B	(2, 1)	(3, 3)	(0, 2)
C	(1, 4)	(2, 0)	(3, 2)

All 9 cells scanned: **return [(0, 0), (1, 1)]** → (A, X) and (B, Y). Total work O(mn·(m+n)); O(mn) if the maxima are precomputed.

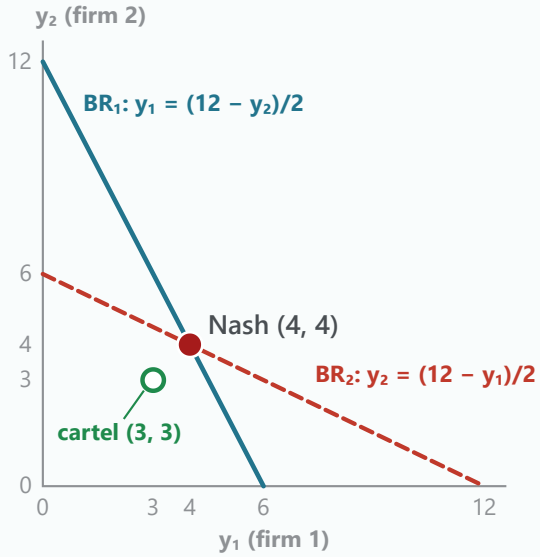
Step 13 / 13

```
1 def pure_nash(U1, U2):
2     m, n, eq = len(U1), len(U1[0]), []
3     for i in range(m):
4         for j in range(n):
5             br1 = max(U1[k][j] for k in range(m))
6             br2 = max(U2[i][l] for l in range(n))
7             if U1[i][j] == br1 and U2[i][j] == br2:
8                 eq.append((i, j))
▶     return eq
```

VARIABLES

eq = [(0, 0), (1, 1)]

Continuous Strategies: Cournot Duopoly



- Two firms simultaneously choose quantities $y_1, y_2 \geq 0$ (an infinite strategy set). Price $P = 12 - (y_1 + y_2)$, zero cost.
- Profit $u_1 = y_1 (12 - y_1 - y_2)$. Setting the derivative $12 - 2y_1 - y_2$ to 0 $\Rightarrow$ **$BR_1(y_2) = \max\{0, (12 - y_2)/2\}$** (quantities cannot be negative; if $y_2 \geq 12$, it is best to produce nothing).
- Nash equilibrium = where the best-response lines cross: **$y_1 = y_2 = 4$**, price 4, profit 16 each.
- A cartel (3 each) gives both 18, but it is not an equilibrium: $BR_1(3) = 4.5$ and the deviating firm earns 20.25.

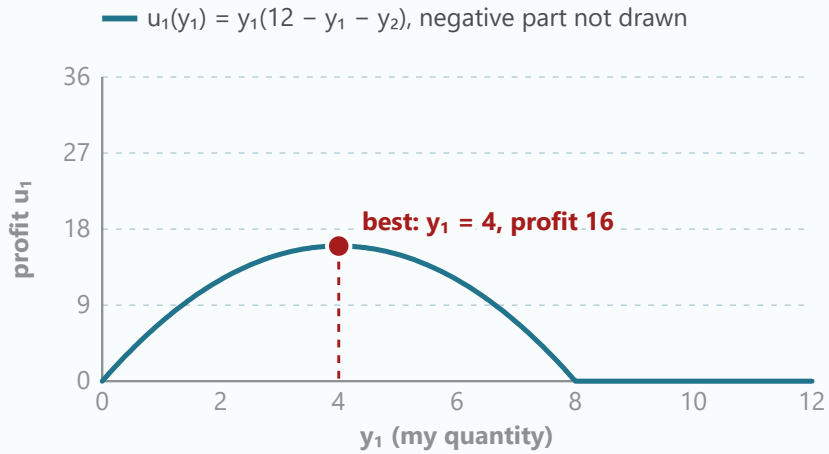
If the firms take turns best-responding, output moves (6, 3), then (4.5, 3.75), … and approaches (4, 4). Cournot, 1838.

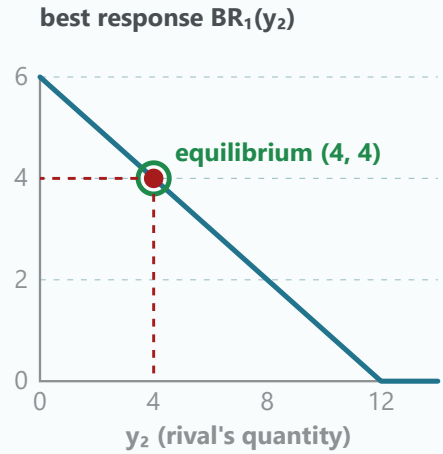
How Much Should I Produce Given My Rival's Output?

$$u_1(y_1) = y_1(12 - y_1 - y_2) \Rightarrow \mathbf{BR_1(y_2) = \max\{0, (12 - y_2)/2\}}$$

y_2 = 4

$BR_1(4) = \max\{0, (12 - 4)/2\} = \mathbf{4}$ | price = **4** | best profit = **16**





The more my rival produces, the less I should produce; once $y_2 \geq 12$, producing nothing is best. The only point where both firms best-respond to each other is $\mathbf{y_1 = y_2 = 4}$.

BEST RESPONSE

Bertrand: Competing on Price Instead of Quantity

- Two firms sell the same good; unit cost c = 2 TL, market size 100 units. They choose **prices simultaneously**: the cheaper firm takes the whole market, and equal prices split it in half.
- At (5, 5) each earns (5 − 2) · 50 = 150. Undercutting to 4.99 yields ≈ 2.99 · 100 ≈ 299, so deviating pays. The same holds at *every* common price above c.
- Pricing below c loses money. The unique Nash equilibrium is $\mathbf{p_1 = p_2 = c = 2}$, with zero profit.



Competing on	Equilibrium	Firm profit
Cournot (quantity), P = 12 − Y, zero cost	y = 4 each, price 4	16 each
Bertrand (price), cost c	p = c	0

Common mistake: "With two firms, profits stay high." Under price competition even two firms drive price down to cost (the **Bertrand paradox**); profit survives only through product differentiation, capacity limits or repeated interaction. This holds for continuous prices; with 1 TL steps, prices just above c can also be equilibria.

Dominant and Dominated Strategies

Strict dominance

s_i' strictly dominates s_i if $u_i(s_i', s_{-i}) > u_i(s_i, s_{-i})$ for every s_{-i}.

`A rational player never plays a strictly dominated strategy.`

Weak dominance

$\geq$ for every s_{-i} and $>$ for at least one.

`Deleting it may lose equilibria; order can matter.`

- A **dominant strategy** dominates all of the player's other strategies (Betray in the Prisoner's Dilemma); no need to predict the opponent. If every player has one, that profile is a **dominant-strategy equilibrium**. Not every game has one (RPS).
- **Iterated elimination (IESDS)**: delete strictly dominated strategies and repeat. Assumes rationality is common knowledge.
- Under strict dominance **order does not matter** and no equilibrium is lost. A single remaining cell is the **unique** Nash equilibrium.
- Test against each opposing choice; “better on average” is not enough. In the 2/3 game, numbers above 60 sometimes tie with 60: that is **weak** dominance.

Back to the Game: The Prisoner's Dilemma

Sam \ John	Silent	Betray
Silent	(−1 , −1)	(−3 , 0)
Betray	(0 , −3)	(−2 , −2)

Payoff = −years: each player's best response marked

- **Silent or betray? Theory's answer:** Betray is dominant: if John is silent, 0 > −1; if he betrays, −2 > −3. The unique Nash equilibrium is (Betray, Betray): (−2, −2), two years each.
- **In practice:** in one-shot play people cooperate far more often than theory predicts. A meta-analysis of experiments from 1958–1992 finds roughly half of participants cooperate, and communication raises that rate markedly (Sally, 1995). Trust, fairness and reciprocity also matter.

Sam's 10-round plan against John's Tit-for-Tat	Sam's total years
Always Silent	10 × 1 = 10
Always Betray	0 + 9 × 2 = 18
Silent 9 rounds, Betray the last	9 × 1 + 0 = **9** (minimum)

In the repeated game, betrayal is punished in later rounds; only the last round goes unpunished. This “last-round effect” is what backward induction is about.

Iterated Elimination: A 3×3 Example

Row player \ Column player	X	Y	Z
A	(3, 2)	(3, 3)	(1, 4)
B	(3, 2)	(0, 4)	(5, 3)
C	(2, 1)	(2, 0)	(6, 6)

Start: no row strictly dominates another (e.g. A vs B: 3 = 3). Look at the column player first.

Iterated Elimination: A 3×3 Example

Row player \ Column player	X	Y	Z
A	(3, 2)	(3, 3)	(1, 4)
B	(3, 2)	(0, 4)	(5, 3)
C	(2, 1)	(2, 0)	(6, 6)

Over the remaining rows, **Z** strictly dominates **X** (u_2: 4 > 2, 3 > 2, 6 > 1) → delete X.

Iterated Elimination: A 3×3 Example

Row player \ Column player	X	Y	Z
A	(3, 2)	(3, 3)	(1, 4)
B	(3, 2)	(0, 4)	(5, 3)
C	(2, 1)	(2, 0)	(6, 6)

Over the remaining columns, **C** strictly dominates **B** (u_1: 2 > 0, 6 > 5) → delete B.

Iterated Elimination: A 3×3 Example

Row player \ Column player	X	Y	Z
A	(3, 2)	(3, 3)	(1, 4)
B	(3, 2)	(0, 4)	(5, 3)
C	(2, 1)	(2, 0)	(6, 6)

Over the remaining rows, **Z** strictly dominates **Y** (u_2: 4 > 3, 6 > 0) → delete Y.

Iterated Elimination: A 3×3 Example

Row player \ Column player	X	Y	Z
A	(3, 2)	(3, 3)	(1, 4)
B	(3, 2)	(0, 4)	(5, 3)
C	(2, 1)	(2, 0)	(6, 6)

Over the remaining columns, **C** strictly dominates **A** (u_1: 6 > 1) → delete A.

Iterated Elimination: A 3×3 Example

Row player \ Column player	X	Y	Z
A	(3, 2)	(3, 3)	(1, 4)
B	(3, 2)	(0, 4)	(5, 3)
C	(2, 1)	(2, 0)	**(6, 6)**

One cell remains: **(C, Z) = (6, 6)**. It is the game's unique Nash equilibrium (marking best responses gives the same answer).

Step by Step: Each Elimination Enables the Next

Row player \ Column player	Left	Center	Right
Top	(1 , 5)	(5 , 6)	(6 , 2)
Middle	(3 , 5)	(6 , 0)	(2 , 2)
Bottom	(2 , 2)	(0 , 2)	(1 , 5)

Neither player has a dominant strategy, and at first there is only one strict dominance. Compare rows using only the **first** numbers, columns using only the **second**.

Step by Step: Each Elimination Enables the Next

Row player \ Column player	Left	Center	Right
Top	(1, 5)	(5, 6)	(6, 2)
Middle	(3, 5)	(6, 0)	(2, 2)
Bottom	(2, 2)	(0, 2)	(1, 5)

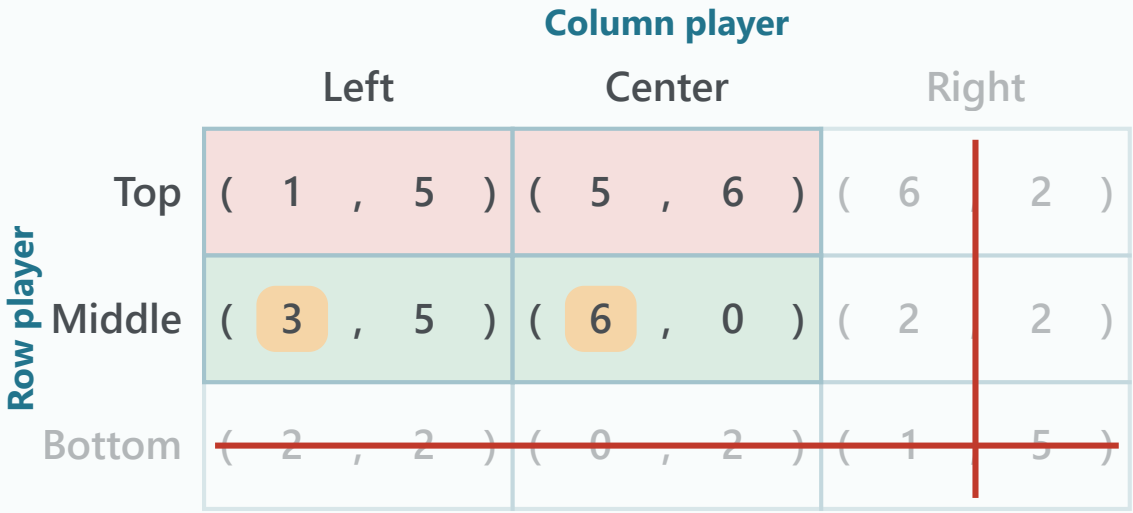
Middle beats Bottom in every column: 3 > 2, 6 > 0, 2 > 1 → **delete Bottom**.

Step by Step: Each Elimination Enables the Next

Row player \ Column player	Left	Center	Right
Top	(1 , 5)	(5 , 6)	(6 , 2)
Middle	(3 , 5)	(6 , 0)	(2 , 2)
~~Bottom~~	~~(2 , 2)~~	~~(0 , 2)~~	~~(1 , 5)~~

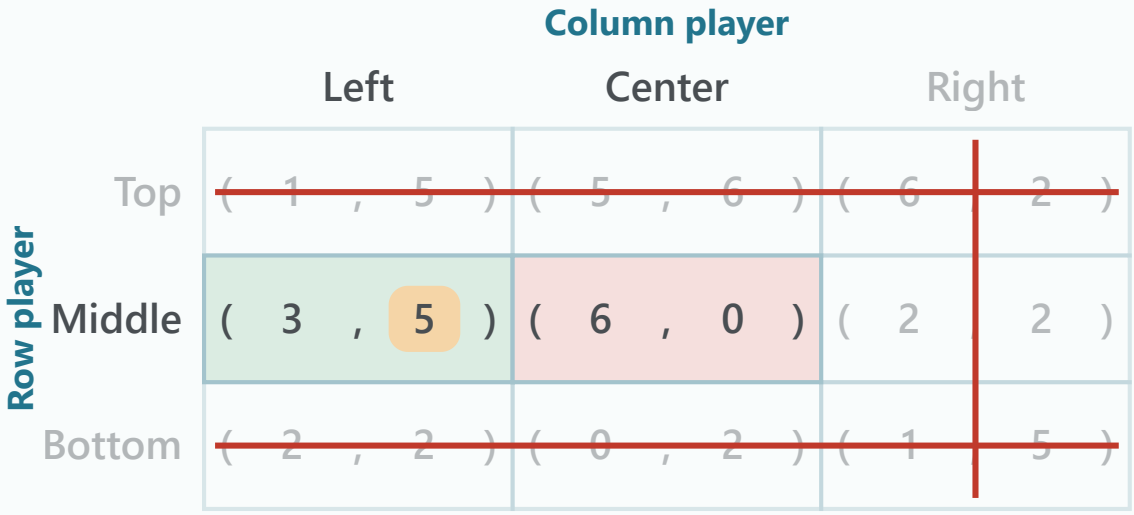
In the remaining rows, Left beats Right: 5 > 2 (Top), 5 > 2 (Middle) → **delete Right**. Not possible while Bottom remained: there Right 5 > Left 2.

Step by Step: Each Elimination Enables the Next



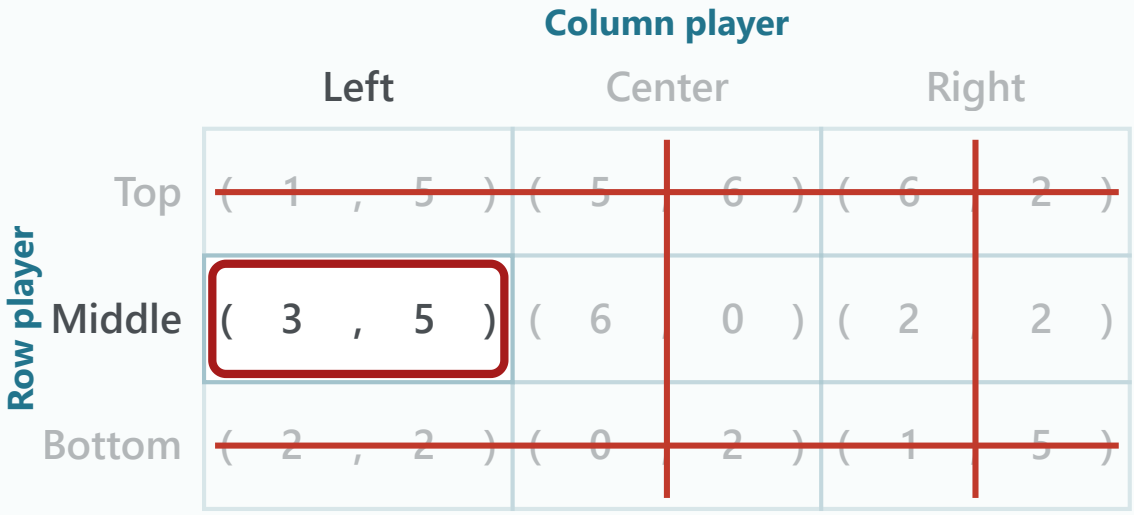
In the remaining columns, Middle beats Top: 3 > 1, 6 > 5 → **delete Top**. While Right remained, Top survived there with 6 > 2.

Step by Step: Each Elimination Enables the Next



Only the Middle row is left: Left 5, Center 0 → **delete Center**.

Step by Step: Each Elimination Enables the Next



The survivor **(Middle, Left) = (3, 5)** is the unique Nash equilibrium. (Top, Center) = (5, 6) is better for both but not an equilibrium: the row player switches to Middle and gets 6.

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3 , 2)	(3 , 3)	(1 , 4)
B	(3 , 2)	(0 , 4)	(5 , 3)
C	(2 , 1)	(2 , 0)	(6 , 6)

Remaining rows and columns: **R = [0, 1, 2]** (A, B, C), **S = [0, 1, 2]** (X, Y, Z).

Step 1 / 11

```
1  def eliminate(U1, U2):
▶      R, S = [0, 1, 2], [0, 1, 2]   # remaining
3      while True:
4          x = [r for r in R for k in R if
5               all(U1[k][c] > U1[r][c] for c in S)]
6          y = [c for c in S for d in S if
7               all(U2[r][d] > U2[r][c] for r in R)]
8          if x: R.remove(x[0])      # drop a row
9          elif y: S.remove(y[0])    # drop a column
10         else: return R, S
```

VARIABLES

R = [0, 1, 2] S = [0, 1, 2]

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3, 2)	(3, 3)	(1, 4)
B	(3, 2)	(0, 4)	(5, 3)
C	(2, 1)	(2, 0)	(6, 6)

No row is dominated (x = []). Column X is dominated by Z → **y = [0]**.

Step 2 / 11

```
1  def eliminate(U1, U2):
2      R, S = [0, 1, 2], [0, 1, 2]   # remaining
3      while True:
▶          x = [r for r in R for k in R if
▶               all(U1[k][c] > U1[r][c] for c in S)]
▶          y = [c for c in S for d in S if
▶               all(U2[r][d] > U2[r][c] for r in R)]
8          if x: R.remove(x[0])      # drop a row
9          elif y: S.remove(y[0])    # drop a column
10         else: return R, S
```

VARIABLES

R = [0, 1, 2] S = [0, 1, 2]

x = [] y = [0]

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3 , 2)	(3 , 3)	(1 , 4)
B	(3 , 2)	(0 , 4)	(5 , 3)
C	(2 , 1)	(2 , 0)	(6 , 6)

S.remove(0): delete column X; the next round rechecks the rows.

Step 3 / 11

```
1  def eliminate(U1, U2):
2      R, S = [0, 1, 2], [0, 1, 2]   # remaining
3      while True:
4          x = [r for r in R for k in R if
5               all(U1[k][c] > U1[r][c] for c in S)]
6          y = [c for c in S for d in S if
7               all(U2[r][d] > U2[r][c] for r in R)]
8          if x: R.remove(x[0])      # drop a row
▶          elif y: S.remove(y[0])    # drop a column
10         else: return R, S
```

VARIABLES

R = [0, 1, 2] S = [1, 2] x = [] y = [0]

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3 , 2)	(3 , 3)	(1 , 4)
B	(3 , 2)	(0 , 4)	(5 , 3)
C	(2 , 1)	(2 , 0)	(6 , 6)

Row B is dominated by C → **x = [1]**. Columns are checked too: y = [].

Step 4 / 11

```
1  def eliminate(U1, U2):
2      R, S = [0, 1, 2], [0, 1, 2]   # remaining
3      while True:
▶          x = [r for r in R for k in R if
▶               all(U1[k][c] > U1[r][c] for c in S)]
▶          y = [c for c in S for d in S if
▶               all(U2[r][d] > U2[r][c] for r in R)]
8          if x: R.remove(x[0])      # drop a row
9          elif y: S.remove(y[0])    # drop a column
10         else: return R, S
```

VARIABLES

R = [0, 1, 2] S = [1, 2] x = [1] y = []

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3 , 2)	(3 , 3)	(1 , 4)
B	(3 , 2)	(0 , 4)	(5 , 3)
C	(2 , 1)	(2 , 0)	(6 , 6)

R.remove(1): delete row B; the loop restarts on the smaller game.

Step 5 / 11

```
1  def eliminate(U1, U2):
2      R, S = [0, 1, 2], [0, 1, 2]   # remaining
3      while True:
4          x = [r for r in R for k in R if
5               all(U1[k][c] > U1[r][c] for c in S)]
6          y = [c for c in S for d in S if
7               all(U2[r][d] > U2[r][c] for r in R)]
▶          if x: R.remove(x[0])      # drop a row
9          elif y: S.remove(y[0])    # drop a column
10         else: return R, S
```

VARIABLES

R = [0, 2] S = [1, 2] x = [1] y = []

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3 , 2)	(3 , 3)	(1 , 4)
B	(3 , 2)	(0 , 4)	(5 , 3)
C	(2 , 1)	(2 , 0)	(6 , 6)

No row is dominated (x = []). Column Y is dominated by Z → **y = [1]**.

Step 6 / 11

```
1  def eliminate(U1, U2):
2      R, S = [0, 1, 2], [0, 1, 2]   # remaining
3      while True:
▶          x = [r for r in R for k in R if
▶               all(U1[k][c] > U1[r][c] for c in S)]
▶          y = [c for c in S for d in S if
▶               all(U2[r][d] > U2[r][c] for r in R)]
8          if x: R.remove(x[0])      # drop a row
9          elif y: S.remove(y[0])    # drop a column
10         else: return R, S
```

VARIABLES

R = [0, 2] S = [1, 2] x = [] y = [1]

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3, 2)	(3, 3)	(1, 4)
B	(3, 2)	(0, 4)	(5, 3)
C	(2, 1)	(2, 0)	(6, 6)

S.remove(1): delete column Y; the next round rechecks the rows.

Step 7 / 11

```
1  def eliminate(U1, U2):
2      R, S = [0, 1, 2], [0, 1, 2]   # remaining
3      while True:
4          x = [r for r in R for k in R if
5               all(U1[k][c] > U1[r][c] for c in S)]
6          y = [c for c in S for d in S if
7               all(U2[r][d] > U2[r][c] for r in R)]
8          if x: R.remove(x[0])      # drop a row
▶          elif y: S.remove(y[0])    # drop a column
10         else: return R, S
```

VARIABLES

R = [0, 2] S = [2] x = [] y = [1]

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3 , 2)	(3 , 3)	(1 , 4)
B	(3 , 2)	(0 , 4)	(5 , 3)
C	(2 , 1)	(2 , 0)	(6 , 6)

Row A is dominated by C → **x = [0]**. Columns are checked too: y = [].

Step 8 / 11

```
1  def eliminate(U1, U2):
2      R, S = [0, 1, 2], [0, 1, 2]   # remaining
3      while True:
▶          x = [r for r in R for k in R if
▶               all(U1[k][c] > U1[r][c] for c in S)]
▶          y = [c for c in S for d in S if
▶               all(U2[r][d] > U2[r][c] for r in R)]
8          if x: R.remove(x[0])      # drop a row
9          elif y: S.remove(y[0])    # drop a column
10         else: return R, S
```

VARIABLES

R = [0, 2] S = [2] x = [0] y = []

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3 , 2)	(3 , 3)	(1 , 4)
B	(3 , 2)	(0 , 4)	(5 , 3)
C	(2 , 1)	(2 , 0)	(6 , 6)

R.remove(0): delete row A; the loop restarts on the smaller game.

Step 9 / 11

```
1  def eliminate(U1, U2):
2      R, S = [0, 1, 2], [0, 1, 2]   # remaining
3      while True:
4          x = [r for r in R for k in R if
5               all(U1[k][c] > U1[r][c] for c in S)]
6          y = [c for c in S for d in S if
7               all(U2[r][d] > U2[r][c] for r in R)]
▶          if x: R.remove(x[0])      # drop a row
9          elif y: S.remove(y[0])    # drop a column
10         else: return R, S
```

VARIABLES

R = [2] S = [2] x = [0] y = []

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3 , 2)	(3 , 3)	(1 , 4)
B	(3 , 2)	(0 , 4)	(5 , 3)
C	(2 , 1)	(2 , 0)	(6 , 6)

No row or column is dominated: **x = y = []**.

Step 10 / 11

```
1  def eliminate(U1, U2):
2      R, S = [0, 1, 2], [0, 1, 2]   # remaining
3      while True:
▶          x = [r for r in R for k in R if
▶               all(U1[k][c] > U1[r][c] for c in S)]
▶          y = [c for c in S for d in S if
▶               all(U2[r][d] > U2[r][c] for r in R)]
8          if x: R.remove(x[0])      # drop a row
9          elif y: S.remove(y[0])    # drop a column
10         else: return R, S
```

VARIABLES

R = [2] | S = [2] | x = [] | y = []

Code Trace: Iterated Elimination of Dominated Strategies

	X	Y	Z
A	(3, 2)	(3, 3)	(1, 4)
B	(3, 2)	(0, 4)	(5, 3)
C	(2, 1)	(2, 0)	**(6, 6)**

return [2], [2] → only (C, Z) is left. Four eliminations removed X, B, Y, A in turn.

Step 11 / 11

```
1 def eliminate(U1, U2):
2     R, S = [0, 1, 2], [0, 1, 2]   # remaining
3     while True:
4         x = [r for r in R for k in R if
5              all(U1[k][c] > U1[r][c] for c in S)]
6         y = [c for c in S for d in S if
7              all(U2[r][d] > U2[r][c] for r in R)]
8         if x: R.remove(x[0])      # drop a row
9         elif y: S.remove(y[0])    # drop a column
▶         else: return R, S
```

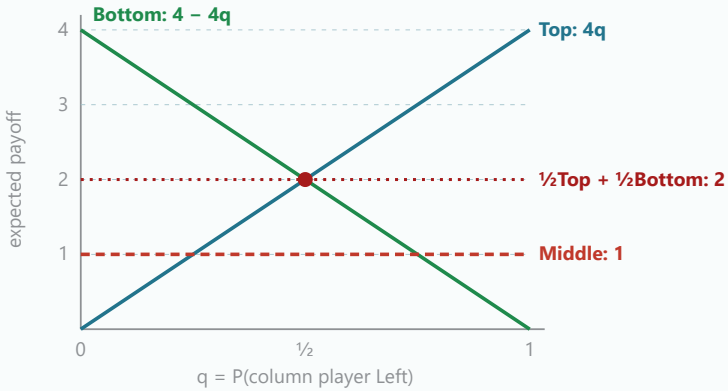
VARIABLES

`R = [2]` `S = [2]` `x = []` `y = []`

Dominance by a Mixed Strategy

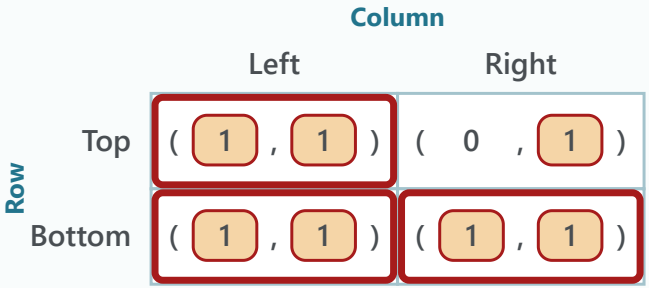
u_1	Left	Right
Top	4	0
Middle	1	1
Bottom	0	4

- No pure row dominates Middle: it beats Top under Right (1 > 0) and Bottom under Left (1 > 0).
- Mixing Top and Bottom ½–½ gives the row player 2 in every column: 2 > 1 and 2 > 1 → Middle is **strictly dominated by a mixed strategy** and can be deleted.
- In the plot, Middle's line (1) lies below the upper envelope of Top and Bottom (at least 2) for every q: Middle is a best response to no belief.



Common mistake: “No single pure strategy beats it, so it cannot be deleted.” Mixtures count too; when eliminating, try pure strategies first, then simple mixtures such as ½–½.

The Trap of Weak Dominance



- For the row player, Bottom **weakly** dominates Top (1 = 1, 1 > 0). The column player is always indifferent.
- Pure Nash equilibria: **(Top, Left), (Bottom, Left), (Bottom, Right)**, three in all.
- Deleting Top loses the (Top, Left) equilibrium; different elimination orders can leave different results.
- Lesson: strict dominance is safe; weak dominance **can lose equilibria**.

Strict elimination does not always shrink a game to one cell: in Battle of the Sexes nothing is dominated. Then we turn to best-response marking and mixed strategies.

Common Mistakes: Eliminating Dominated Strategies

Wrong numbers

Comparing two rows using the column player's payoffs. Row ↔ first number, column ↔ second number.

"Mostly bigger" is enough

Strict dominance needs bigger in *every* remaining column; a single tie or loss stops the elimination.

Counting deleted strategies

A deleted column no longer takes part. In an elimination chain, a strategy (e.g. Top) may become removable only after another (Right) is gone.

Mistaking weak for strict

With a tie, the dominance is weak; deleting may lose equilibria and the result may depend on the order.

Dominant ≠ dominated

A dominant strategy beats *all* others; a dominated one loses to at least *one*. Elimination works even without a dominant strategy.

Every survivor is an equilibrium

If elimination leaves several cells (e.g. a 2×2 in the language game), find the equilibria of the reduced game by underlining.

Mixed Strategies and the Indifference Principle

- A **mixed strategy** assigns a probability to each pure strategy (e.g. Rock 1/3, Paper 1/3, Scissors 1/3). Payoffs become expected payoffs. In 2×2 games, $p = \sigma_1$(first row), $q = \sigma_2$(first column).
- In games with no pure equilibrium (Matching Pennies, RPS, penalty kicks, inspection, bluffing), players want to be **unpredictable**. The RPS equilibrium is (1/3, 1/3, 1/3): every action earns 0. Any other mix can be exploited.
- **Support**: the pure strategies played with positive probability.

Indifference principle: in equilibrium, each player earns the *same* expected payoff from every pure strategy in their support (strategies outside it earn no more). Otherwise they would shift all probability to the better one.

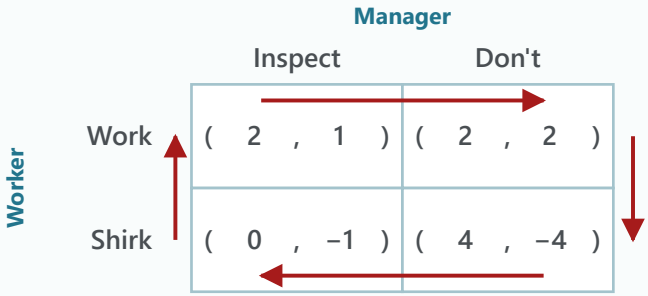
A surprise

The row player's mix is computed from the *column player*'s payoffs (and vice versa).

Interpretation

Probabilities can be read as “rolling dice” or as behavior shares in a large population (e.g. directions of penalty kicks).

Computing a Mixed Equilibrium: The Inspection Game



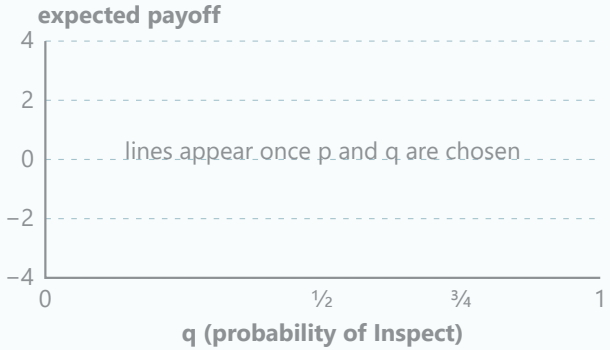
Inspection game

Row: worker, column: manager. Wage 4, effort cost 2, output 6, inspection cost 1. A shirker who is caught gets no wage. Example: in Work–Inspect the worker gets $4 - 2 = 2$ and the manager $6 - 4 - 1 = 1$.

Arrows: in every cell some player gains by deviating → best responses form a cycle, **no pure Nash equilibrium**.

Computing a Mixed Equilibrium: The Inspection Game

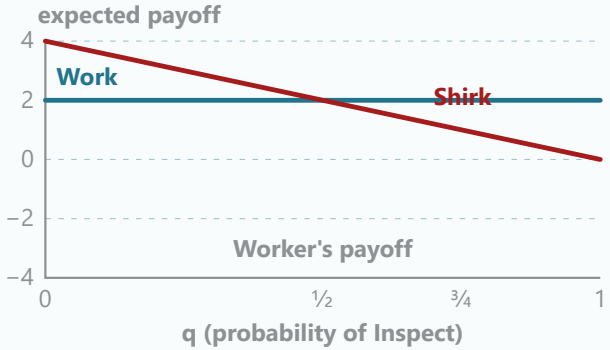
	Inspect	Don't
Work	(2 , 1)	(2 , 2)
Shirk	(0 , −1)	(4 , −4)



1. The worker plays Work with probability **p**, the manager Inspect with probability **q**.

Computing a Mixed Equilibrium: The Inspection Game

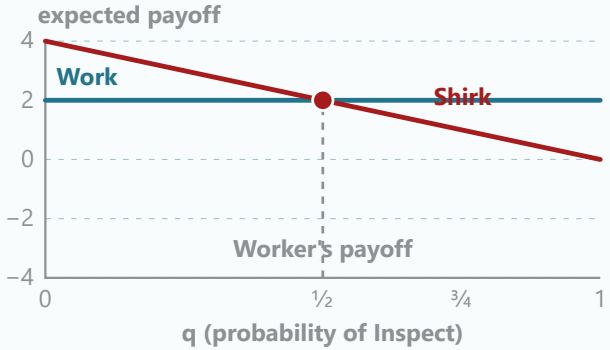
	Inspect	Don't
Work	(2 , 1)	(2 , 2)
Shirk	(0 , −1)	(4 , −4)



1. The worker plays Work with probability **p**, the manager Inspect with probability **q**.
2. Worker: E[Work] = 2q + 2(1−q) = **2**; E[Shirk] = 0·q + 4(1−q) = **4 − 4q**

Computing a Mixed Equilibrium: The Inspection Game

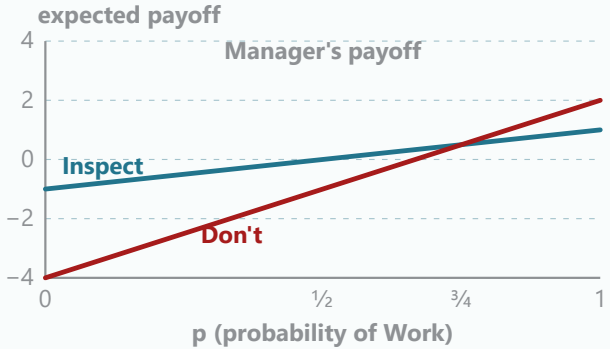
	Inspect	Don't
Work	(2 , 1)	(2 , 2)
Shirk	(0 , −1)	(4 , −4)



1. The worker plays Work with probability **p**, the manager Inspect with probability **q**.
2. Worker: E[Work] = 2q + 2(1−q) = **2**; E[Shirk] = 0·q + 4(1−q) = **4 − 4q**
3. Worker indifferent: 2 = 4 − 4q ⇒ **q* = 1/2**

Computing a Mixed Equilibrium: The Inspection Game

	Inspect	Don't
Work	(2 , 1)	(2 , 2)
Shirk	(0 , −1)	(4 , −4)



1. The worker plays Work with probability **p**, the manager Inspect with probability **q**.
2. Worker: E[Work] = 2q + 2(1−q) = **2**; E[Shirk] = 0·q + 4(1−q) = **4 − 4q**
3. Worker indifferent: 2 = 4 − 4q ⇒ **q* = 1/2**
4. Manager: E[Inspect] = 1·p − 1·(1−p) = **2p − 1**; E[Don't] = 2p − 4(1−p) = **6p − 4**

Computing a Mixed Equilibrium: The Inspection Game

	Inspect	Don't
Work	(2 , 1)	(2 , 2)
Shirk	(0 , −1)	(4 , −4)



1. The worker plays Work with probability **p**, the manager Inspect with probability **q**.
2. Worker: E[Work] = 2q + 2(1−q) = **2**; E[Shirk] = 0·q + 4(1−q) = **4 − 4q**
3. Worker indifferent: 2 = 4 − 4q ⇒ **q* = 1/2**
4. Manager: E[Inspect] = 1·p − 1·(1−p) = **2p − 1**; E[Don't] = 2p − 4(1−p) = **6p − 4**
5. Manager indifferent: 2p − 1 = 6p − 4 ⇒ **p* = 3/4**

Computing a Mixed Equilibrium: The Inspection Game

	Inspect	Don't
Work	(2 , 1)	(2 , 2)
Shirk	(0 , −1)	(4 , −4)



1. The worker plays Work with probability **p**, the manager Inspect with probability **q**.
2. Worker: E[Work] = 2q + 2(1−q) = **2**; E[Shirk] = 0·q + 4(1−q) = **4 − 4q**
3. Worker indifferent: 2 = 4 − 4q ⇒ **q* = 1/2**
4. Manager: E[Inspect] = 1·p − 1·(1−p) = **2p − 1**; E[Don't] = 2p − 4(1−p) = **6p − 4**
5. Manager indifferent: 2p − 1 = 6p − 4 ⇒ **p* = 3/4**
6. Equilibrium: (p*, q*) = (3/4, 1/2). Expected payoffs: worker 2, manager 1/2.

Computing a Mixed Equilibrium: The Inspection Game

	Inspect	Don't
Work	(2 , 1)	(2 , 2)
Shirk	(0 , −1)	(4 , −4)



1. The worker plays Work with probability **p**, the manager Inspect with probability **q**.
2. Worker: E[Work] = 2q + 2(1−q) = **2**; E[Shirk] = 0·q + 4(1−q) = **4 − 4q**
3. Worker indifferent: 2 = 4 − 4q ⇒ **q* = 1/2**
4. Manager: E[Inspect] = 1·p − 1·(1−p) = **2p − 1**; E[Don't] = 2p − 4(1−p) = **6p − 4**
5. Manager indifferent: 2p − 1 = 6p − 4 ⇒ **p* = 3/4**
6. Equilibrium: (p*, q*) = (3/4, 1/2). Expected payoffs: worker 2, manager 1/2.
7. With a harsher penalty (a caught shirker gets −2): q* becomes 1/3, while p* = ¾ **stays the same**!

When the Penalty Changes, Whose Probability Moves?

Worker indifferent: $2 = c \cdot q + 4(1 - q) \Rightarrow \mathbf{q^* = 2/(4 - c)}$ · Manager indifferent: $2p - 1 = 6p - 4 \Rightarrow \mathbf{p^* = 3/4}$

c = 0

$q^* = 2/(4 - c) =$ **1/2 = 0.5** | $p^* =$ **3/4**, independent of c | worker's equilibrium payoff = **2**

Manager

Worker	Inspect	Don't inspect
Work	(2, 1)	(2, 2)
Shirk	(0, −1)	(4, −4)

cell: (worker, manager)

c: payoff of a caught shirker





c changes the worker's payoff, yet the worker stays at $p^* = 3/4$; the manager's q^* moves. Each player's mix is set by the **opponent's** payoffs.

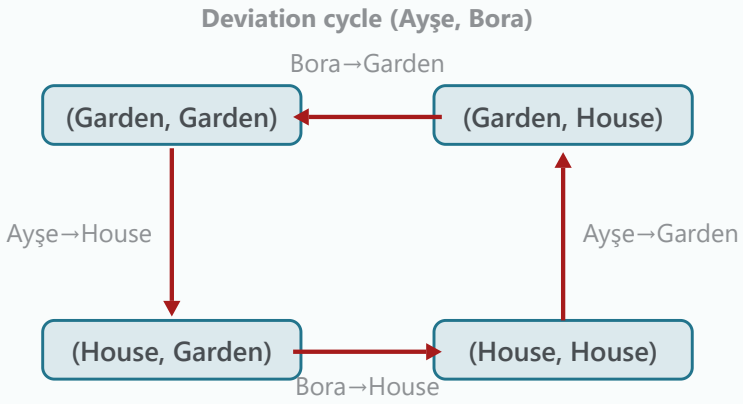
Step by Step: A Mixed Equilibrium by Indifference

Ayşe (hides) \ Bora (seeks)	Garden	House
Garden	(0 , 3)	(2 , 0)
House	(1 , 0)	(0 , 1)

Ayşe hides (row), Bora seeks (column); cells show fun points (Ayşe, Bora). Bora gets 3 for finding her in the garden, 1 in the house; Ayşe gets 2 for not being found in the garden, 1 in the house.

Step by Step: A Mixed Equilibrium by Indifference

Ayşe (hides) \ Bora (seeks)	Garden	House
Garden	(0 , 3)	(2 , 0)
House	(1 , 0)	(0 , 1)



After underlining, no cell has two marks: best responses form a **cycle** (arrows). No pure Nash equilibrium → look for a mixed one.

Step by Step: A Mixed Equilibrium by Indifference

Ayşe (hides) \ Bora (seeks)	Garden	House
Garden	(0 , 3)	(2 , 0)
House	(1 , 0)	(0 , 1)

p = P(Ayşe Garden), q = P(Bora Garden)

Variables: Ayşe picks Garden with probability **p**, Bora with probability **q**. If Ayşe mixes, both options must give her the **same** payoff; that equation gives q.

Step by Step: A Mixed Equilibrium by Indifference

Ayşe (hides) \ Bora (seeks)	Garden	House
Garden	(0 , 3)	(2 , 0)
House	(1 , 0)	(0 , 1)

p = P(Ayşe Garden), q = P(Bora Garden)

Ayşe: E[Garden] = 0·q + 2·(1 − q) = 2 − 2q

Ayşe in the Garden: Bora searches the garden with prob. q (0) and the house with 1 − q (2). Expected payoff **2 − 2q**.

Step by Step: A Mixed Equilibrium by Indifference

Ayşe (hides) \ Bora (seeks)	Garden	House
Garden	(0 , 3)	(2 , 0)
House	(1 , 0)	(0 , 1)

p = P(Ayşe Garden), q = P(Bora Garden)

Ayşe: E[Garden] = 0·q + 2·(1 − q) = 2 − 2q

Ayşe: E[House] = 1·q + 0·(1 − q) = q

Ayşe in the House: 1 if Bora searches the garden (q), 0 if the house. Expected payoff **q**.

Step by Step: A Mixed Equilibrium by Indifference

Ayşe (hides) \ Bora (seeks)	Garden	House
Garden	(0 , 3)	(2 , 0)
House	(1 , 0)	(0 , 1)

p = P(Ayşe Garden), q = P(Bora Garden)

Ayşe: E[Garden] = 0·q + 2·(1 − q) = 2 − 2q

Ayşe: E[House] = 1·q + 0·(1 − q) = q

2 − 2q = q ⇒ 2 = 3q ⇒ q* = 2/3

Equate and solve: 2 − 2q = q ⇒ **q* = 2/3**. Bora must search the garden with probability 2/3 to keep Ayşe indifferent.

Step by Step: A Mixed Equilibrium by Indifference

Ayşe (hides) \ Bora (seeks)	Garden	House
Garden	(0 , 3)	(2 , 0)
House	(1 , 0)	(0 , 1)

p = P(Ayşe Garden), q = P(Bora Garden)

Ayşe: E[Garden] = 0·q + 2·(1 − q) = 2 − 2q

Ayşe: E[House] = 1·q + 0·(1 − q) = q

2 − 2q = q ⇒ 2 = 3q ⇒ q* = 2/3

Bora: E[Garden] = 3·p + 0·(1 − p) = 3p

Bora: E[House] = 0·p + 1·(1 − p) = 1 − p

Same for Bora: given Ayşe's p, his two options give E[Garden] = **3p** and E[House] = **1 − p**.

Step by Step: A Mixed Equilibrium by Indifference

Ayşe (hides) \ Bora (seeks)	Garden	House
Garden	(0 , 3)	(2 , 0)
House	(1 , 0)	(0 , 1)

p = P(Ayşe Garden), q = P(Bora Garden)

Ayşe: E[Garden] = 0·q + 2·(1 − q) = 2 − 2q

Ayşe: E[House] = 1·q + 0·(1 − q) = q

2 − 2q = q ⇒ 2 = 3q ⇒ q* = 2/3

Bora: E[Garden] = 3·p + 0·(1 − p) = 3p

Bora: E[House] = 0·p + 1·(1 − p) = 1 − p

3p = 1 − p ⇒ 4p = 1 ⇒ p* = 1/4

3p = 1 − p ⇒ **p* = 1/4**. Note: p* comes from **Bora's** numbers, not Ayşe's.

Step by Step: A Mixed Equilibrium by Indifference

Ayşe (hides) \ Bora (seeks)	Garden	House
Garden	(0 , 3)	(2 , 0)
House	(1 , 0)	(0 , 1)

p = P(Ayşe Garden), q = P(Bora Garden)

Ayşe: E[Garden] = 0·q + 2·(1 − q) = 2 − 2q

Ayşe: E[House] = 1·q + 0·(1 − q) = q

2 − 2q = q ⇒ 2 = 3q ⇒ q* = 2/3

Bora: E[Garden] = 3·p + 0·(1 − p) = 3p

Bora: E[House] = 0·p + 1·(1 − p) = 1 − p

3p = 1 − p ⇒ 4p = 1 ⇒ p* = 1/4

Payoffs: Ayşe 2/3, Bora 3/4

Check: 0 < 1/4 < 1 and 0 < 2/3 < 1 ✓. Payoffs: Ayşe 2 − 2·(2/3) = **2/3**, Bora 3·(1/4) = **3/4**. Ayşe likes the garden, yet hides there only 25% of the time.

Code Trace: `mixed_2x2(A, B)`

row \ column player	Opera	Football
Opera	(3 , 2)	(0 , 0)
Football	(0 , 0)	(2 , 3)

Battle of the Sexes: A holds the row player's payoffs, B the column player's (0 = Opera, 1 = Football).

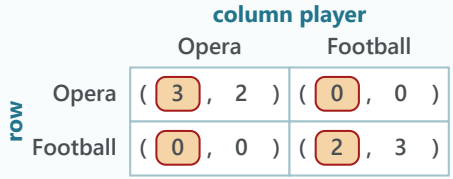
Step 1 / 8

```
1  from fractions import Fraction as F
▶ def mixed_2x2(A, B):
3      da = A[0][0] - A[0][1] - A[1][0] + A[1][1]
4      if da == 0: return None      # degenerate
5      q = F(A[1][1] - A[0][1], da)  # row indifferent
6      db = B[0][0] - B[0][1] - B[1][0] + B[1][1]
7      if db == 0: return None
8      p = F(B[1][1] - B[1][0], db)  # col indifferent
9      if not (0 < p < 1 and 0 < q < 1):
10         return None               # no interior eq.
11     u1 = A[0][0] * q + A[0][1] * (1 - q)
12     return p, q, u1
```

VARIABLES

—

Code Trace: `mixed_2x2(A, B)`



Denominator from the row player's payoff differences: da = 3 − 0 − 0 + 2 = **5**.

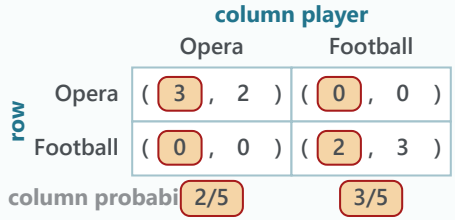
Step 2 / 8

```
1  from fractions import Fraction as F
2  def mixed_2x2(A, B):
▶      da = A[0][0] - A[0][1] - A[1][0] + A[1][1]
4      if da == 0: return None      # degenerate
5      q = F(A[1][1] - A[0][1], da)  # row indifferent
6      db = B[0][0] - B[0][1] - B[1][0] + B[1][1]
7      if db == 0: return None
8      p = F(B[1][1] - B[1][0], db)  # col indifferent
9      if not (0 < p < 1 and 0 < q < 1):
10         return None               # no interior eq.
11     u1 = A[0][0] * q + A[0][1] * (1 - q)
12     return p, q, u1
```

VARIABLES

da = 5

Code Trace: `mixed_2x2(A, B)`



da ≠ 0, so the check passes. Setting u_1(Opera, q) = 3q equal to u_1(Football, q) = 2(1 − q) ⇒ q = (2 − 0)/5 = **2/5**: the column player plays Opera with probability 2/5.

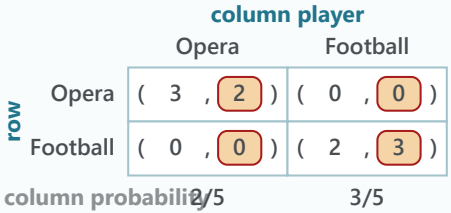
Step 3 / 8

```
1  from fractions import Fraction as F
2  def mixed_2x2(A, B):
3      da = A[0][0] - A[0][1] - A[1][0] + A[1][1]
▶      if da == 0: return None     # degenerate
▶      q = F(A[1][1] - A[0][1], da)  # row indifferent
6      db = B[0][0] - B[0][1] - B[1][0] + B[1][1]
7      if db == 0: return None
8      p = F(B[1][1] - B[1][0], db)  # col indifferent
9      if not (0 < p < 1 and 0 < q < 1):
10         return None               # no interior eq.
11     u1 = A[0][0] * q + A[0][1] * (1 - q)
12     return p, q, u1
```

VARIABLES

da = **5** q = **2/5**

Code Trace: `mixed_2x2(A, B)`



Now the column player's payoffs: db = 2 − 0 − 0 + 3 = **5**.

Step 4 / 8

```
1  from fractions import Fraction as F
2  def mixed_2x2(A, B):
3      da = A[0][0] - A[0][1] - A[1][0] + A[1][1]
4      if da == 0: return None      # degenerate
5      q = F(A[1][1] - A[0][1], da)  # row indifferent
▶      db = B[0][0] - B[0][1] - B[1][0] + B[1][1]
7      if db == 0: return None
8      p = F(B[1][1] - B[1][0], db)  # col indifferent
9      if not (0 < p < 1 and 0 < q < 1):
10         return None               # no interior eq.
11     u1 = A[0][0] * q + A[0][1] * (1 - q)
12     return p, q, u1
```

VARIABLES

da = **5** q = **2/5** db = **5**

Code Trace: `mixed_2x2(A, B)`



db ≠ 0. Setting $u_2(p, \text{Opera}) = 2p$ equal to $u_2(p, \text{Football}) = 3(1 - p) \Rightarrow p =$ **3/5**: the row player plays Opera with probability 3/5.

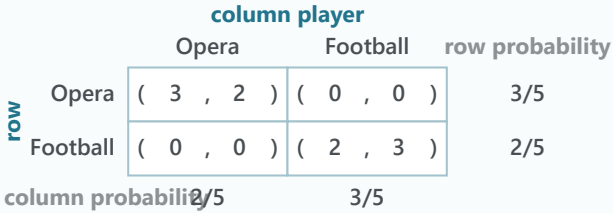
Step 5 / 8

```
1  from fractions import Fraction as F
2  def mixed_2x2(A, B):
3      da = A[0][0] - A[0][1] - A[1][0] + A[1][1]
4      if da == 0: return None      # degenerate
5      q = F(A[1][1] - A[0][1], da)  # row indifferent
6      db = B[0][0] - B[0][1] - B[1][0] + B[1][1]
▶      if db == 0: return None
▶      p = F(B[1][1] - B[1][0], db)  # col indifferent
9      if not (0 < p < 1 and 0 < q < 1):
10         return None               # no interior eq.
11     u1 = A[0][0] * q + A[0][1] * (1 - q)
12     return p, q, u1
```

VARIABLES

`da = 5` `q = 2/5` `db = 5` `p = 3/5`

Code Trace: `mixed_2x2(A, B)`



0 < p, q < 1 holds: both are genuine mixtures ⇒ an interior mixed equilibrium exists, so return None is skipped.

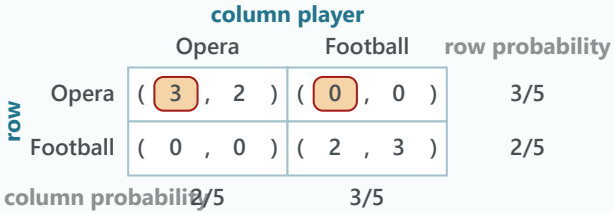
Step 6 / 8

```
1  from fractions import Fraction as F
2  def mixed_2x2(A, B):
3      da = A[0][0] - A[0][1] - A[1][0] + A[1][1]
4      if da == 0: return None      # degenerate
5      q = F(A[1][1] - A[0][1], da)  # row indifferent
6      db = B[0][0] - B[0][1] - B[1][0] + B[1][1]
7      if db == 0: return None
8      p = F(B[1][1] - B[1][0], db)  # col indifferent
▶      if not (0 < p < 1 and 0 < q < 1):
10         return None               # no interior eq.
11     u1 = A[0][0] * q + A[0][1] * (1 - q)
12     return p, q, u1
```

VARIABLES

`da = 5` `q = 2/5` `db = 5` `p = 3/5`

Code Trace: `mixed_2x2(A, B)`



Expected payoff $u_1 = 3\cdot(2/5) + 0\cdot(3/5) = \mathbf{6/5}$; $u_2 = 2\cdot(3/5) = 6/5$. Worse than either pure equilibrium (2 or 3).

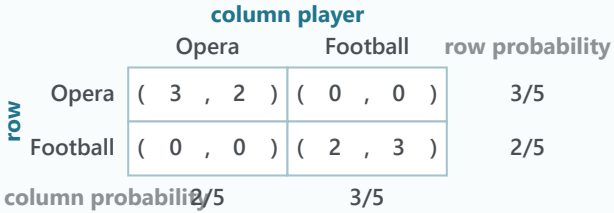
Step 7 / 8

```
1  from fractions import Fraction as F
2  def mixed_2x2(A, B):
3      da = A[0][0] - A[0][1] - A[1][0] + A[1][1]
4      if da == 0: return None      # degenerate
5      q = F(A[1][1] - A[0][1], da)  # row indifferent
6      db = B[0][0] - B[0][1] - B[1][0] + B[1][1]
7      if db == 0: return None
8      p = F(B[1][1] - B[1][0], db)  # col indifferent
9      if not (0 < p < 1 and 0 < q < 1):
10         return None               # no interior eq.
▶      u1 = A[0][0] * q + A[0][1] * (1 - q)
12     return p, q, u1
```

VARIABLES

da = 5 | q = 2/5 | db = 5 | p = 3/5 | u1 = 6/5

Code Trace: `mixed_2x2(A, B)`



Result (3/5, 2/5, 6/5). In Free Money q = 0 ⇒ None. In the Prisoner's Dilemma da = 0 ⇒ the first check returns None.

Step 8 / 8

```
1  from fractions import Fraction as F
2  def mixed_2x2(A, B):
3      da = A[0][0] - A[0][1] - A[1][0] + A[1][1]
4      if da == 0: return None      # degenerate
5      q = F(A[1][1] - A[0][1], da)  # row indifferent
6      db = B[0][0] - B[0][1] - B[1][0] + B[1][1]
7      if db == 0: return None
8      p = F(B[1][1] - B[1][0], db)  # col indifferent
9      if not (0 < p < 1 and 0 < q < 1):
10         return None               # no interior eq.
11     u1 = A[0][0] * q + A[0][1] * (1 - q)
▶      return p, q, u1
```

VARIABLES

`da = 5` `q = 2/5` `db = 5` `p = 3/5` `u1 = 6/5`

Mixed Equilibria of Six Classic Games

Game	Pure NE	Mixed NE (p*, q*)	(u_1, u_2)	In the mixed equilibrium
Matching Pennies	0	1/2, 1/2	(0, 0)	Each outcome 25%; unexploitable
Chicken	2	9/10, 9/10	(−1/10, −1/10)	Crash $(1/10)^2$ = 1%; with crash cost −k, swerve (k − 1)/k
Battle of the Sexes	2	3/5, 2/5	(6/5, 6/5)	Miss each other 52%
Stag Hunt (stag 4, hare 3)	2	3/4, 3/4	(3, 3)	Miscoordinate 37.5%; both stag ≈ 56%
Pure Coordination	2	1/3, 1/3	(10/3, 10/3)	Mismatch ≈ 44%; a better party is chosen less
Free Money	2	none	—	For q > 0, Heads is strictly better; only boundary (0, 0)

p*, q*: probability of the first strategy. **Chicken:** (Swerve, Swerve) = (0, 0), (Swerve, Straight) = (−1, 1), (Straight, Straight) = (−10, −10). **Pure Coordination:** (Party, Party) = (10, 10), (Home, Home) = (5, 5). **Free Money:** (Heads, Heads) = (1, 1), else 0.

Common Mistakes: Mixed Equilibrium

Finding p from your own numbers

The row player's p makes the *column player* indifferent, so it is computed from the column player's payoffs.

Forgetting (1 − q)

E = 0·q + 2·(1 − q) = 2 − 2q, not "2q". Each term is multiplied by its own probability.

Not checking the result

If p or q falls outside (0, 1), there is no interior mixed equilibrium (usually some strategy is dominated).

Skipping pure equilibria

The mixed calculation does not replace the pure equilibria; underline first, then look for the mixed one.

"Indifferent, so any p works"

A player is indifferent among their own options, but in equilibrium plays the unique p that leaves the opponent indifferent.

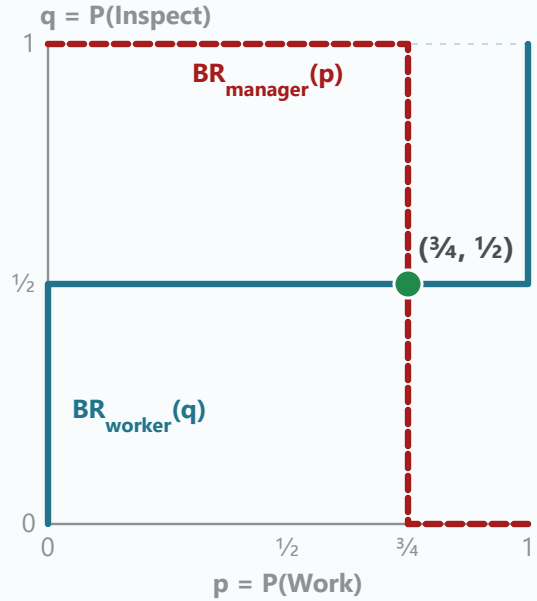
Equating the wrong things

Set equal the expected payoffs of *one player's* two options, not the two players' payoffs.

Nash's Existence Theorem (1950)

Theorem. Every game with finitely many players and finite strategy sets has at least one (mixed-strategy) Nash equilibrium.

- Proof idea: a mixed profile in which everyone best-responds to everyone always exists; fixed-point theorems (**Kakutani**, Brouwer) guarantee it. That profile is a Nash equilibrium.
- The guarantee is for **mixed** strategies; a pure equilibrium may not exist (Matching Pennies).
- With infinite strategy sets an equilibrium may not exist (e.g. "whoever names the largest number wins").



Inspection game: the two best-response curves cross at the equilibrium

Step by Step: Three Equilibria in Channel Sharing

Device 1 \ Device 2	Send	Wait
Send	(−2, −2)	(3, 1)
Wait	(1, 3)	(0, 0)

Two devices want to send a packet on the same channel. If both send, they collide (−2 each); if one sends, it gets 3 and the waiting one 1; if both wait, the slot is wasted (0). This is the Chicken structure.

Step by Step: Three Equilibria in Channel Sharing

Device 1 \ Device 2	Send	Wait
Send	(−2, −2)	(3, 1)
Wait	(1, 3)	(0, 0)

Device 1 (first numbers): if the other Sends, $-2 < 1$ → **Wait**; if the other Waits, $3 > 0$ → **Send**. It wants to do the opposite of the other.

Step by Step: Three Equilibria in Channel Sharing

Device 1 \ Device 2	Send	Wait
Send	(−2, −2)	(3, 1)
Wait	(1, 3)	(0, 0)

Same for Device 2 (second numbers). Doubly underlined cells: **(Send, Wait) = (3, 1)** and **(Wait, Send) = (1, 3)**: two pure equilibria.

Step by Step: Three Equilibria in Channel Sharing

Device 1 \ Device 2	Send	Wait
Send	(−2, −2)	(3, 1)
Wait	(1, 3)	(0, 0)

q = P(Device 2 Sends)

u_1(Send, q) = −2·q + 3·(1 − q) = 3 − 5q

The third equilibrium is mixed: q = P(Device 2 Sends). Device 1's expected payoff from Send is −2q + 3(1 − q) = **3 − 5q**.

Step by Step: Three Equilibria in Channel Sharing

Device 1 \ Device 2	Send	Wait
Send	(−2, −2)	(3, 1)
Wait	(1, 3)	(0, 0)

q = P(Device 2 Sends)

$u_1(\text{Send}, q) = -2 \cdot q + 3 \cdot (1 - q) = 3 - 5q$

$u_1(\text{Wait}, q) = 1 \cdot q + 0 \cdot (1 - q) = q$

From Wait: 1 if the other sends, 0 if it waits → **q**.

Step by Step: Three Equilibria in Channel Sharing

Device 1 \ Device 2	Send	Wait
Send	(−2, −2)	(3, 1)
Wait	(1, 3)	(0, 0)

q = P(Device 2 Sends)

$u_1(\text{Send}, q) = -2 \cdot q + 3 \cdot (1 - q) = 3 - 5q$

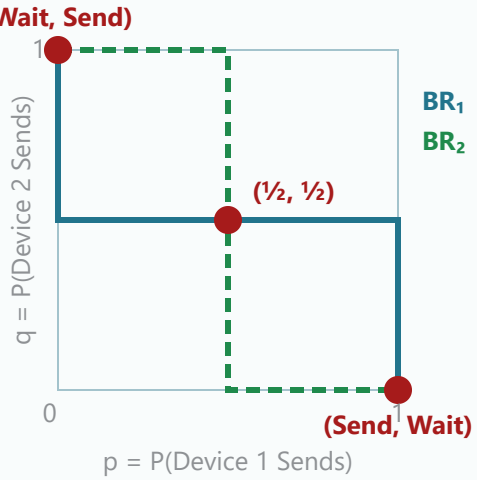
$u_1(\text{Wait}, q) = 1 \cdot q + 0 \cdot (1 - q) = q$

$3 - 5q = q \Rightarrow q^* = 1/2$; symmetry: $p^* = 1/2$

Indifference: $3 - 5q = q \Rightarrow$ **$q^* = 1/2$**. The game is symmetric, so Device 2's indifference gives **$p^* = 1/2$**.

Step by Step: Three Equilibria in Channel Sharing

Device 1 \ Device 2	Send	Wait
Send	(−2, −2)	(3, 1)
Wait	(1, 3)	(0, 0)



Best-response curves BR_1 (solid) and BR_2 (dashed) cross at **three points**: the two corners are the pure equilibria, the middle point the mixed one.

Step by Step: Three Equilibria in Channel Sharing

Device 1 \ Device 2	Send	Wait
Send	(−2, −2)	(3, 1)
Wait	(1, 3)	(0, 0)

Outcomes in the mixed equilibrium (½, ½):

collision (S, S): ¼ → −2

one sender (success): ½

idle channel (W, W): ¼ → 0

expected payoff: ½ (3 or 1 when pure)

The mixed equilibrium is fair but inefficient: collisions ¼ of the time, an idle channel another ¼; each device averages ½. Random backoff in network protocols is this randomization in practice.

How Many Equilibria? The Oddness Rule

Game	Pure equilibria	Mixed equilibria	Total
Prisoner's Dilemma	1	0	1
Hide and Seek, Inspection, Matching Pennies	0	1	1
Channel sharing, Chicken, Battle of the Sexes, Stag Hunt, Pure Coordination	2	1	3
Language choice (3×3)	2	1 (Ayşe picks Python with prob. 3/4, Bora with prob. 1/5)	3
The Trap of Weak Dominance (ties)	3	infinitely many	—
Free Money (ties)	2	0	2 (even)

Oddness (Wilson, 1971)

In nondegenerate games (no special ties among payoffs), the number of equilibria is finite and *odd* (1, 3, 5, ...).

Rule of thumb

Found two pure equilibria? There is very likely a mixed one too; look for it with indifference. With ties, the rule can fail.

Multiple Equilibria: Stag Hunt

Hunter 1 \ Hunter 2	Stag	Hare
Stag	(4, 4)	(0, 3)
Hare	(3, 0)	(3, 3)

Stag Hunt: a stag can only be caught together (4 each); a hare guarantees 3 alone, whatever the partner does.

Step 1 / 5

Players: Hunter 1 on rows, Hunter 2 on columns; cells are (Hunter 1, Hunter 2).

Multiple Equilibria: Stag Hunt

Hunter 1 \ Hunter 2	Stag	Hare
Stag	(4, 4)	(0, 3)
Hare	(3, 0)	(3, 3)

Best responses coincide in two cells: **(Stag, Stag)** and **(Hare, Hare)**. Which one will be played?

Players: Hunter 1 on rows, Hunter 2 on columns; cells are (Hunter 1, Hunter 2).

Multiple Equilibria: Stag Hunt

Hunter 1 \ Hunter 2	Stag	Hare
Stag	(4, 4)	(0, 3)
Hare	(3, 0)	(3, 3)

The third equilibrium is mixed: if the other picks Stag with probability q, 4q = 3 ⇒ **q = ¾**. Stag is a best response only if you believe your partner comes with probability ≥ 75%.

Players: Hunter 1 on rows, Hunter 2 on columns; cells are (Hunter 1, Hunter 2).

Multiple Equilibria: Stag Hunt

Hunter 1 \ Hunter 2	Stag	Hare
Stag	(4, 4)	(0, 3)
Hare	(3, 0)	(3, 3)

Payoff dominance: (Stag, Stag) gives both hunters more (4 > 3) → the efficiency criterion picks it.

Players: Hunter 1 on rows, Hunter 2 on columns; cells are (Hunter 1, Hunter 2).

Multiple Equilibria: Stag Hunt

Hunter 1 \ Hunter 2	Stag	Hare
Stag	(4, 4)	(0, 3)
Hare	(3, 0)	(3, 3)

Risk dominance (Harsanyi–Selten): product of deviation losses at (S,S): (4−3)·(4−3) = 1; at (H,H): (3−0)·(3−0) = 9 → **(Hare, Hare)** is “safer”.

Players: Hunter 1 on rows, Hunter 2 on columns; cells are (Hunter 1, Hunter 2).

Equilibrium Selection: Which One Is Played?

Payoff dominance

Pick the equilibrium that makes everyone better off: (S, S) in Stag Hunt. Requires trust.

Risk dominance

Unsure what the other side will do? Pick the one with lower risk of loss: (H, H) in Stag Hunt.

Focal point (Schelling, 1960)

Culture, habit or salience makes one equilibrium stand out: “If we meet in Istanbul, it's Taksim at noon.”

Correlated equilibrium

Aumann (1974): a shared signal (a traffic light) steers players to one of the equilibria; it resolves the Chicken game at an intersection.

Learning and evolution

In repeated play, players learn by trial; which equilibrium they reach depends on initial behavior.

Fairness

In Battle of the Sexes each pure equilibrium favors one player; the mixed equilibrium gives both only 6/5.

Change the Assumption: What If a Player Makes Mistakes?

Prisoner's Dilemma (minus prison years)

	Silent	**Betray**
Silent	(−1, −1)	(−3, 0)
Betray	(0, −3)	(−2, −2)

Stag Hunt

	Stag	**Hare**
Stag	(4, 4)	(0, 3)
Hare	(3, 0)	(3, 3)

If the opponent errs with probability ε, **does the best response change?**

Change the Assumption: What If a Player Makes Mistakes?

Prisoner's Dilemma (minus prison years)

	Silent	**Betray**
Silent	(−1, −1)	(−3, 0)
Betray	(0, −3)	(−2, −2)

Stag Hunt

	Stag	**Hare**
Stag	(4, 4)	(0, 3)
Hare	(3, 0)	(3, 3)

If the opponent errs with probability ε, **does the best response change?**

Prisoner's Dilemma: no change

Opponent Silent with probability p.
Silent: $-1 \cdot p - 3 \cdot (1 - p) = 2p - 3$
Betray: $0 \cdot p - 2 \cdot (1 - p) = 2p - 2$
Betray wins by 1 for every p.

Change the Assumption: What If a Player Makes Mistakes?

Prisoner's Dilemma (minus prison years)

	Silent	Betray
Silent	(−1, −1)	(−3, 0)
Betray	(0, −3)	(−2, −2)

Stag Hunt

	Stag	Hare
Stag	(4, 4)	(0, 3)
Hare	(3, 0)	(3, 3)

If the opponent errs with probability ε, **does the best response change?**

Prisoner's Dilemma: no change

Opponent Silent with probability p.
Silent: $-1 \cdot p - 3 \cdot (1 - p) = 2p - 3$
Betray: $0 \cdot p - 2 \cdot (1 - p) = 2p - 2$
Betray wins by 1 for every p.

Stag Hunt: depends on the error rate

Opponent slips to Hare with probability q.
Stag: $4 \cdot (1 - q)$ · Hare: always 3
Stag is a best response only if $4(1 - q) \geq 3$: **$q \leq 1/4$**. Otherwise play the risk-dominant Hare.

Dominant-strategy equilibria survive mistakes; coordination equilibria may not.

Correlated Equilibrium (Aumann, 1974): A Traffic Light at the Intersection

Driver 1 \ Driver 2	Yield	Go
Yield	(0, 0)	(−1, 1)
Go	(1, −1)	(−10, −10)

Pure equilibria: (Yield, Go), (Go, Yield)

Mixed equilibrium: yield 9/10

crash 1/100, payoff −1/10

Two drivers reach an unsignalled intersection at once: give way (**Yield**) or drive through (**Go**). This is Chicken: each pure equilibrium favors one driver; without coordination, in the symmetric mixed equilibrium each driver yields with probability 9/10, they crash 1% of the time and the expected payoff is −1/10.

Correlated Equilibrium (Aumann, 1974): A Traffic Light at the Intersection

Driver 1 \ Driver 2	Yield	Go
Yield	(0, 0) μ = 1/3	(−1, 1) μ = 1/3
Go	(1, −1) μ = 1/3	(−10, −10) μ = 0

The light picks one of 3 cells w.p. 1/3 and tells each driver ONLY their own recommendation.

A **mediator** (the traffic light) draws a cell from a joint distribution μ and privately tells each driver only their own recommendation.

Correlated Equilibrium (Aumann, 1974): A Traffic Light at the Intersection

Driver 1 \ Driver 2	Yield	Go
Yield	(0, 0) μ = 1/3	(−1, 1) μ = 1/3
Go	(1, −1) μ = 1/3	(−10, −10) μ = 0

D1 told “Yield” ⇒ D2: Yield ½, Go ½

Obey (Yield): $0 \cdot \frac{1}{2} + (-1) \cdot \frac{1}{2} = -\frac{1}{2}$

Deviate (Go): $1 \cdot \frac{1}{2} + (-10) \cdot \frac{1}{2} = -4.5$

$-\frac{1}{2} > -4.5 \Rightarrow$ obeying is better

If told “Yield”, Bayes' rule makes the opponent ½–½ Yield/Go; obeying ($-\frac{1}{2}$) beats deviating (-4.5).

Correlated Equilibrium (Aumann, 1974): A Traffic Light at the Intersection

Driver 1 \ Driver 2	Yield	Go
Yield	(0, 0) μ = 1/3	(−1, 1) μ = 1/3
Go	(1, −1) μ = 1/3	(−10, −10) μ = 0

D1 told “Go” ⇒ D2 surely Yields

Obey (Go): 1 Deviate (Yield): 0

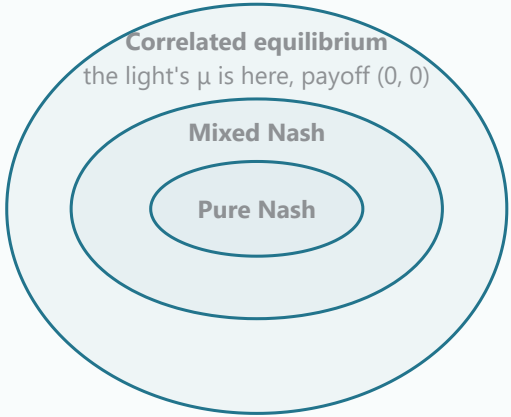
1 > 0 ⇒ obeying is better

Symmetric for Driver 2.

If told “Go”, the opponent was surely told “Yield”; obeying (1) beats deviating (0). No recommendation is worth deviating from ⇒ a **correlated equilibrium**.

Correlated Equilibrium (Aumann, 1974): A Traffic Light at the Intersection

Driver 1 \ Driver 2	Yield	Go
Yield	(0, 0) μ = 1/3	(−1, 1) μ = 1/3
Go	(1, −1) μ = 1/3	(−10, −10) μ = 0



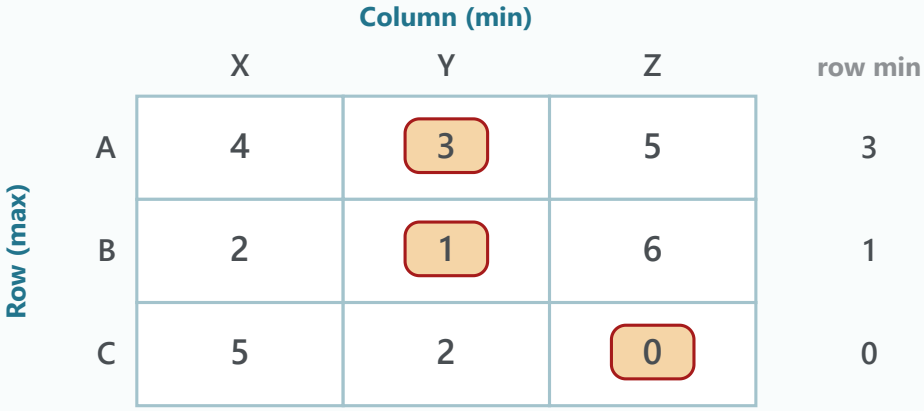
Expected payoff 0 each (better than −1/10 in the mixed equilibrium), with no crashes. Every Nash equilibrium is a correlated equilibrium, but not conversely. Correlated equilibria can be found in polynomial time by linear programming; tuning light durations to traffic is a separate design choice.

Zero-Sum Games: Maximin and Minimax

Row (max) \ Column (min)	X	Y	Z
A	4	3	5
B	2	1	6
C	5	2	0

Zero-sum: each cell shows only the row player's payoff; the column player **loses** the same amount ($u_2 = -u_1$).

Zero-Sum Games: Maximin and Minimax



The row player is pessimistic: in each row, the **worst** value the opponent can leave: 3, 1, 0.

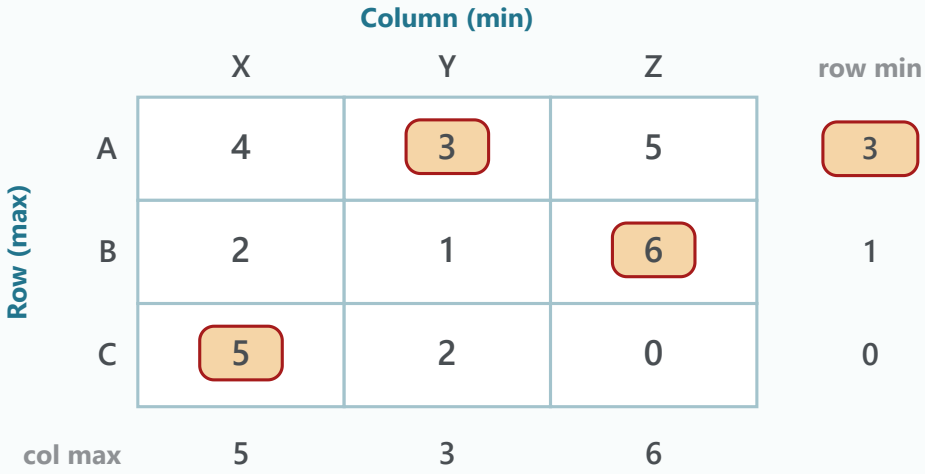
Zero-Sum Games: Maximin and Minimax

Row (max)	X	Y	Z	row min
A	4	3	5	3
B	2	1	6	1
C	5	2	0	0

Column (min)

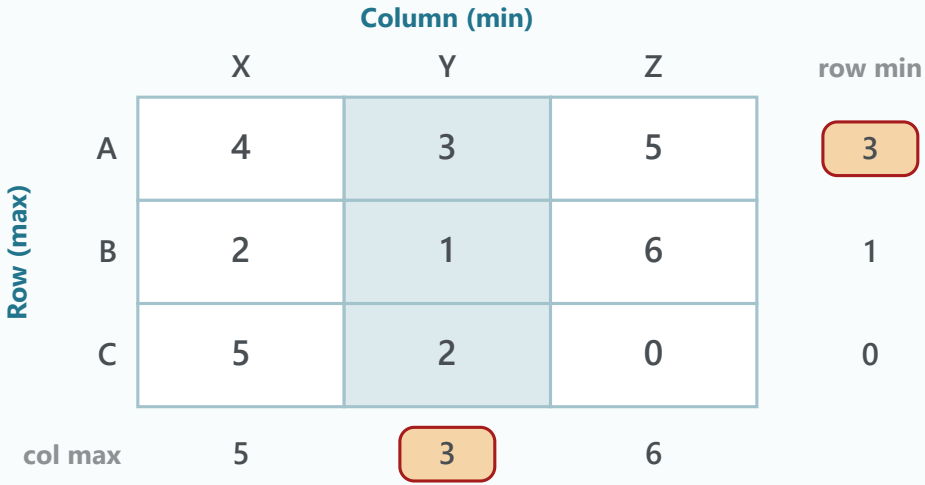
Maximin = max(3, 1, 0) = **3** (row A). The row player can guarantee at least 3.

Zero-Sum Games: Maximin and Minimax



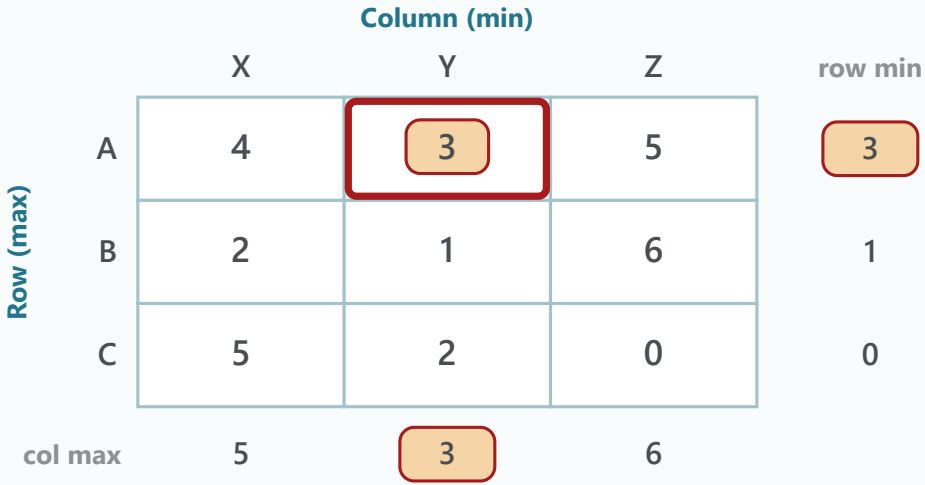
The column player is pessimistic too: in each column, the **highest** value the row player can get: 5, 3, 6.

Zero-Sum Games: Maximin and Minimax



Minimax = min(5, 3, 6) = **3** (column Y). The column player can guarantee losing at most 3.

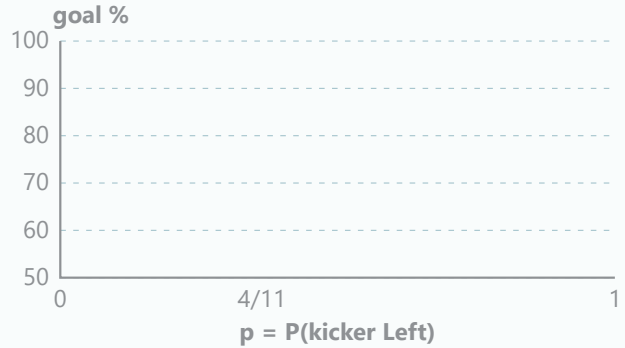
Zero-Sum Games: Maximin and Minimax



Maximin = minimax = 3 ⇒ **saddle point** (A, Y): smallest in its row, largest in its column. It is a pure Nash equilibrium; **the value of the game is v = 3**.

No Saddle Point: Penalty Kicks

Kicker \ Keeper	Left	Right	min
Left	60	95	60
Right	90	70	70
max	90	95	

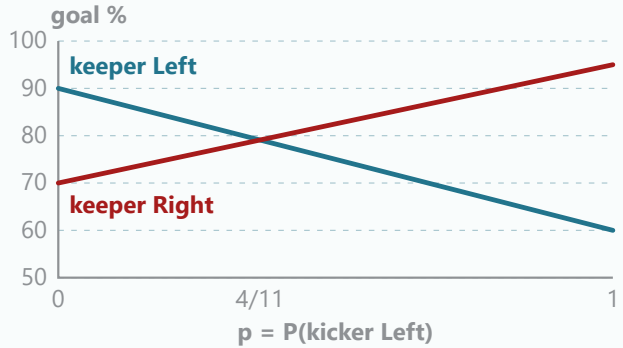


1. Row minima 60, 70 → maximin 70; column maxima 90, 95 → minimax 90. 70 < 90: **no saddle point**.

Illustrative values (real data show a similar pattern: Palacios-Huerta, 2003). Left/Right are from the kicker's view.

No Saddle Point: Penalty Kicks

Kicker \ Keeper	Left	Right
Left	60	95
Right	90	70

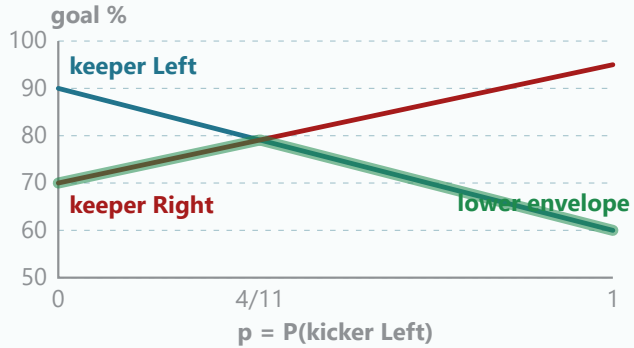


1. Row minima 60, 70 → maximin 70; column maxima 90, 95 → minimax 90. 70 < 90: **no saddle point**.
2. Kicker goes Left with prob. p: keeper Left ⇒ 60p + 90(1−p) = **90 − 30p**; keeper Right ⇒ 95p + 70(1−p) = **70 + 25p**.

Illustrative values (real data show a similar pattern: Palacios-Huerta, 2003). Left/Right are from the kicker's view.

No Saddle Point: Penalty Kicks

Kicker \ Keeper	Left	Right
Left	60	95
Right	90	70

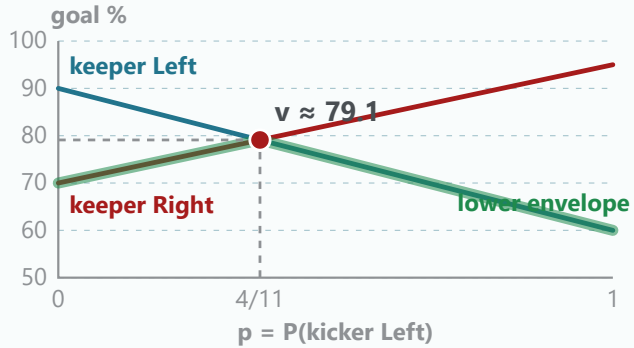


1. Row minima 60, 70 → maximin 70; column maxima 90, 95 → minimax 90. 70 < 90: **no saddle point**.
2. Kicker goes Left with prob. p: keeper Left ⇒ 60p + 90(1−p) = **90 − 30p**; keeper Right ⇒ 95p + 70(1−p) = **70 + 25p**.
3. The keeper minimizes the scoring rate, so the kicker's guarantee is the **lower envelope** of the two lines.

Illustrative values (real data show a similar pattern: Palacios-Huerta, 2003). Left/Right are from the kicker's view.

No Saddle Point: Penalty Kicks

Kicker \ Keeper	Left	Right
Left	60	95
Right	90	70

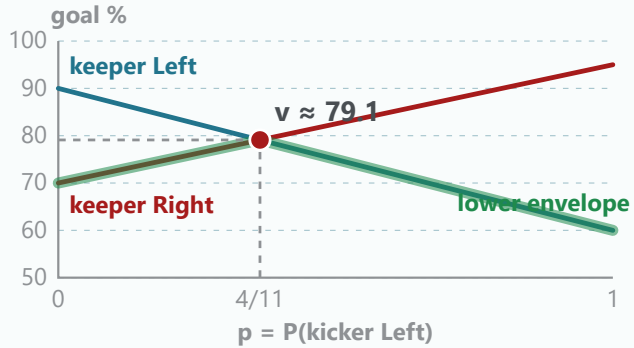


1. Row minima 60, 70 → maximin 70; column maxima 90, 95 → minimax 90. 70 < 90: **no saddle point**.
2. Kicker goes Left with prob. p: keeper Left ⇒ 60p + 90(1−p) = **90 − 30p**; keeper Right ⇒ 95p + 70(1−p) = **70 + 25p**.
3. The keeper minimizes the scoring rate, so the kicker's guarantee is the **lower envelope** of the two lines.
4. Peak of the lower envelope: 90 − 30p = 70 + 25p ⇒ **p* = 4/11 ≈ 0.36**, **v = 870/11 ≈ 79.1%**.

Illustrative values (real data show a similar pattern: Palacios-Huerta, 2003). Left/Right are from the kicker's view.

No Saddle Point: Penalty Kicks

Kicker \ Keeper	Left	Right
Left	60	95
Right	90	70

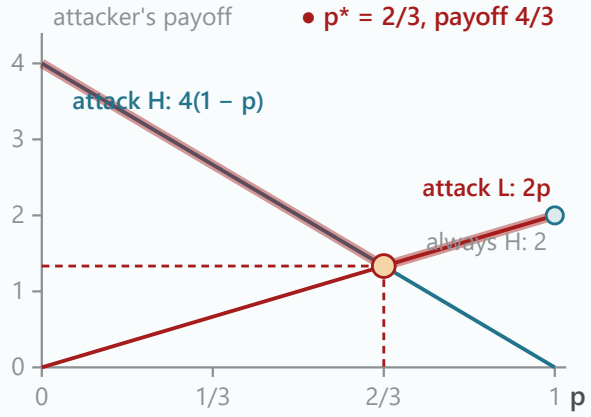


1. Row minima 60, 70 → maximin 70; column maxima 90, 95 → minimax 90. 70 < 90: **no saddle point**.
2. Kicker goes Left with prob. p: keeper Left ⇒ 60p + 90(1−p) = **90 − 30p**; keeper Right ⇒ 95p + 70(1−p) = **70 + 25p**.
3. The keeper minimizes the scoring rate, so the kicker's guarantee is the **lower envelope** of the two lines.
4. Peak of the lower envelope: 90 − 30p = 70 + 25p ⇒ **p* = 4/11 ≈ 0.36**, **v = 870/11 ≈ 79.1%**.
5. Keeper goes Left with prob. q: 60q + 95(1−q) = 90q + 70(1−q) ⇒ **q* = 5/11**; the keeper also guarantees 79.1% ⇒ max-min = min-max. Step 5 / 5

Illustrative values (real data show a similar pattern: Palacios-Huerta, 2003). Left/Right are from the kicker's view.

Security Games: Randomized Patrols

- Many targets, limited resources: airports, ports, forest patrols. A defender covering k of n targets has C(n, k) pure strategies; for n = 20, k = 5 that is **15,504**.
- The attacker can **observe** the patrol pattern: a fixed schedule can be exploited, so the defender randomizes. The ARMOR system at Los Angeles International Airport (2007) scheduled patrols with this model (Tambe, 2011).



One guard, two targets

Airport H is worth 4, port L is worth 2. The guard covers H with probability p and L with 1 − p; attacking an uncovered target wins its value, attacking a covered one wins 0. Equate: 4(1 − p) = 2p ⇒ **p = 2/3**; the attacker gets **4/3**. If the guard always covered H, the attacker would take 2 from L.

Von Neumann's Minimax Theorem (1928)

In every finite two-player zero-sum game $\max_x \min_y x^T A y = \min_y \max_x x^T A y = v$

Equilibria = minimax strategies

(x*, y*) is a Nash equilibrium ⇔ x* is a maximin and y* a minimax strategy. Every equilibrium has the same value v.

Interchangeability

If (x, y) and (x′, y′) are equilibria, so is (x, y′): no coordination problem.

Linear programming

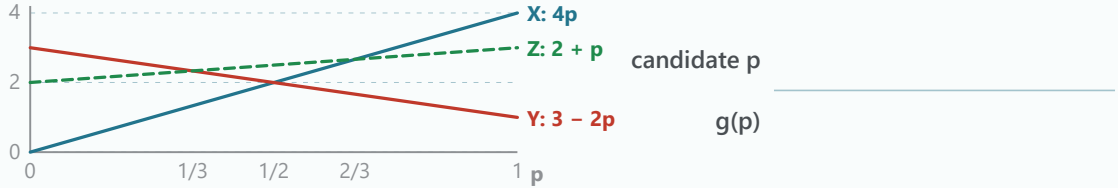
The row player seeks the mix x and the largest v such that x earns at least v against every column; this is a linear program that computers solve quickly.

Link to learning

If players learn by increasing the weight of strategies that did well, their average strategies approach minimax.

Nash's theorem can be seen as a generalization of the minimax theorem to non-zero-sum n-player games; but in general games the value is not unique and LP does not solve them.

Code Trace: `maximin(A)` — 2 × n Zero-Sum



A = [[4, 1, 3], [0, 3, 2]]: if the row player plays row 1 with probability p, each column gives a line. Pure maximin is 1, minimax is 3: no saddle point.

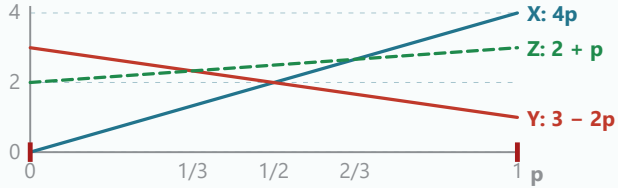
Step 1 / 13

```
1  from fractions import Fraction as F
▶  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
6              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
7              if d != 0:
8                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
10     p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

VARIABLES

—

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1
g(p)	·	·

n = 3 columns. The candidates start with the endpoints: p = 0 and p = 1.

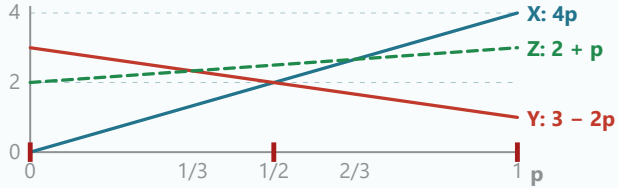
Step 2 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
▶      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
6              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
7              if d != 0:
8                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
10     p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

VARIABLES

`n = 3` `cand = [0, 1]`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2
g(p)	·	·	·

Columns X and Y: d = 6 ≠ 0 ⇒ the lines cross at p = **1/2**; add it to the candidates.

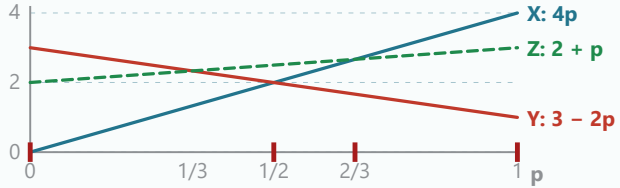
Step 3 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
▶              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
▶              if d != 0:
▶                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
10     p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

VARIABLES

`n = 3` `cand = [0, 1, 1/2]` `j = 0` `k = 1` `d = 6`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2	2/3
g(p)	·	·	·	·

Columns X and Z: d = 3 ≠ 0 ⇒ the lines cross at p = **2/3**; add it to the candidates.

Step 4 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
▶              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
▶              if d != 0:
▶                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
10     p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

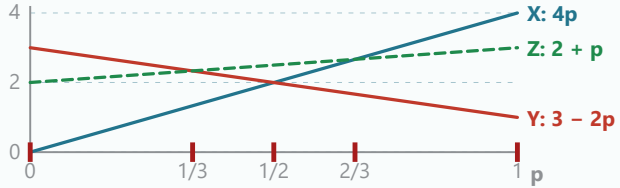
VARIABLES

`n = 3`

`cand = [0, 1, 1/2, 2/3]`

`j = 0` `k = 2` `d = 3`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2	2/3	1/3
g(p)	·	·	·	·	·

Columns Y and Z: d = −3 ≠ 0 ⇒ the lines cross at p = **1/3**; add it to the candidates.

Step 5 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
▶              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
▶              if d != 0:
▶                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
10     p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

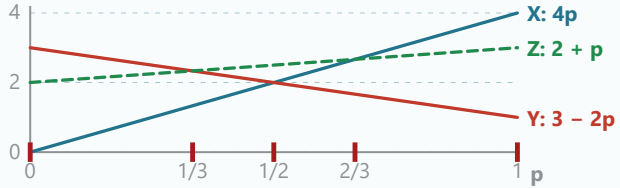
VARIABLES

`n = 3`

`cand = [0, 1, 1/2, 2/3, 1/3]`

`j = 1` `k = 2` `d = -3`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2	2/3	1/3
g(p)	·	·	·	·	·

$g(p) = \min_j [p \cdot A[0][j] + (1 - p) \cdot A[1][j]]$: the column player answers each p with the worst column (the lower envelope).

Step 6 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
6              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
7              if d != 0:
8                  cand.append(F(A[1][k] - A[1][j], d))
▶      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
10     p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

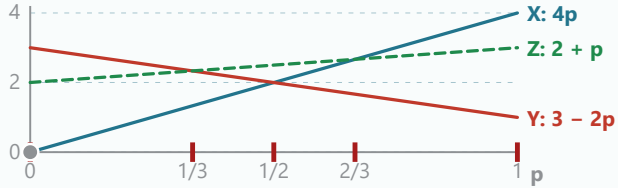
VARIABLES

`n = 3`

`cand = [0, 1, 1/2, 2/3, 1/3]`

`j = 2` `k = 2` `d = -3`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2	2/3	1/3
g(p)	0	·	·	·	·

max tries a candidate: g(0) = 0. Best so far.

Step 7 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
6              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
7              if d != 0:
8                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
▶      p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

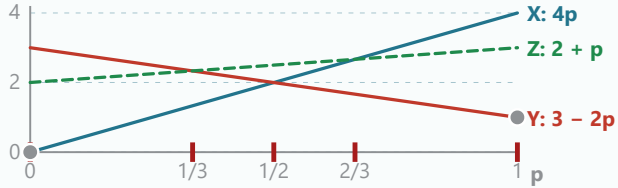
VARIABLES

`n = 3`

`cand = [0, 1, 1/2, 2/3, 1/3]`

`j = 2` `k = 2` `d = -3`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2	2/3	1/3
g(p)	0	1	·	·	·

max tries a candidate: g(1) = 1. Best so far.

Step 8 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
6              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
7              if d != 0:
8                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
▶      p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

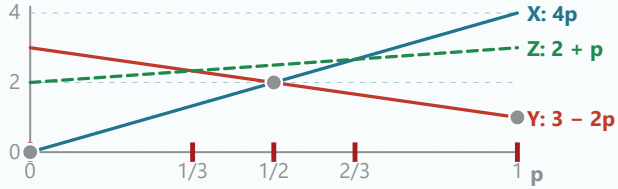
VARIABLES

`n = 3`

`cand = [0, 1, 1/2, 2/3, 1/3]`

`j = 2` `k = 2` `d = -3`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2	2/3	1/3
g(p)	0	1	2	·	·

max tries a candidate: g(1/2) = 2. Best so far.

Step 9 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
6              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
7              if d != 0:
8                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
▶      p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

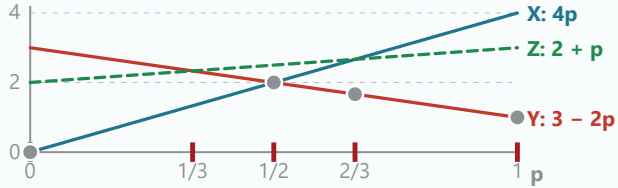
VARIABLES

`n = 3`

`cand = [0, 1, 1/2, 2/3, 1/3]`

`j = 2` `k = 2` `d = -3`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2	2/3	1/3
g(p)	0	1	2	5/3	·

max tries a candidate: g(2/3) = 5/3.

Step 10 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
6              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
7              if d != 0:
8                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
▶      p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

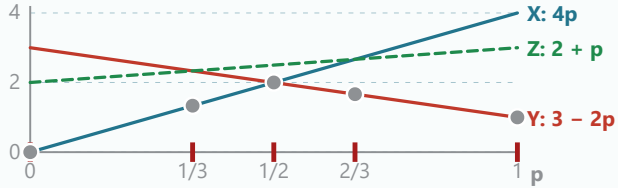
VARIABLES

`n = 3`

`cand = [0, 1, 1/2, 2/3, 1/3]`

`j = 2` `k = 2` `d = -3`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2	2/3	1/3
g(p)	0	1	2	5/3	4/3

max tries a candidate: g(1/3) = 4/3.

Step 11 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
6              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
7              if d != 0:
8                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
▶      p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

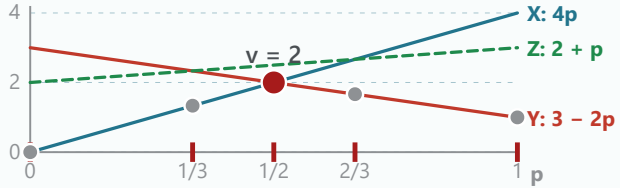
VARIABLES

`n = 3`

`cand = [0, 1, 1/2, 2/3, 1/3]`

`j = 2` `k = 2` `d = -3`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2	2/3	1/3
g(p)	0	1	2	5/3	4/3

The largest guarantee is at p = 1/2: **v = 2**. Column Z's line stays above the envelope, so the column player never plays Z.

Step 12 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
6              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
7              if d != 0:
8                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
▶      p = max((c for c in cand if 0 <= c <= 1), key=g)
11     return p, g(p)
```

VARIABLES

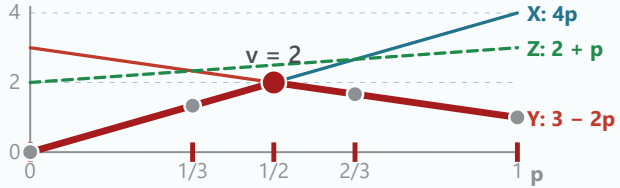
`n = 3`

`cand = [0, 1, 1/2, 2/3, 1/3]`

`j = 2` `k = 2` `d = -3`

`p = 1/2`

Code Trace: `maximin(A)` — 2 × n Zero-Sum



candidate p	0	1	1/2	2/3	1/3
g(p)	0	1	2	5/3	4/3

Returns (1/2, 2). The minimax side: if the column player plays X with 1/3 and Y with 2/3, the row player gets at most 2 ⇒ max-min = min-max = 2.

Step 13 / 13

```
1  from fractions import Fraction as F
2  def maximin(A):                    # A: 2 x n
3      n, cand = len(A[0]), [F(0), F(1)]
4      for j in range(n):
5          for k in range(j + 1, n):
6              d = A[0][j] - A[1][j] - A[0][k] + A[1][k]
7              if d != 0:
8                  cand.append(F(A[1][k] - A[1][j], d))
9      g = lambda p: min(p*a+(1-p)*b for a, b in zip(*A))
10     p = max((c for c in cand if 0 <= c <= 1), key=g)
▶      return p, g(p)
```

VARIABLES

`n = 3`

`cand = [0, 1, 1/2, 2/3, 1/3]`

`j = 2` `k = 2` `d = -3`

`p = 1/2` `return = (1/2, 2)`

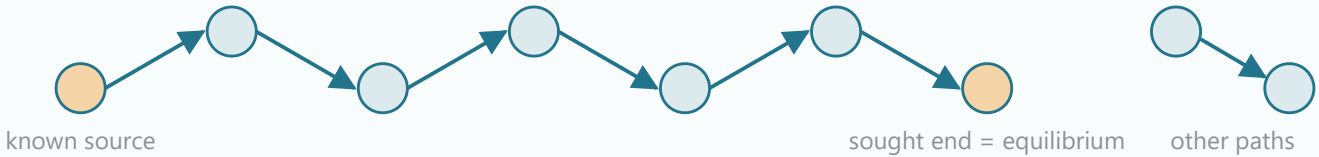
How Hard Is It to Compute an Equilibrium?

Problem	Complexity	Method / source
Pure Nash equilibrium (matrix)	Polynomial	Best-response marking, O(mn)
Two-player zero-sum, mixed	Polynomial	Linear programming (minimax)
Two-player general game, mixed	**PPAD-complete**	Chen–Deng (2006)
3+ players, exact	**FIXP-complete**	Etessami–Yannakakis (2007)
3+ players, ε-approximate (small ε)	**PPAD-complete**	Daskalakis–Goldberg–Papadimitriou (2006)
Max-welfare equilibrium / unique?	NP-hard	Gilboa–Zemel (1989); Conitzer–Sandholm (2008)

- **Support enumeration** and Lemke–Howson work in practice but are exponential in the worst case.
- With three players an equilibrium can be **irrational**, so we seek ε-approximate ones (deviating gains at most ε). Two-player games with rational payoffs have rational equilibria.
- Further reading: for fixed ε (e.g. 0.1), two-player games admit a quasi-polynomial algorithm (Lipton–Markakis–Mehta, 2003); multiplayer games stay PPAD-hard (Rubinstein, 2015).

PPAD: Finding a Solution Known to Exist

- A Nash equilibrium **always exists** → "is there an equilibrium?" is trivial; NP-completeness does not fit problems whose solution is guaranteed.
- **PPAD** (Papadimitriou, 1994): the class of problems whose solution is known to exist, but finding it may require following a very long path step by step.



- Analogy: walk from the start of a path looking for its end; the end surely exists, but the path may be exponentially long.
- Finding a Nash equilibrium is in this class. Consequence: in general games equilibria may not be efficiently computable → **do players actually "find" equilibria?**

In Practice: Approximate Equilibria and Software

- **ε-Nash equilibrium**: with payoffs scaled to [0, 1], no player can gain more than ε by deviating unilaterally. Real-time systems (ad auctions, markets) often settle for this.
- An exact equilibrium may be hard to find, but a profile where "almost nobody wants to deviate" is often found quickly; even simple profiles mixing a few strategies approximate well.
- **Further reading:** a ½-approximate equilibrium built from two best responses and a 50–50 mix (Daskalakis–Mehta–Papadimitriou, 2006); small-support ε-equilibria (Lipton–Markakis–Mehta, 2003).

```python
import nashpy as nash
A = [[2, 2], [0, 4]]    # worker: Work / Shirk
B = [[1, 2], [-1, -4]]  # manager: Inspect / Don't
game = nash.Game(A, B)
for x, y in game.support_enumeration():
    print(x, y)  # [0.75 0.25] [0.5 0.5]
```

The inspection game in Nashpy: support enumeration returns p* = 3/4, q* = 1/2. Gambit does the same (support enumeration, Lemke–Howson, LP) with a GUI.

There are other equilibrium concepts too, e.g. **correlated equilibrium** and, in sequential games, **subgame perfect equilibrium**.

SUMMARY

Discussion: Equilibrium, Fairness and Rationality

Is an equilibrium fair?

Not necessarily. It is only a set of mutual best responses: payoffs are unequal in Battle of the Sexes, and the Prisoner's Dilemma has an outcome better for everyone. *Fairness ≠ equilibrium*.

Are people rational?

Real decisions are shaped by missing information, time pressure, emotions and social norms (bounded rationality, Herbert Simon). The model is a clear benchmark for studying strategic interdependence.

Ethical limits

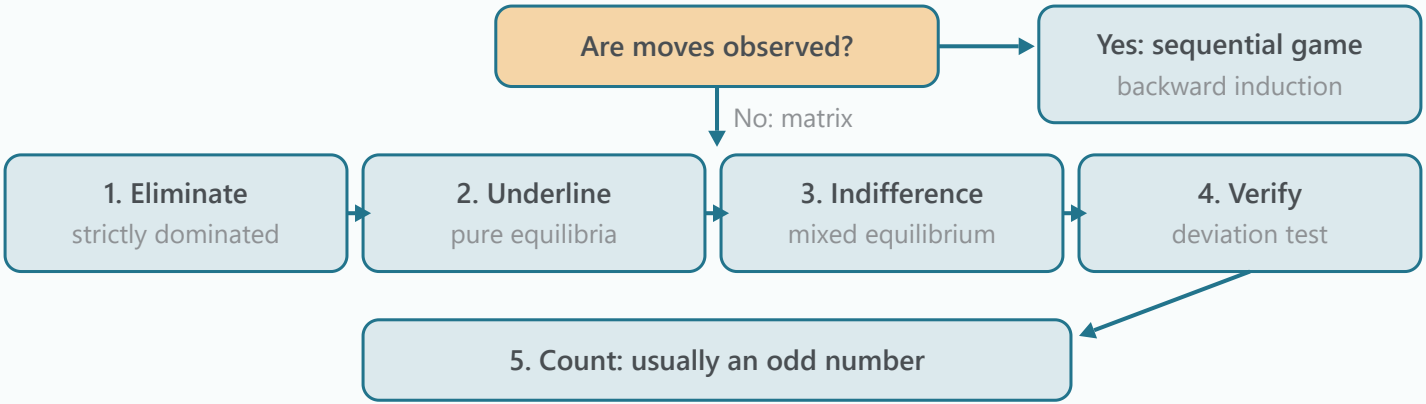
Deception, manipulation, unfair advantage… "Anything to win?" Rules and institutions set the limits of strategic behavior — which is the subject of mechanism design.

Summary

Concept	What it says	How to find it
Nash equilibrium	Nobody wants to deviate alone	Mutual best responses
Eliminating dominated strategies	A rational player never plays a dominated strategy	Iteratively delete strictly dominated ones
Mixed equilibrium	Strategies in the support pay equally	Indifference equations
Existence (Nash 1950)	Every finite game has a mixed equilibrium	Kakutani / Brouwer fixed point
Multiple equilibria	Theory may not pick a single outcome	Payoff / risk dominance, focal points
Correlated equilibrium	Following the mediator is in everyone's interest	Obedience constraints + linear programming
Minimax	Zero-sum: max-min = min-max = v	Saddle point or LP
Complexity	Exact (2 players) and approximate (many players) equilibria are PPAD-complete	Lemke–Howson, support enumeration

SUMMARY

Checklist: Solving a Game Step by Step



Read first

Who is row, who is column? Whose payoff comes first in a cell? Are the numbers payoffs or costs (years, minutes)?

Then write

Write each equilibrium as a profile: (Middle, Left) = (3, 5); for a mixed one, give both players' probabilities and expected payoffs.

SUMMARY

Test Yourself (1/2)

Game A

	Left	Middle	Right
Top	(3 , 1)	(1 , 2)	(2 , 0)
Bottom	(0 , 2)	(2 , 3)	(1 , 1)

Game B	Left	Right
Top	(2, 1)	(0, 0)
Bottom	(0, 0)	(1, 2)

1. Does the column player have a dominant strategy in Game A?
2. Find Game A's pure Nash equilibria by underlining.
3. How many equilibria does Game B have? In the mixed one, what are p = P(Top) and q = P(Left)?
4. In Game B, are the mixed-equilibrium payoffs better than the pure ones?

▶ **Show answers**

SUMMARY

Test Yourself (2/2)

True or false?

1. A strictly dominated strategy can be played in a Nash equilibrium.
2. In a mixed equilibrium, the row player's probability is computed from their own payoffs.
3. A subgame perfect equilibrium (found by backward induction on the game tree) is always a Nash equilibrium.
4. In a three-player game, joint deviations by two players are also checked.

Sequential game

5. Player 1 chooses Left or Right. Right → the game ends with (1, 1). Left → Player 2 chooses a → (3, 0) or b → (0, 2). Find the equilibrium and outcome by backward induction.
6. If Player 2 promises “I'll choose a if you go Left”, should Player 1 trust it?

▶ **Show answers**

Sample Problems

Game 1	L	M	R
U	(2, 1)	(1, 2)	(0, 0)
D	(3, 0)	(0, 1)	(1, 3)

Game 2	L	R
U	(4, 1)	(0, 3)
D	(1, 2)	(2, 0)

1. Game 1: pure Nash equilibria by marking best responses.
2. Game 2: the mixed equilibrium via indifference.
3. Zero-sum A = [[3, −1, 2], [1, 0, 4], [−2, −3, 5]]: saddle point? Value?
4. “Every Nash equilibrium is Pareto optimal.” True?
5. Why is existence stated for mixed strategies?
6. Cournot, P = 15 − Y, zero cost: equilibrium quantities? In an equal-split cartel, each firm's output, and why is it not an equilibrium?

▶ **Show answers**

Exercises (1/2)

Game 1	Left	Right
Top	(4, 3)	(1, 2)
Bottom	(3, 1)	(2, 4)

Game 2	0	1	2
0	(2, 5)	(1, 3)	(3, 1)
1	(5, 2)	(3, 3)	(1, 2)
2	(0, 0)	(5, 5)	(0, 0)

1. Does Game 1 have a dominant strategy? Find its pure Nash equilibria by marking best responses.
2. In Game 2 (row i, column j), iteratively delete strictly dominated strategies. What remains? Compare with the pure Nash equilibrium.
3. In RPS your opponent plays Rock and Paper 50–50 (never Scissors). What is your best response and expected payoff?

▶ **Show answers**

Exercises (2/2)

4. Find the maximin and minimax of the zero-sum matrices A = [[3, 1, 4], [2, 0, 1], [5, 2, 3]] and B = [[0, 2], [3, 1]]. Which has a saddle point?
5. (X, X) = (4, 1), (Y, Y) = (1, 4), other cells (0, 0). In the mixed equilibrium, what are p* = P(row X), q* = P(column X) and the probability of failing to meet?
6. In Chicken, if a crash cost −20 instead of −10, what would the swerve and crash probabilities be in the mixed equilibrium?
7. In Stag Hunt (stag 4, hare 3), if a hare paid 2 instead of 3, what would the stag probability be in the mixed equilibrium? Does trust get easier?

▶ **Show answers**

References

- J. F. Nash, “Equilibrium points in n-person games”, *PNAS* 36(1), 1950; “Non-cooperative games”, *Annals of Mathematics* 54(2), 1951.
- J. von Neumann, “Zur Theorie der Gesellschaftsspiele”, *Mathematische Annalen* 100, 1928.
- M. J. Osborne, *An Introduction to Game Theory*, Oxford University Press, 2004 (Chapters 2–4).
- N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani (eds.), *Algorithmic Game Theory*, Cambridge, 2007 (Chapters 1–3).
- T. Roughgarden, *Twenty Lectures on Algorithmic Game Theory*, Cambridge, 2016 (Lectures 13, 18 and 20).
- C. Daskalakis, P. Goldberg, C. Papadimitriou, “The complexity of computing a Nash equilibrium”, *SIAM J. Computing* 39(1), 2009.
- K. Etessami, M. Yannakakis, “On the complexity of Nash equilibria and other fixed points”, *SIAM J. Computing* 39(6), 2010 · R. J. Lipton, E. Markakis, A. Mehta, “Playing large games using simple strategies”, *ACM EC*, 2003 · A. Rubinstein, “Inapproximability of Nash equilibrium”, *STOC*, 2015.

References (2/2)

- J. C. Harsanyi, R. Selten, *A General Theory of Equilibrium Selection in Games*, MIT Press, 1988.
- A. R. Karlin, Y. Peres, *Game Theory, Alive*, AMS, 2017 (zero-sum and general games, Nash's theorem).
- D. Easley, J. Kleinberg, *Networks, Crowds, and Markets*, Cambridge, 2010 (Chapter 6: Games).
- M. Tambe, *Security and Game Theory*, Cambridge University Press, 2011 (security games, ARMOR).
- D. Sally, "Conversation and cooperation in social dilemmas: a meta-analysis of experiments from 1958 to 1992", *Rationality and Society* 7(1), 1995 · R. Wilson, "Computing equilibria of n-person games", *SIAM J. Applied Mathematics* 21, 1971.
- Course pages: **Roughgarden CS364A** · **Shah CSC304** · Software: **Gambit**, **Nashpy** · Video: **Game Theory 101**

The End

Chapter 3: Nash Equilibrium

Previous: Chapter 2: Games and Representations · Next: Chapter 4: Mechanism Design