Enumerative Combinatoric Algorithms

&

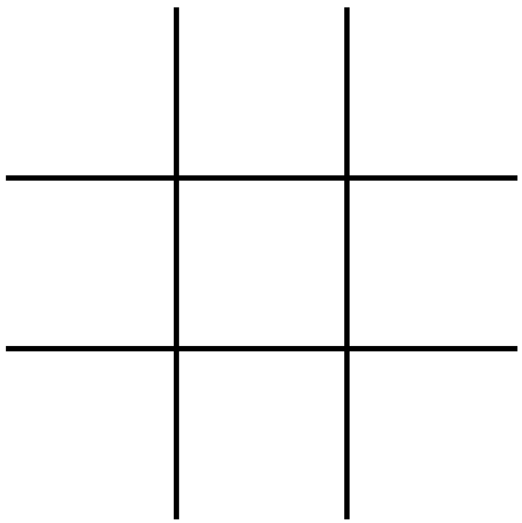
Algorithms and Games

Combinatorial 2 Player Games

Connect-4 aka 4 Gewinnt
Nine Men's Morris aka Mühle
Hexapawn
and more to come ...

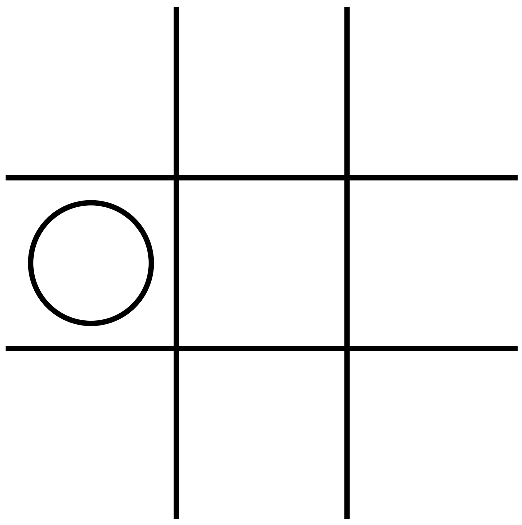
Example: Tik Tak Toe

Tic Tac Toe: Two players play in turns, placing one of their stones (o starts, $\mathtt{x}$ follows) on a 3×3 grid. The player who first has three stones in a line (horizontal, vertical, or diagonal) wins.



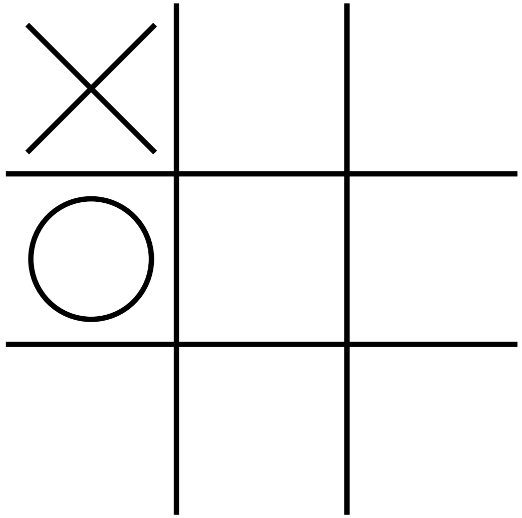
Example: Tik Tak Toe

Tic Tac Toe: Two players play in turns, placing one of their stones (o starts, $\mathtt{x}$ follows) on a 3×3 grid. The player who first has three stones in a line (horizontal, vertical, or diagonal) wins.



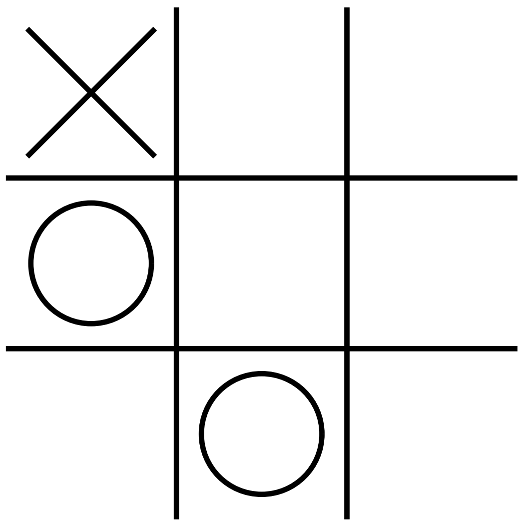
Example: Tik Tak Toe

Tic Tac Toe: Two players play in turns, placing one of their stones (o starts, $\mathtt{x}$ follows) on a 3×3 grid. The player who first has three stones in a line (horizontal, vertical, or diagonal) wins.



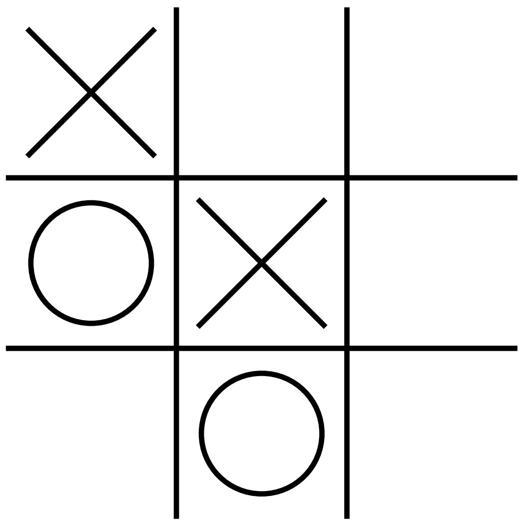
Example: Tik Tak Toe

Tic Tac Toe: Two players play in turns, placing one of their stones (o starts, $\mathtt{x}$ follows) on a 3×3 grid. The player who first has three stones in a line (horizontal, vertical, or diagonal) wins.



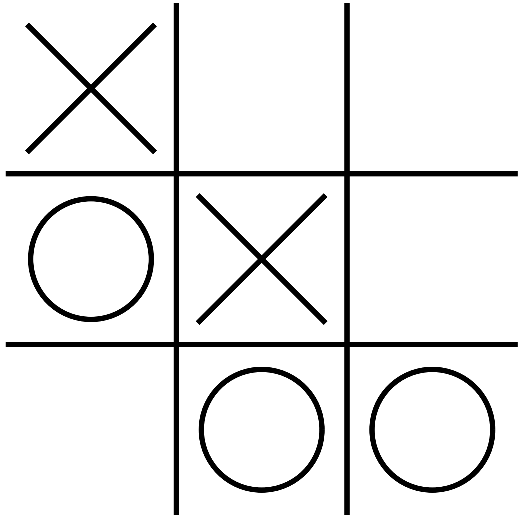
Example: Tik Tak Toe

Tic Tac Toe: Two players play in turns, placing one of their stones (o starts, $\mathtt{x}$ follows) on a 3×3 grid. The player who first has three stones in a line (horizontal, vertical, or diagonal) wins.



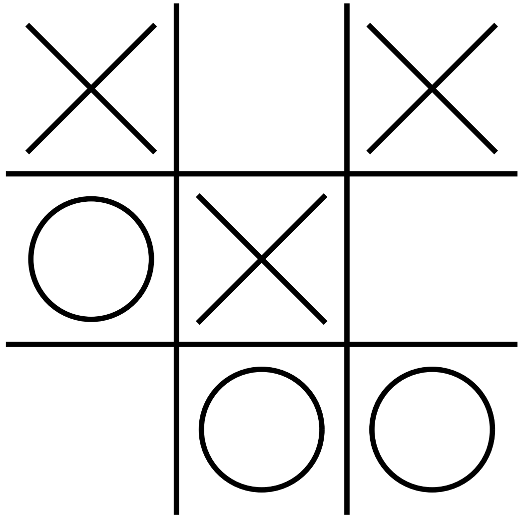
Example: Tik Tak Toe

Tic Tac Toe: Two players play in turns, placing one of their stones (o starts, $\mathtt{x}$ follows) on a 3×3 grid. The player who first has three stones in a line (horizontal, vertical, or diagonal) wins.



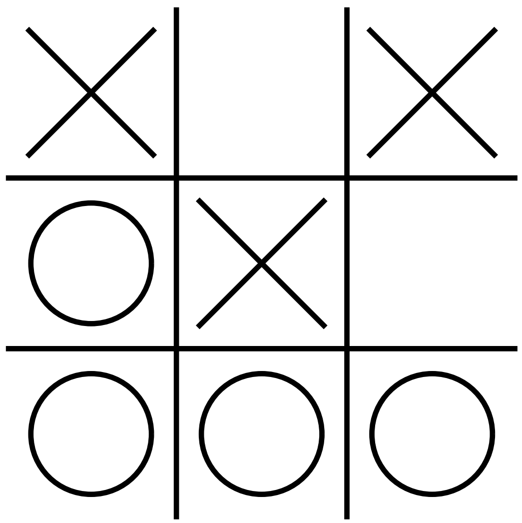
Example: Tik Tak Toe

Tic Tac Toe: Two players play in turns, placing one of their stones (o starts, $\mathtt{x}$ follows) on a 3×3 grid. The player who first has three stones in a line (horizontal, vertical, or diagonal) wins.



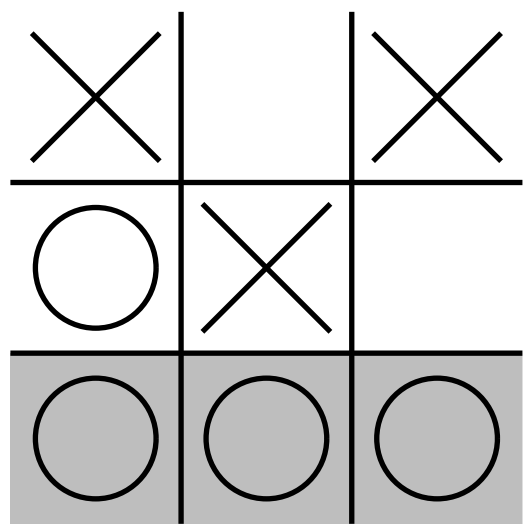
Example: Tik Tak Toe

Tic Tac Toe: Two players play in turns, placing one of their stones (o starts, $\mathtt{x}$ follows) on a 3×3 grid. The player who first has three stones in a line (horizontal, vertical, or diagonal) wins.



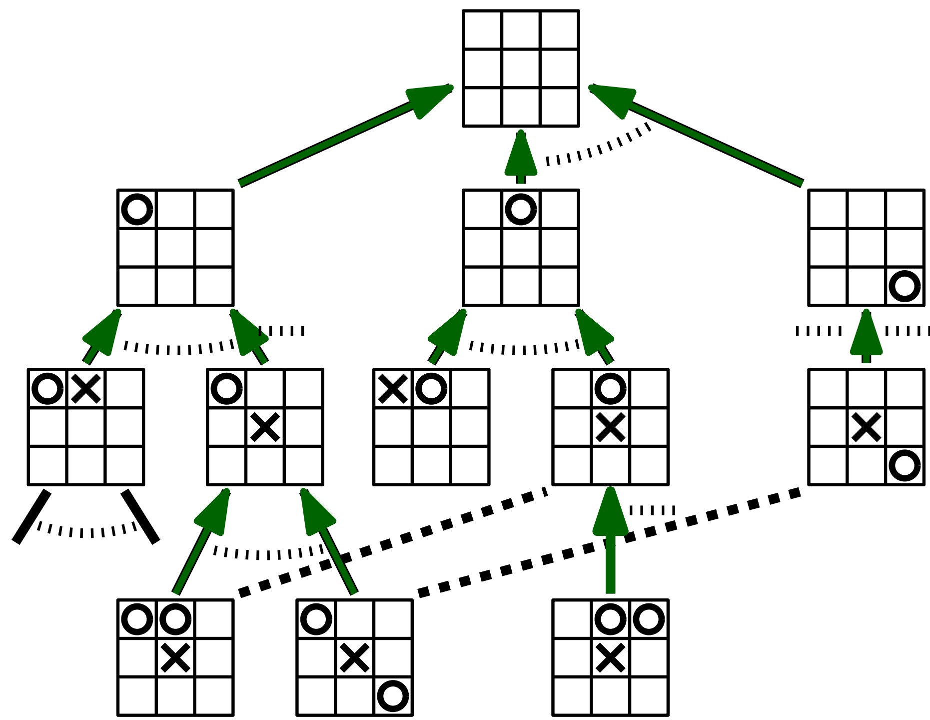
Example: Tik Tak Toe

Tic Tac Toe: Two players play in turns, placing one of their stones (o starts, $\mathtt{x}$ follows) on a 3×3 grid. The player who first has three stones in a line (horizontal, vertical, or diagonal) wins.



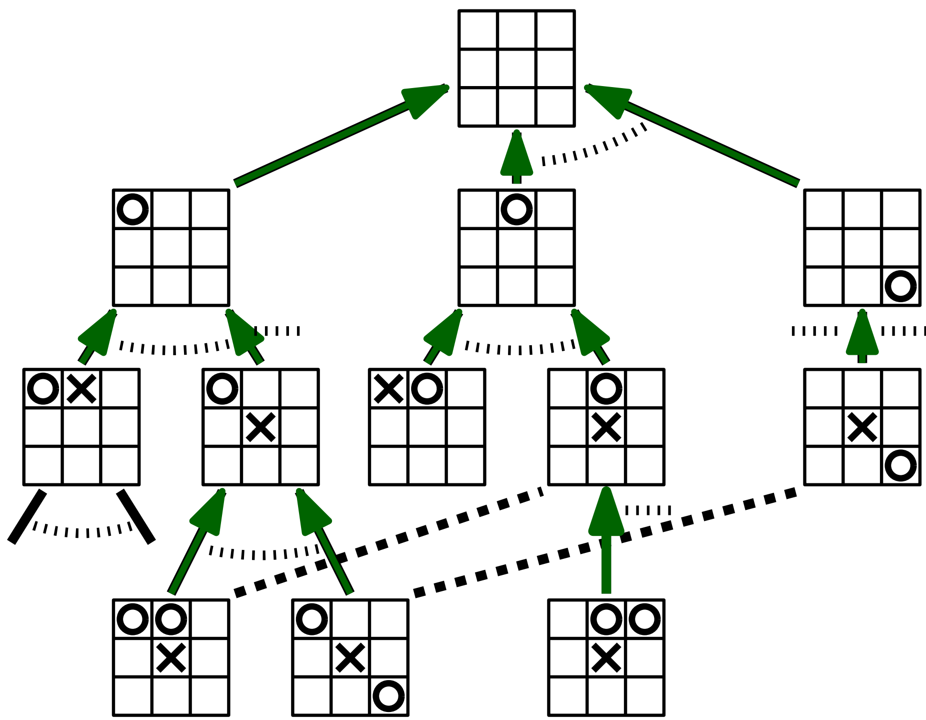
Example: Tik Tak Toe

- Games are often represented as decision trees
- In Contrast: Enumerate all **different** valid game positions



Example: Tik Tak Toe

Enumerate all **different** valid game positions of Tic Tac Toe after k half-moves, **considering symmetries**.
o starts, $\mathtt{x}$ follows $\rightarrow \lfloor \frac{k+1}{2} \rfloor$ o, and $\lfloor \frac{k}{2} \rfloor$ $\mathtt{x}$ tokens are placed after k half-moves.



Example: Tik Tak Toe

Storing a board:

2 bit per square:

$2 \times 9 = 18$ bit, thus $2^{18} = 262144$ possible boards.

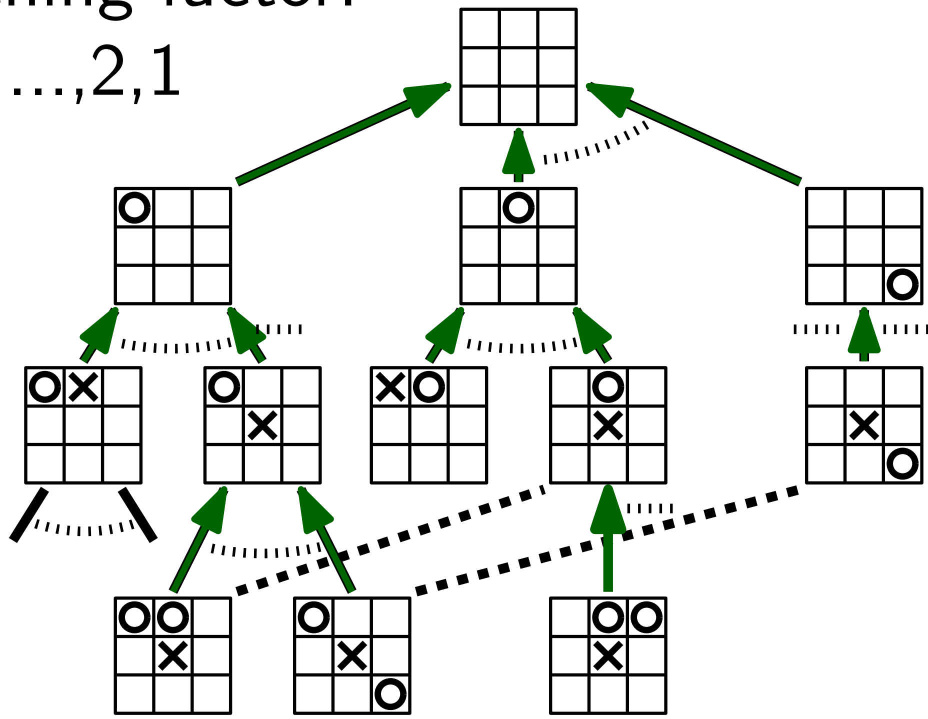
Example: Tik Tak Toe

Storing a board:
2 bit per square:
$2 \times 9 = 18$ bit, thus $2^{18} = 262144$ possible boards.
3 possibilities per square:
$3^9 = 19683$ possible boards with $\lceil \log_2 3^9 \rceil = 15$ bit.

Example: Tik Tak Toe

n half-moves	game-tree	different boards
0	1	
1	9	
2	72	
3	504	
4	3024	
5	15120	
6	60480	
7	181440	
8	362880	
9	362880	
sum	986410	

branching factor:
9,8,7,...,2,1



Example: Tik Tak Toe

n half-moves	game-tree	different boards
0	1	1
1	9	3
2	72	12
3	504	38
4	3024	108
5	15120	174
6	60480	228
7	181440	174
8	362880	89
9	362880	23
sum	986410	850

Example: Tik Tak Toe

n half-moves	game-tree	different boards
0	1	1
1	9	3
2	72	12
3	504	38
4	3024	108
5	15120	174
6	60480	228
7	181440	174
8	362880	89
9	362880	23
sum	986410	850

Only 3 without winning line

Example: Tik Tak Toe

n half-moves	game-tree	different boards	
0	1	1	
1	9	3	
2	72	12	
3	504	38	
4	3024	108	
5	15120	174	
6	60480	228	
7	181440	174	
8	362880	89	
9	362880	23	Only 3 without winning line
sum	986410	850	Actually only 765 states when not continued after win

Example: Tik Tak Toe

n half-moves	game-tree	different boards
0	1	1
1	9	3
2	72	12
3	504	38
4	3024	108
5	15120	174
6	60480	228
7	181440	174
8	362880	89
9	362880	23
sum	986410	850

- $986410 =$ game-tree complexity
- $262144 = 2^{18}$
- $19683 = 3^9$
- 850 different boards $=$ state space complexity

Basics on 2 Player Games

- We consider 2 player perfect information games. The players are called *first player* (**A**lice) and *second player* (**B**ob).
- No hidden information, no randomness or chance, both players have all information.
- Players play in turns (not simultaniously).
- There is a finite number of game states (but a game might last forever which is considered a draw).
- Note that the game might be asymmetric, i.e., Alice and Bob have different tasks (e.g. Fox and Geese).

Basics for Computer Playing

- A game state needs to be stored memory efficient and complete (coding and decoding of states including player information etc.).
- Move generator (successors of a game state).
- Identify final states: win, lose, and draw states.
- Backwards move generator (predecessors of a game state). Not always possible.

Game states are important!

Different game states?

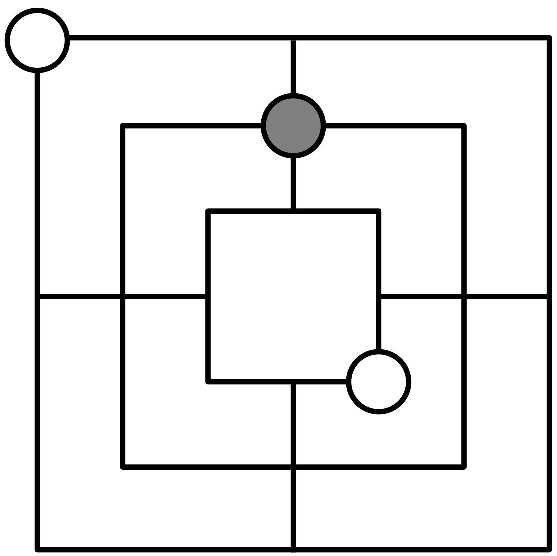
Two game states are equivalent, if they allow the same moves (w.r.t. the state), resulting in the same successor states (w.r.t. the state). Typically reflection, rotation, inversion, color-change, ... can be applied.

We only need to store the move information for one of the equivalent states (canonical state, fingerprint), as for all other states it follows by the equivalence operations.

Nine Men's Morris (aka Mühle)

Enumerate all **different** valid game positions for Nine Men's Morris (number of non-equivalent states)...

How many non-equivalent states exist after 2 white tokens and 1 black token have been placed (white player starts)?



(perfect play always results in a draw)

Nine Men's Morris (aka Mühle)

How many non-equivalent states exist after 2 white tokens and 1 black token have been placed (white player starts)?

Pólya-Redfield Enumeration Theorem: 16 Operations:

R_0: ID: $r_0 = \binom{24}{2} \times 22 = 6072$

R_1 Rotation $90°$ (R_3 Rotation $270°$): $r_1 = r_3 = 0$

R_2 Rotation $180°$: $r_2 = 0$

$R_4 \dots R_7$ Reflections: $r_4 = \dots = r_7 = 6 \times (9 + \binom{5}{2}) = 114$

R_8: In-Out Inversion: $r_8 = 8 \times (8 + \binom{7}{2}) = 232$

$R_9 \dots R_{15}$: In-Out-Inversion plus $R_1 \dots R_7$

$r_9 = r_{10} = r_{11} = 0$

$r_{12} = \dots = r_{15} = 2 \times 11 = 22$

Number of orbits$= \frac{6072+4\times 114+232+4\times 22}{16} = \frac{6848}{16} = 428$

Nine Men's Morris (aka Mühle)

How many non-equivalent states exist after 2 white tokens and 1 black token have been placed (white player starts)?

Pólya-Redfield Enumeration Theorem: 16 Operations:

R_0: ID: $r_0 = \binom{24}{2} \times 22 = 6072$

R_1 Rotation $90°$ (R_3 Rotation $270°$): $r_1 = r_3 = 0$

R_2 Rotation $180°$: $r_2 = 0$

$R_4 \ldots R_7$ Reflections: $r_4 = \ldots = r_7 = 6 \times (9 + \binom{5}{2}) = 114$

R_8: In-Out Inversion: $r_8 = 8 \times (8 + \binom{7}{2}) = 232$

$R_9 \ldots R_{15}$: In-Out-Inversion plus $R_1 \ldots R_7$

$r_9 = r_{10} = r_{11} = 0$

$r_{12} = \ldots = r_{15} = 2 \times 11 = 22$

Number of orbits$= \frac{6072+4\times114+232+4\times22}{16} = \frac{6848}{16} = 428$

Nine Men's Morris (aka Mühle)

How many non-equivalent states exist after 2 white tokens and 1 black token have been placed (white player starts)?

Pólya-Redfield Enumeration Theorem: 16 Operations:

R_0: ID: $r_0 = \binom{24}{2} \times 22 = 6072$

R_1 Rotation 90° (R_3 Rotation 270°): $r_1 = r_3 = 0$

R_2 Rotation 180°: $r_2 = 0$

$R_4 \ldots R_7$ Reflections: $r_4 = \ldots = r_7 = 6 \times (9 + \binom{5}{2}) = 114$

R_8: In-Out Inversion: $r_8 = 8 \times (8 + \binom{7}{2}) = 232$

$R_9 \ldots R_{15}$: In-Out-Inversion plus $R_1 \ldots R_7$

$r_9 = r_{10} = r_{11} = 0$

$r_{12} = \ldots = r_{15} = 2 \times 11 = 22$

$24 * 23 * 22 =$
12144 games

Number of orbits$= \frac{6072+4\times114+232+4\times22}{16} = \frac{6848}{16} = 428$

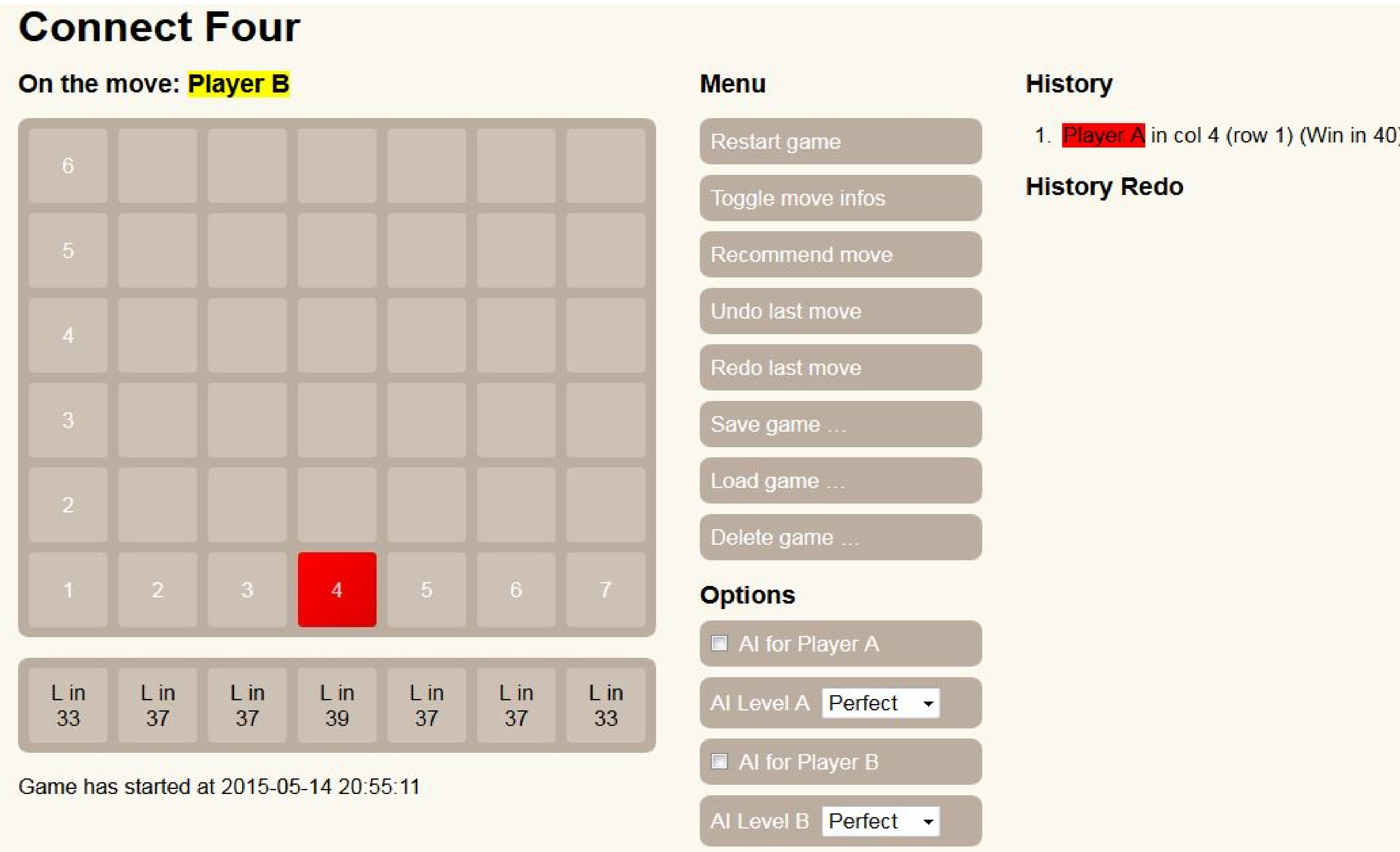
Levels of Game Solutions

- Ultra-weakly solved: We know which player can win, but not how (no strategy! Example: Chomp)
- Weakly solved: A strategy is known (from a start situation following the strategy)
- Strongly solved: A strategy is known from any valid state.
- **Ultra-strongly solved**: For any valid game state and any possible move it is known whether it is a win, draw or lose and in how many half-moves this happens.

We aim for ultra-strongly solved!

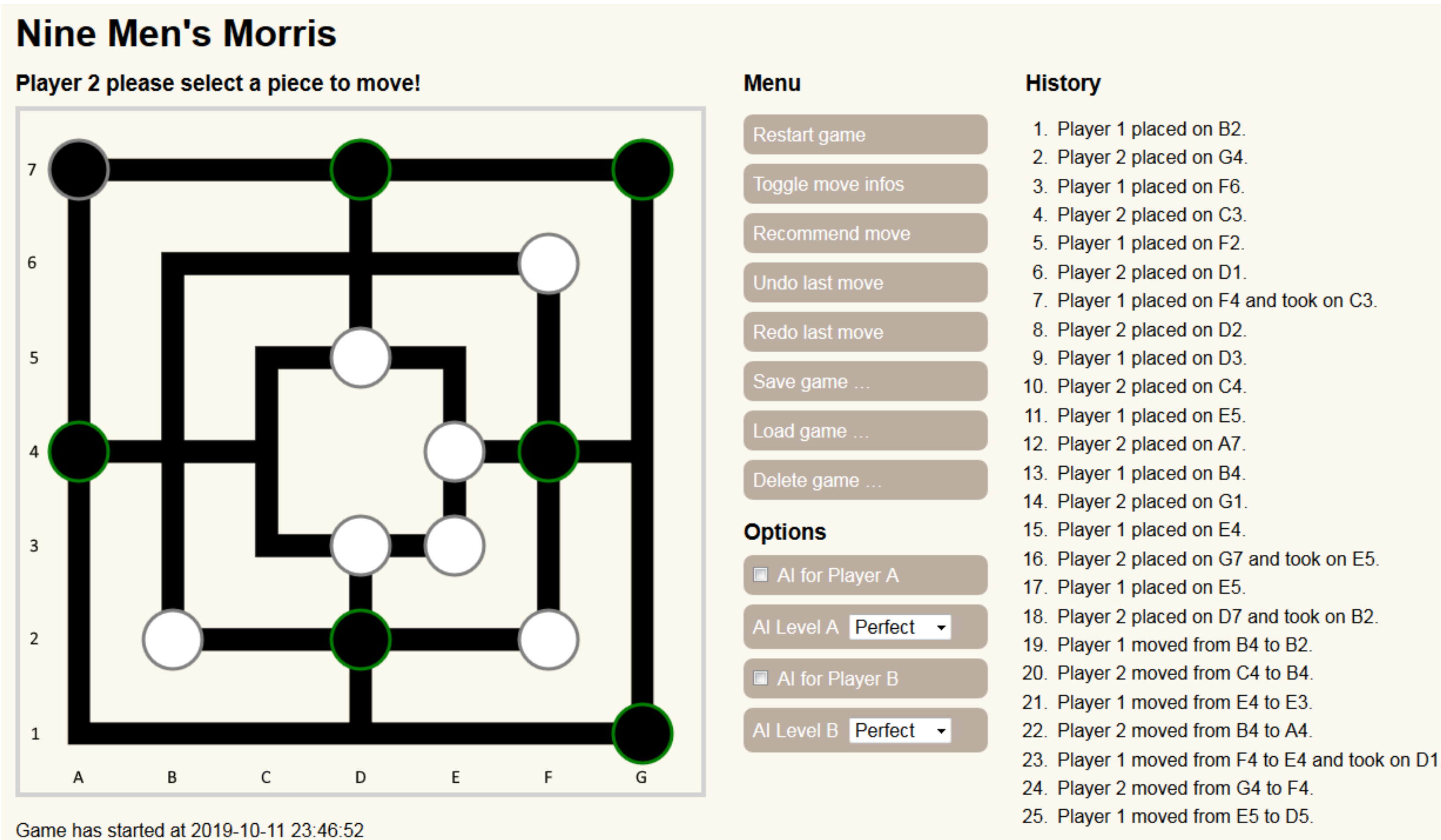
Connect-4

http://connect4.ist.tugraz.at:8080



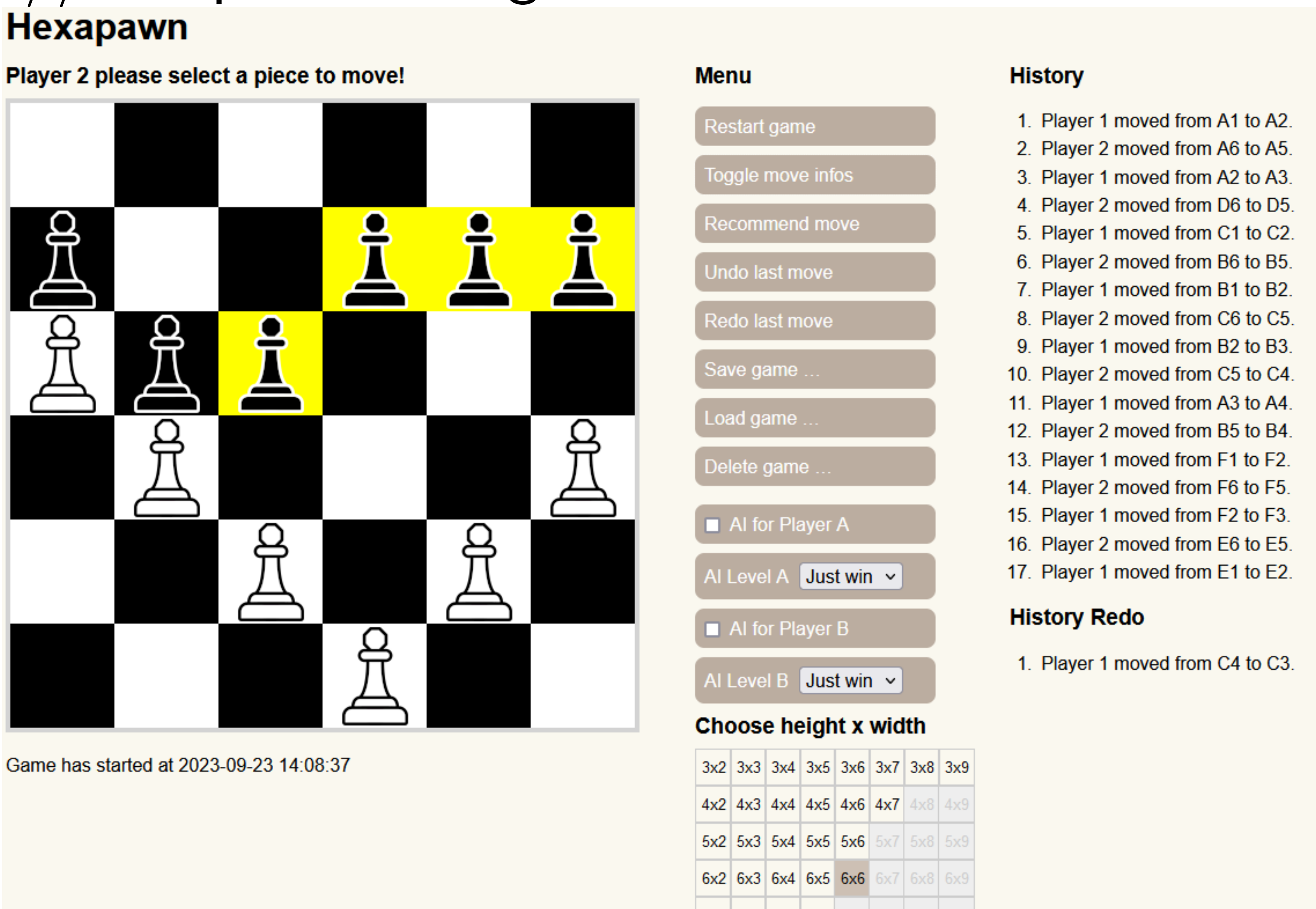
Nine Men's Morris

http://ninemensmorris.ist.tugraz.at:8080



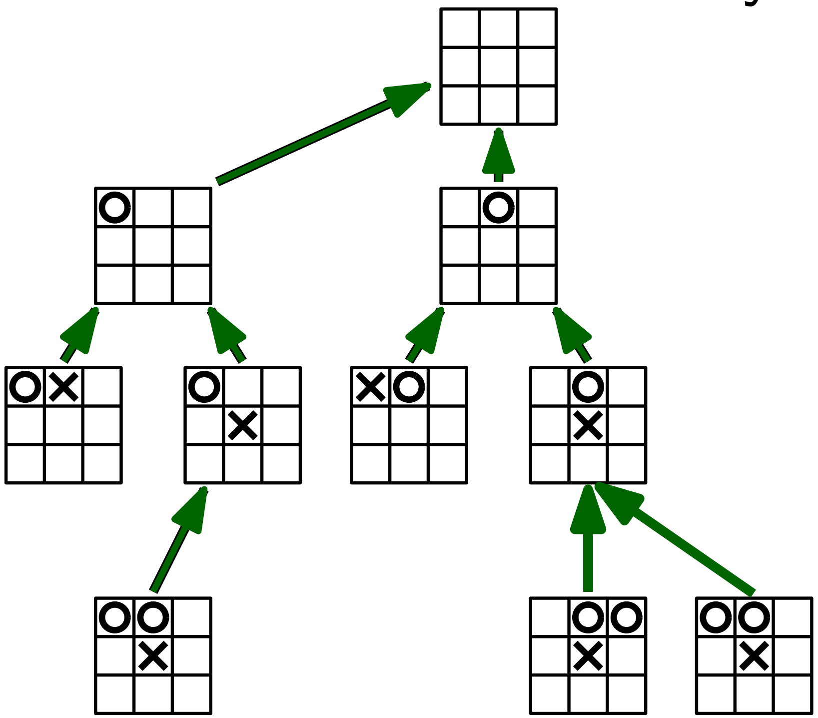
Hexapawn

http://hexapawn.ist.tugraz.at



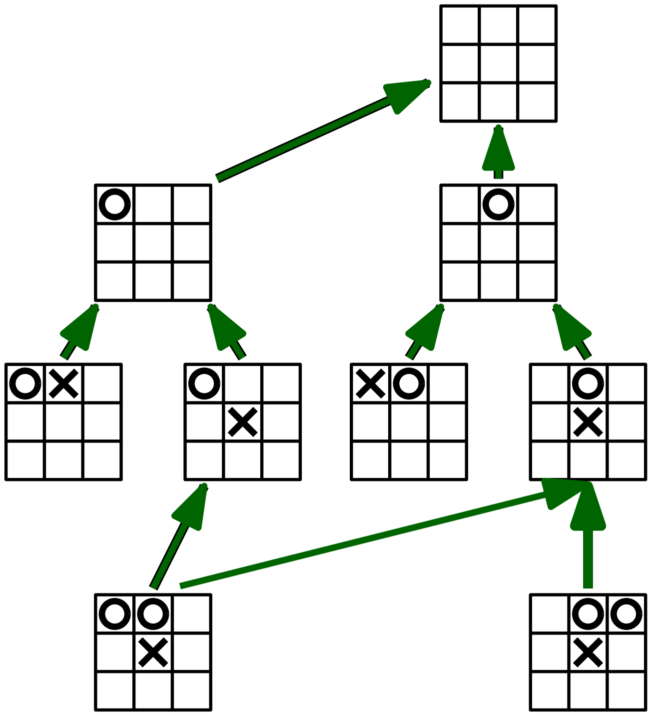
Game-Tree vs. State-Space Complexity

- **Game-Tree Complexity** Number of nodes the complete decision tree for a whole game has
- **State-Space Complexity** Number of states which can be reached from the start state by valid moves



Game-Tree vs. State-Space Complexity

- **Game-Tree Complexity** Number of nodes the complete decision tree for a whole game has
- **State-Space Complexity** Number of states which can be reached from the start state by valid moves



Game-Tree vs. State-Space Complexity

- **Game-Tree Complexity** Number of nodes the complete decision tree for a whole game has
- **State-Space Complexity** Number of states which can be reached from the start state by valid moves

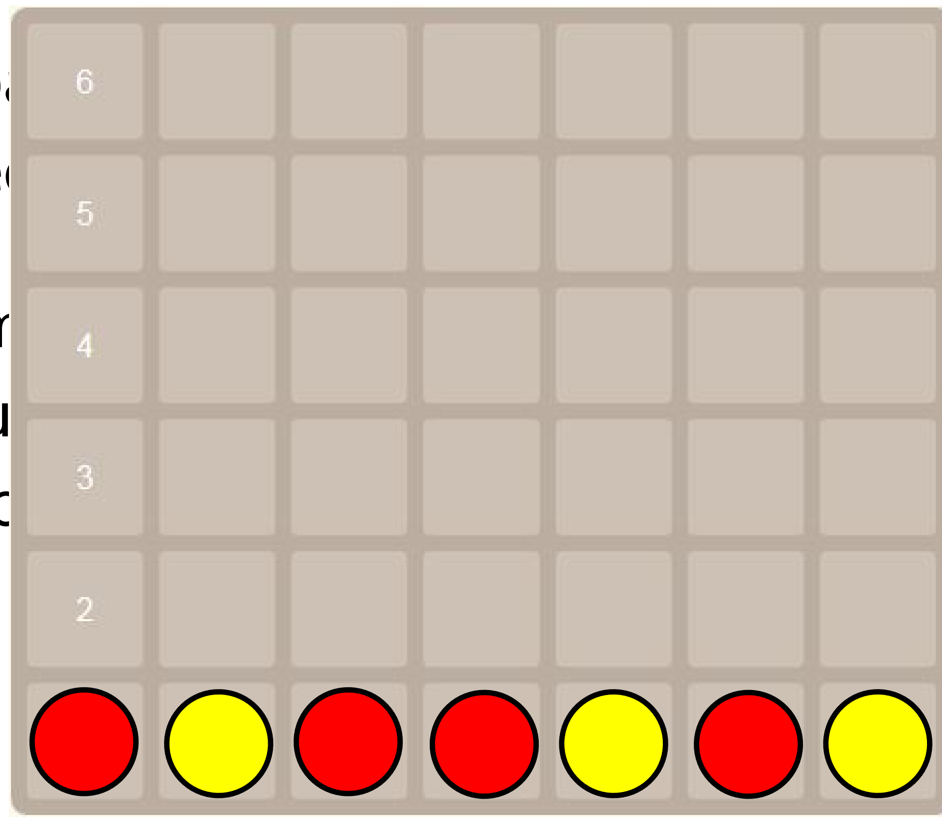
For most games a state might be reachable via many different sequences of valid moves. Cycles might even result in unbounded possibilities.

Game-Tree vs. State-Space Complexity

- **Game-Tree Complexity** Number of nodes the complete decision tree for a whole game has

- **State-Sp** ...tates which can be reache... moves

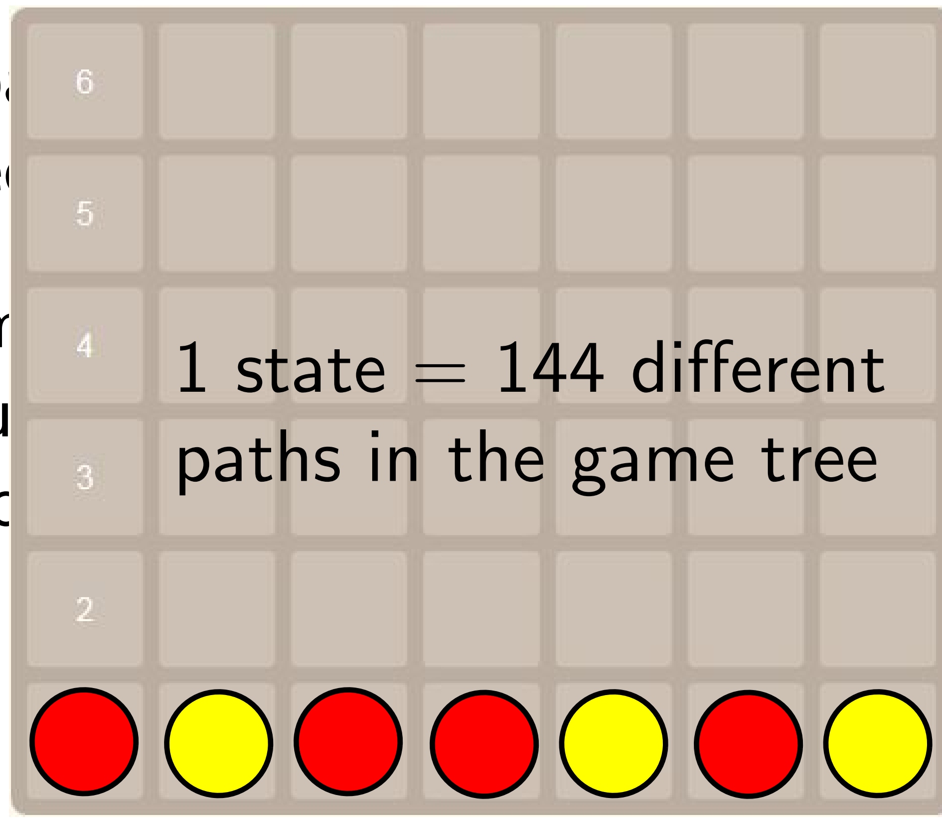
For most gam... via many different sequ... night even result in unbo...



Game-Tree vs. State-Space Complexity

- **Game-Tree Complexity** Number of nodes the complete decision tree for a whole game has
- **State-Sp**... ...tates which can be reache... ...moves

For most gam... ...via many different sequ... ...night even result in unbo...



Game-Tree vs. State-Space Complexity

- **Game-Tree Complexity** Number of nodes the complete decision tree for a whole game has
- **State-Space Complexity** Number of states which can be reached from the start state by valid moves

game	state-space complexity	game-tree complexity	branching factor
Tic Tac Toe	10^3	10^5	5
Nine Men's Morris	10^{10}	10^{50}	10-30
Pyraos	10^{11}	10^{33}	9
Awari	10^{12}	10^{32}	5-6
Connect-4	10^{14}	10^{21}	5-7
Abalone	10^{25}	10^{180}	65-70
Reversi	10^{28}	10^{58}	5-15
Chess	10^{50}	10^{123}	35
Go	10^{171}	10^{360}	300-400

Game-Tree vs. State-Space Complexity

- **Game-Tree Complexity** Number of nodes the complete decision tree for a whole game has
- **State-Space Complexity** Number of states which can be reached from the start state by valid moves

game	state-space complexity	game-tree complexity	branching factor
Tic Tac Toe	10^3	10^5	5
Nine Men's Morris	10^{10}	10^{50}	10-30
Pyraos	10^{11}	10^{33}	9
Awari	10^{12}	10^{32}	5-6
Connect-4	10^{14}	10^{21}	5-7
Abalone	10^{25}	10^{180}	65-70
Reversi	10^{28}	10^{58}	5-15
Chess	10^{50}	10^{123}	35
Go	10^{171}	10^{360}	300-400

Enumerate all states

We store all (non-equivalent) states in a set S, i.e., our approach is based on the state space complexity

Initialize S with the starting state

$\forall$ non-processed states $s \in S$ DO
/* process newly added states */
  $\forall$ successors t of s DO
    compute canonical state t' of t
    IF $t' \notin S$ THEN add t' to S

/* S contains all states which are reachable from the start state via valid moves */

'Code' of a State

For every game state we compute a **code** (integer number ≥ 0) which contains all information when playing starts/continues from this state.

- **WIN**: code odd: number of half-moves in which a win can be forced (if player plays perfect).
- **LOSE**: code even: number of half-moves in which the game is at most lost (if opponent plays perfect).
- **DRAW**: special code, e.g. -1; no number of half-moves possible.

How to compute 'Codes'

Init all states without valid moves (with code 0, draw, ...)
/* terminal states without successors */

IF successor state with even code exists THEN
    code := (smallest even code of a succesor state) + 1
    /* WIN in that number of moves */
ELSE IF successor state with draw code exists THEN
    code := draw
    /* DRAW */
ELSE
    code := (largest (odd) code of a successor state) + 1
    /* LOSE in that number of moves */

Pseudocode to compute 'Codes'

Init all states without valid moves
/* terminal states without successors */
Init all remaining states with 'undefined'

max-depth ... until no new codes can be determined

FOR $k := 1$ TO max-depth /* $k = \#$ of half-moves */
  $\forall$ states $s \in S$ with still undefined code DO
    IF k is odd THEN
      IF s has a successor with code $k-1$ THEN
        code of s is k/* WIN state */
    ELSE /* k is even */
      IF **all** successors of s have odd codes THEN
        code of s is k/* LOSE state */
Set all 'undefined' states to draw.

How to use the 'Code'

How to play for a current state s:

- Compute all possible successors of s and their codes.
- IF a successor with even code exist, make the move which leads to the successor with the smallest even code k. Message: "I will win in k half-moves.".
- ELSE IF a successor with code draw exists, make the draw move. Message: "You might make a draw.".
- ELSE make the move to the successor with the highest (odd) code k. Message: "You might win in k half-moves.".

How to use the 'Code'

How to play for a current state s:

- Compute all possible successors of s and their codes.
- IF a successor with even code exist, make the move which leads to the successor with the smallest even

Alternatively in a win situation any win-move can be chosen and comments can be made on the opponents level of play ('You just made a rather sub-optimal move and can not win anymore ...")

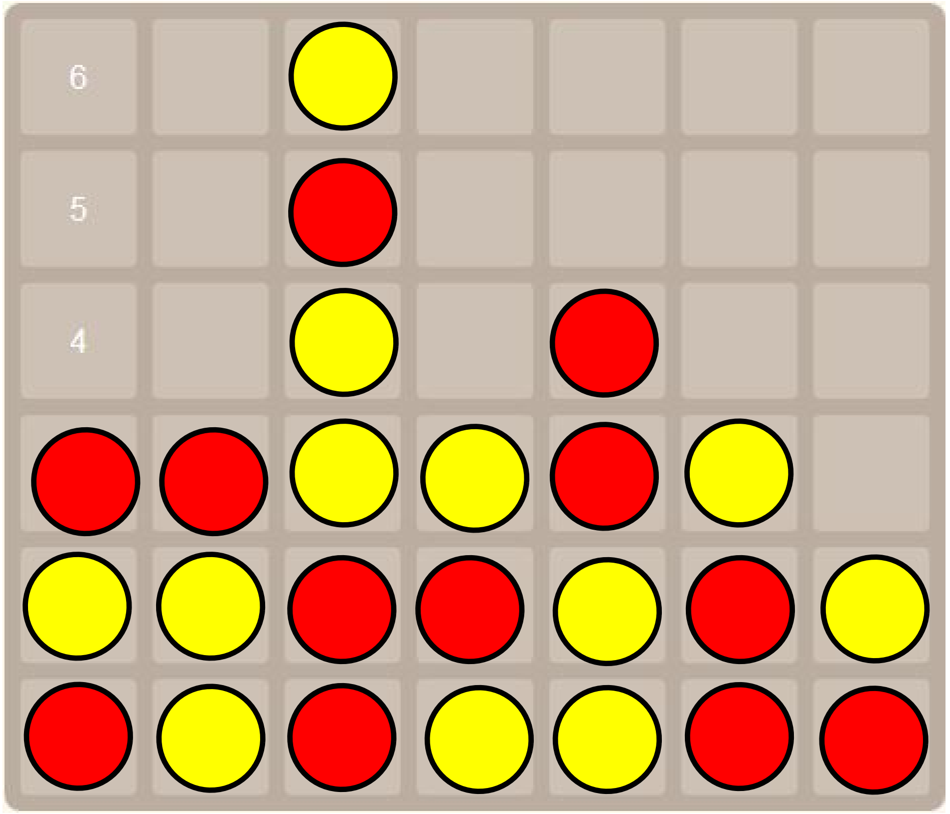
- ELSE make the move to the successor with the highest (odd) code k. Message: "You might win in k half-moves.".

Saving a state of Connect-4

With how many (well, few) byte can you store a game state of Connect-4? You are allowed to use the information on how many half-moves (0 to 42) have already been made.

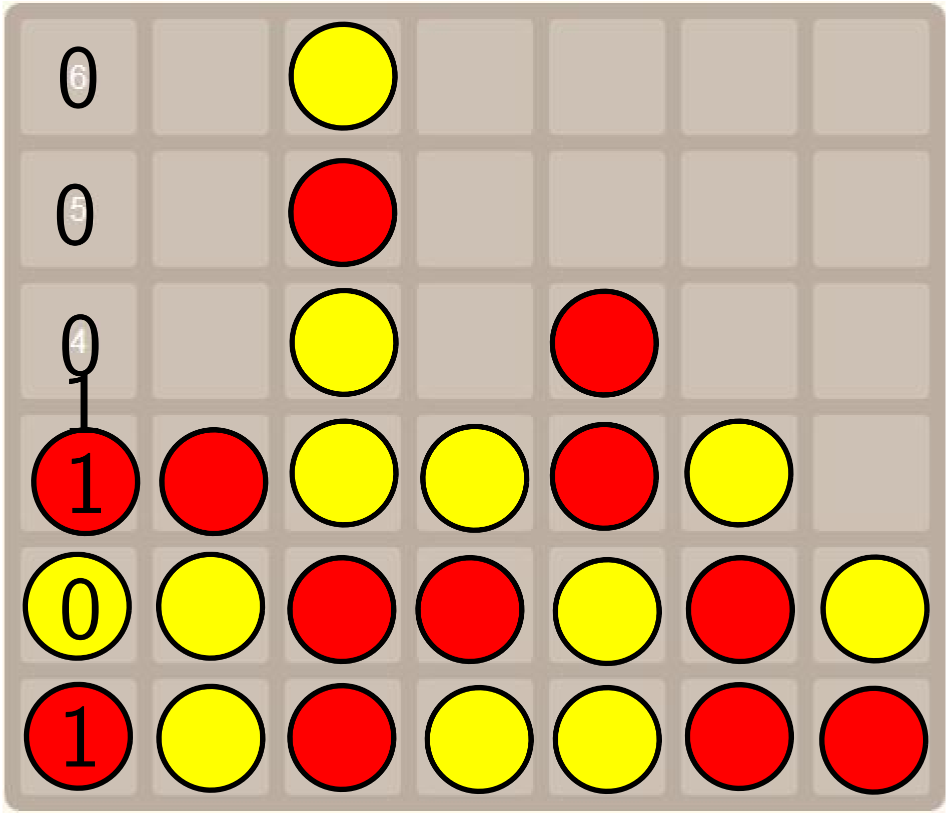
Saving a state of Connect-4

For each column from above: write 0 for each empty field, then a 1 befor the first non-empty field. Starting from there write 0 for a yellow token, and 1 for a red token.



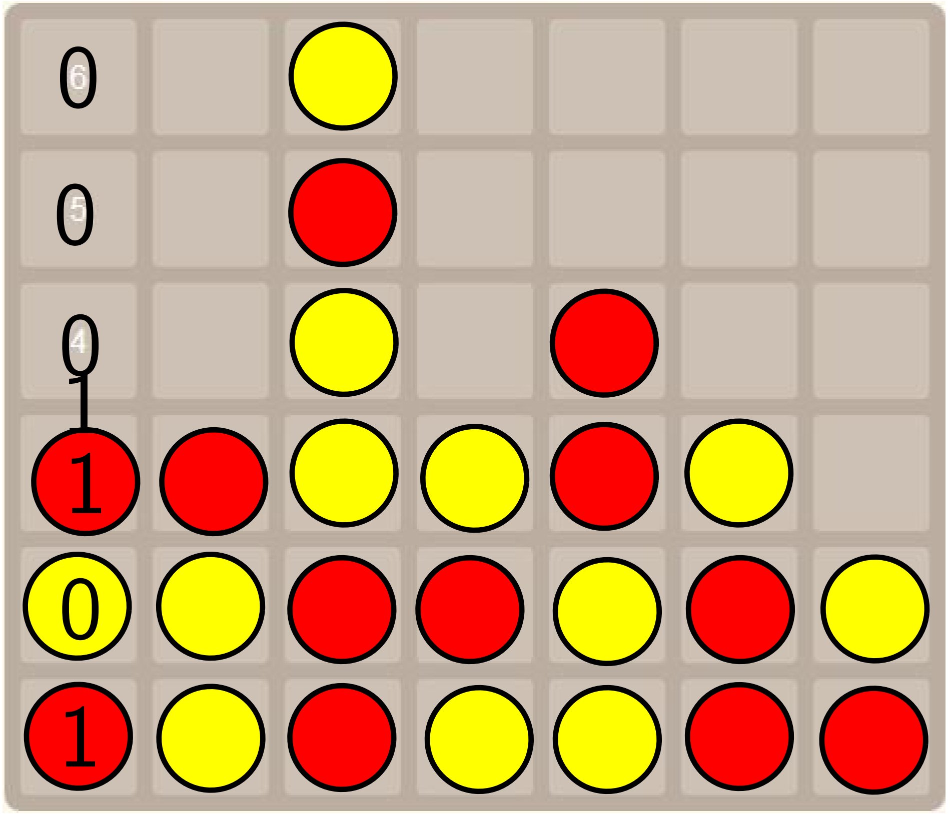
Saving a state of Connect-4

For each column from above: write 0 for each empty field, then a 1 befor the first non-empty field. Starting from there write 0 for a yellow token, and 1 for a red token.



Saving a state of Connect-4

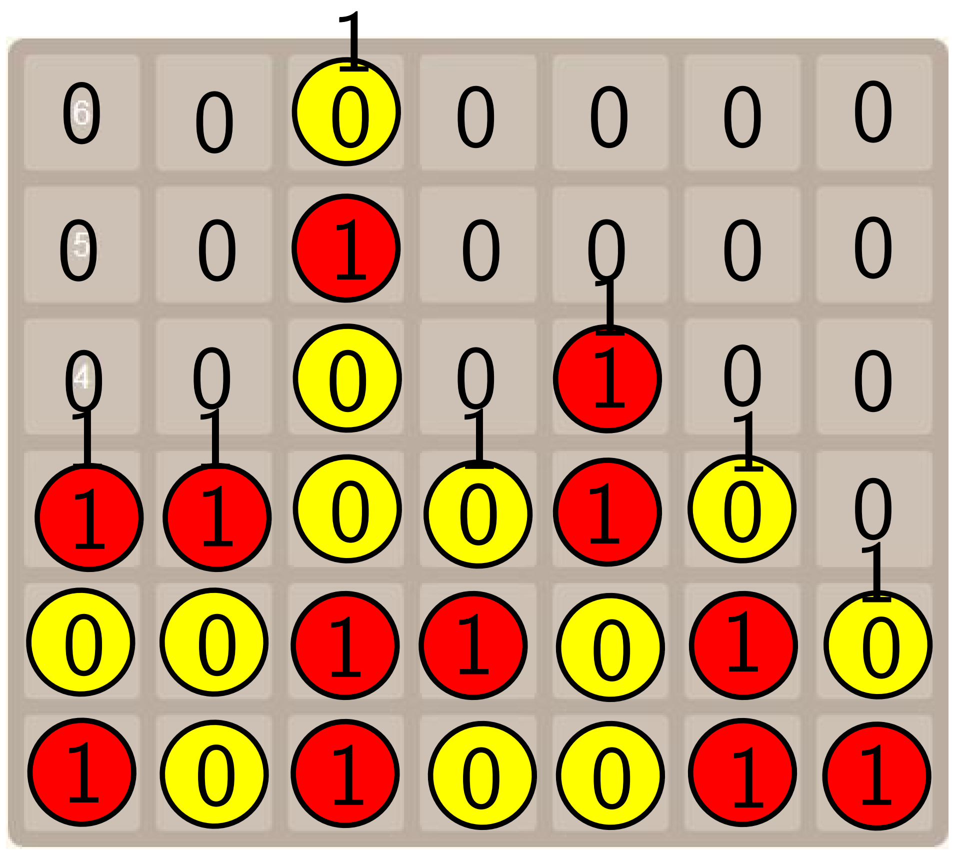
For each column from above: write 0 for each empty field, then a 1 befor the first non-empty field. Starting from there write 0 for a yellow token, and 1 for a red token.



7 bit per column

Saving a state of Connect-4

For each column from above: write 0 for each empty field, then a 1 befor the first non-empty field. Starting from there write 0 for a yellow token, and 1 for a red token.

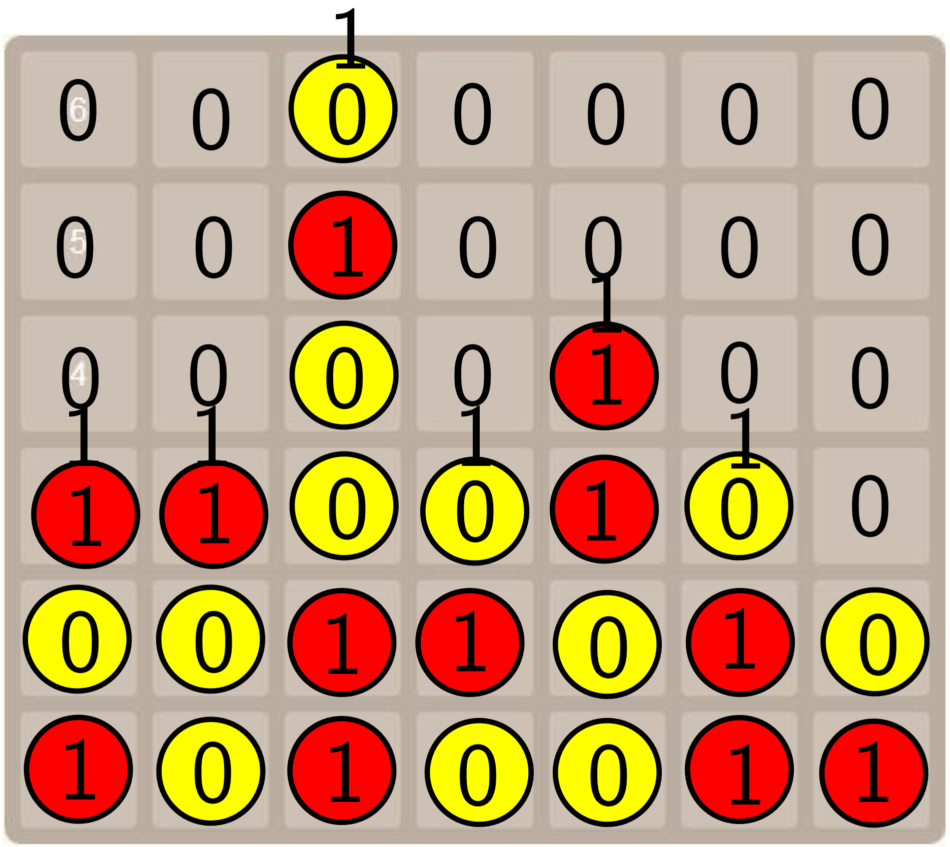


7 bit per column

$7 \times 7 = 49$ bit in total > 6 byte

Saving a state of Connect-4

For each column from above: write 0 for each empty field, then a 1 befor the first non-empty field. Starting from there write 0 for a yellow token, and 1 for a red token.



7 bit per column

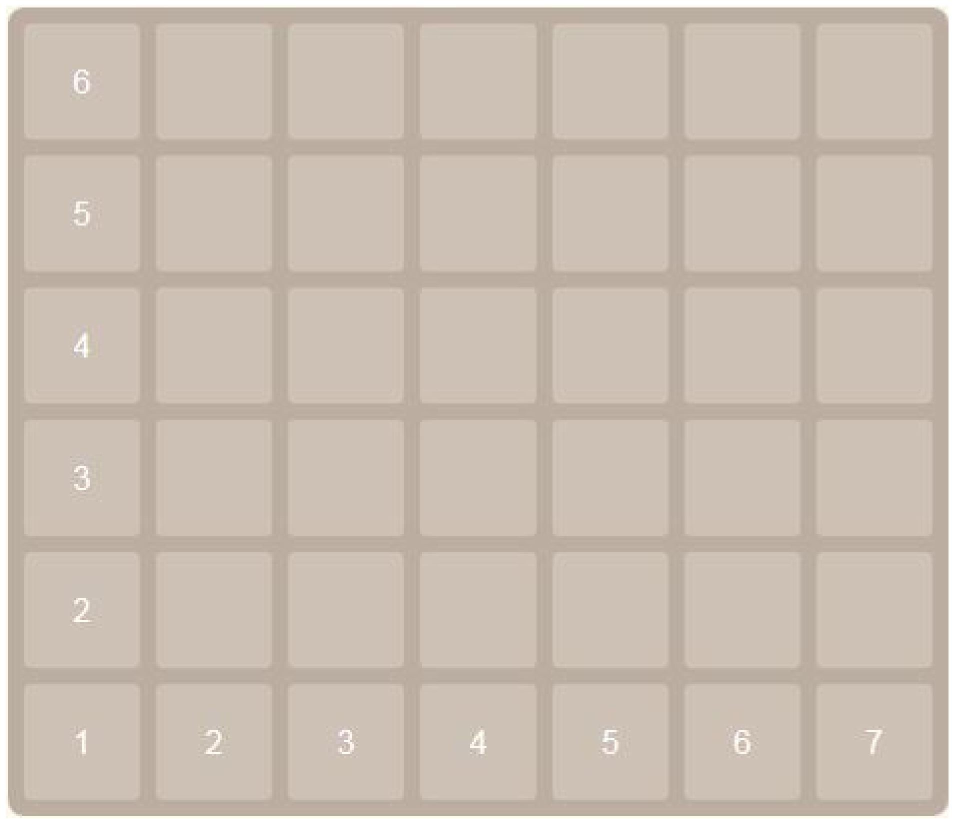
$7 \times 7 = 49$ bit in total > 6 byte

Number of tokens: save 'stop' bit in last column: **6 byte**

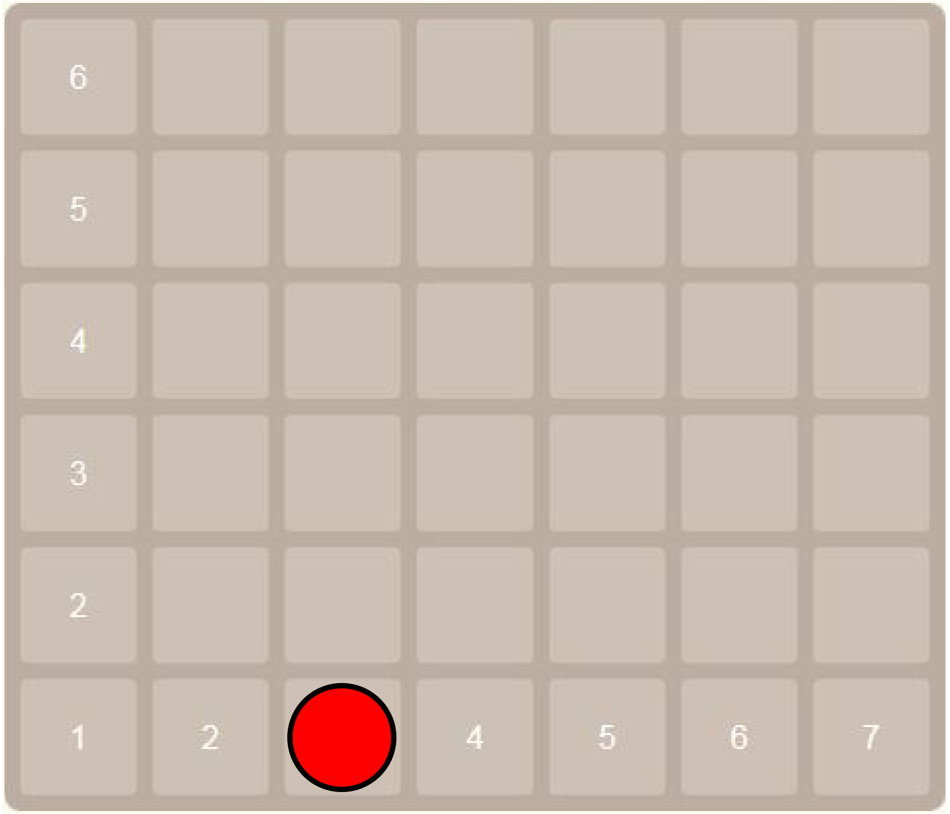
Computer Playing: Connect-4

- All game states need to be stored memory efficient and complete: DONE!
- Move generator (successors of a game state): Just add a token in a non-full column. At most 7 successors exist.
- Identify final states: For lose (i.e., previous player win) just check up to 11 4-tuples (including new token).
  No win and 42 tokens placed: terminal draw state.
- For efficiency: backwards move generator (predecessors of a game state): **???**

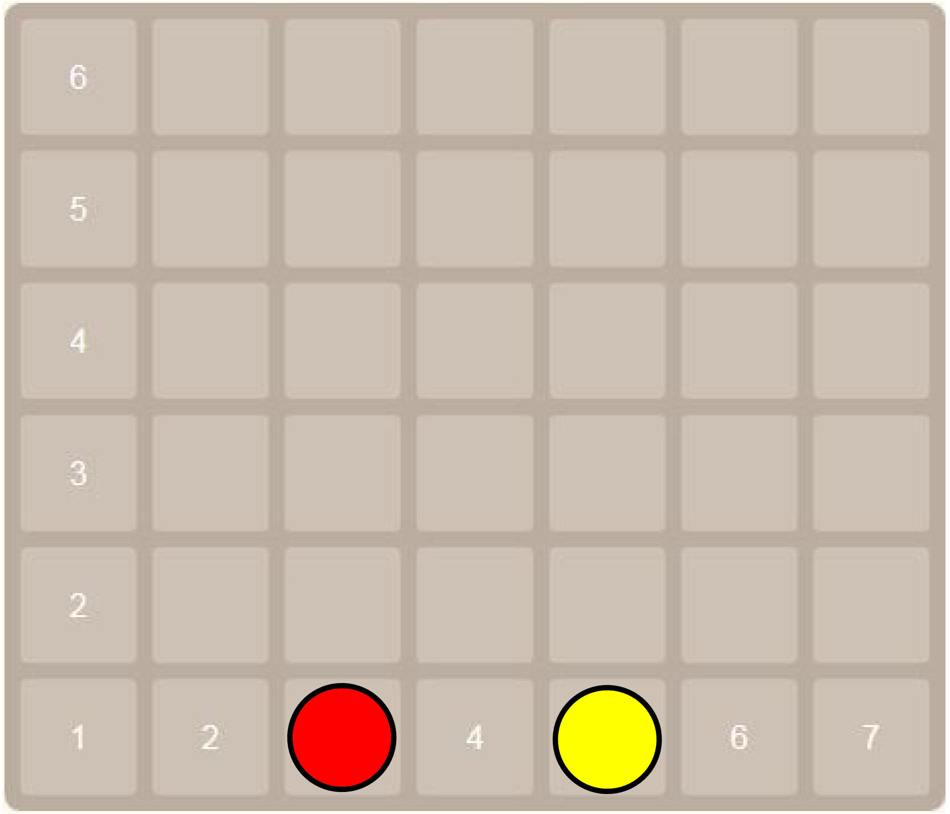
Connect-4: no backward moves



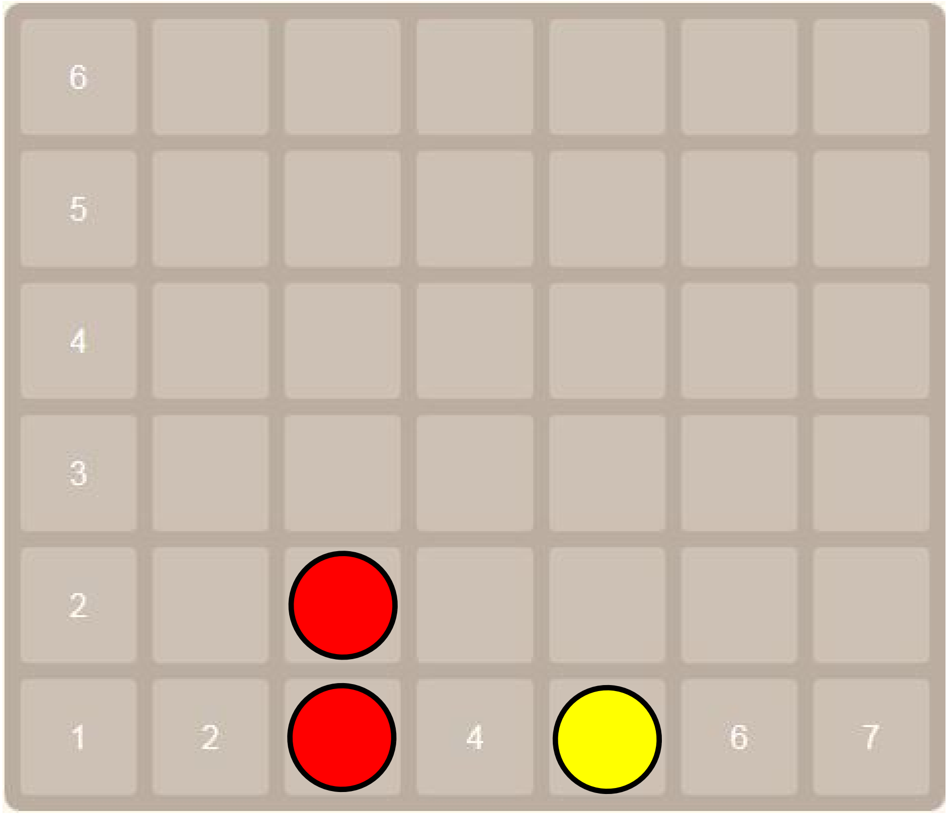
Connect-4: no backward moves



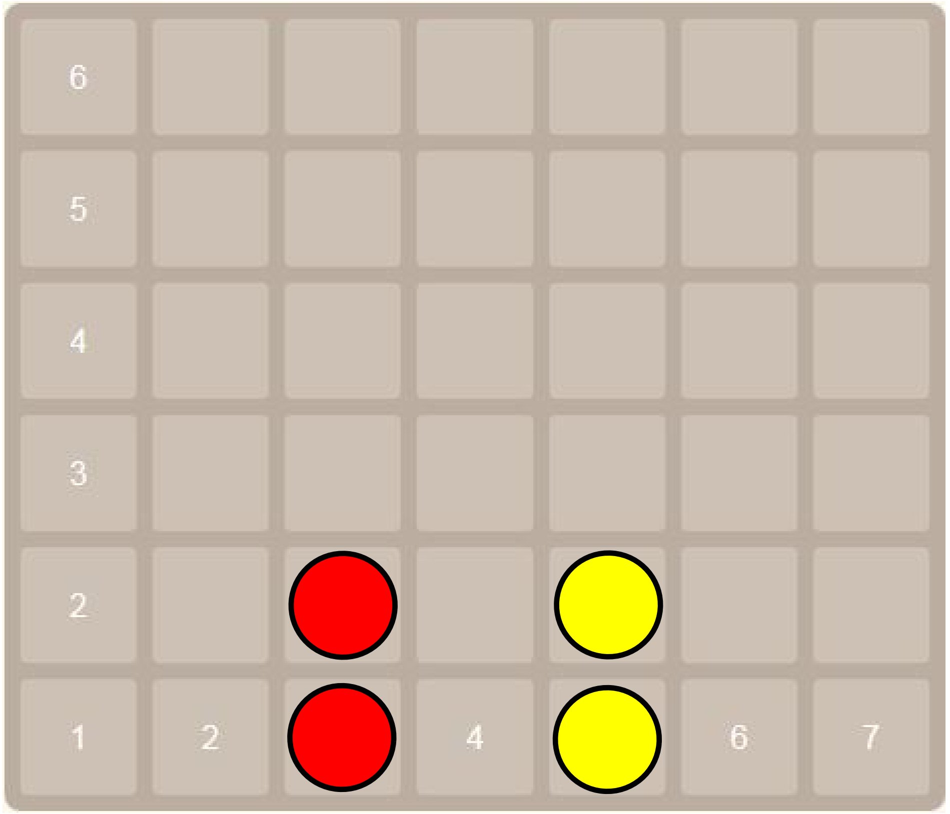
Connect-4: no backward moves



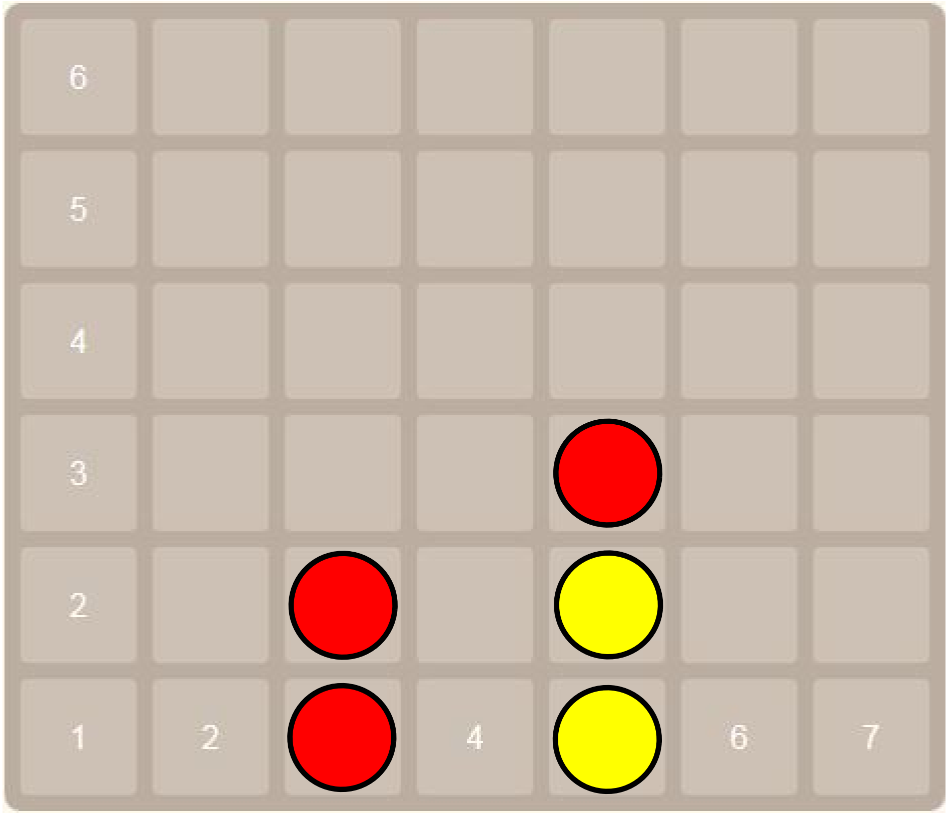
Connect-4: no backward moves



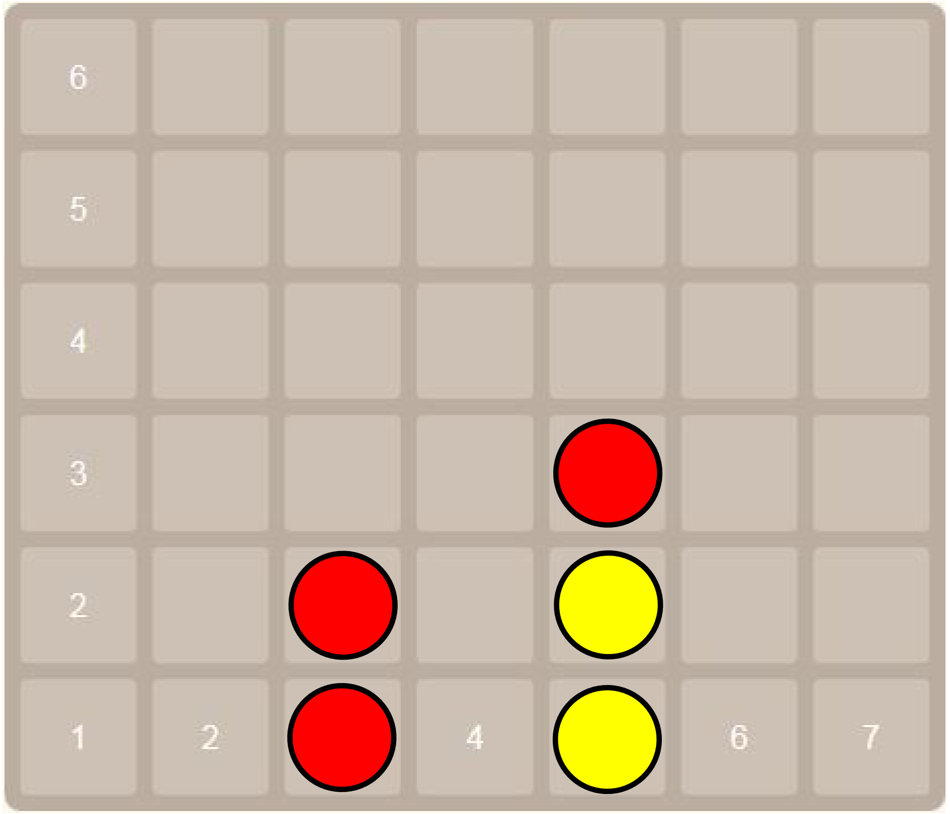
Connect-4: no backward moves



Connect-4: no backward moves

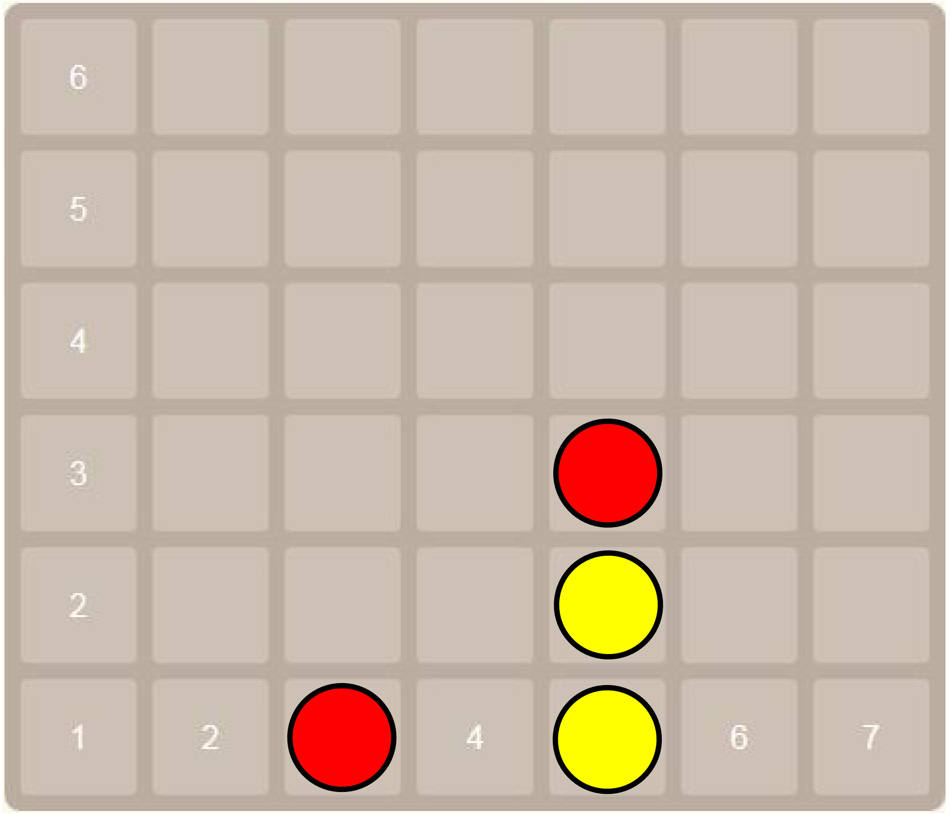


Connect-4: no backward moves



Valid position with **two** predecessors?

Connect-4: no backward moves



Valid position with **two** predecessors?

Not a valid position! For general sized boards it is NP-complete to decide if a position is valid.

Connect-4: Number of States

half-moves	different boards
0	1
1	4
2	25
3	121
4	568
5	2144
6	8231
7	27109

half-moves	different boards
8	91295
9	269531
10	809464
11	2148087
12	5832236
13	14105207
14	35045629
15	77785047

half-moves	different boards
16	177841160
17	363798195
18	767435580
19	1448894267
20	2818993420
21	4907390200
22	8788132016
23	14066554884
sum	33475164421

Connect-4: Number of States

half-moves	different boards	half-moves	different boards	half-moves	different boards
0	1	8	91295	16	177841160
1	4	9	269531	17	363798195
2	25	10	809464	18	767435580
3	121	11	2148087	19	1448894267
4	568	12	5832236	20	2818993420
5	2144	13	14105207	21	4907390200
6	8231	14	35045629	22	8788132016
7	27109	15	77785047	23	14066554884
				sum	33475164421

33475164421 states with 6 byte each: 200 GB

Connect-4: Number of States

half-moves	different boards
0	1
1	4
2	25
3	121
4	568
5	2144
6	8231
7	27109

half-moves	different boards
8	91295
9	269531
10	809464
11	2148087
12	5832236
13	14105207
14	35045629
15	77785047

half-moves	different boards
16	177841160
17	363798195
18	767435580
19	1448894267
20	2818993420
21	4907390200
22	8788132016
23	14066554884
sum	33475164421

33475164421 states with 6 byte each: 200 GB + 34 GB

Connect-4: Current Data Base

Storing all possible positions?

- 2 bit per square: $2^{2*6*7} = 2^{84} \approx 2 \times 10^{25}$ states: 1 byte each, 17592186044416 TB.
- 3 possibilities per square: $3^{6*7} = 3^{42} \approx 10^{20}$ states: 1 byte each, 99515990 TB.
- 6 byte per state: $2^{6*8} = 281474976710656 \approx 10^{14}$ states: 1 byte each, 281 TB.

Storing all possible states up to 23 half-moves: 234GB.
Maximal remaining search depth: $42 - 23 = 19$, with ≈ 5 possible moves in average.

Summary

- Aim for ultra-strongly solved games.
- Use hybrid approach (space-time tradeoff):
  - Enumerate/code states for a limited number of half-moves
  - Evaluate remaining end-games via the game-tree
- Provide all possible kind of information to fully analyse games: connect-4 is a first player win (play column 4) in 41 half-moves, ...
- Future: Find compact representation of data base via a small set of rules with list of exceptional states.