Course 9

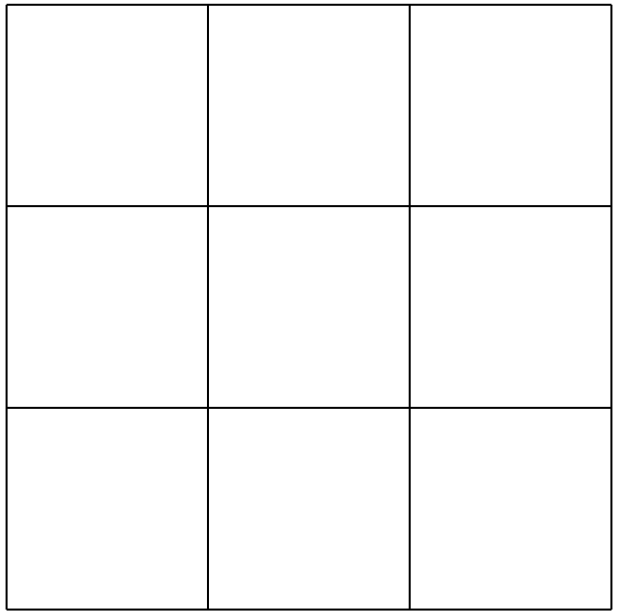
Games

Rules of the game

- Two players: MAX and MIN
- Both have as goal to win the game
- Only one can win or else it will be a draw
- In the initial modeling there is no chance (but it can be simulated)
- Examples:
  - chess
  - checkers
  - tic-tac-toe
  - ...

The tic-tac-toe game

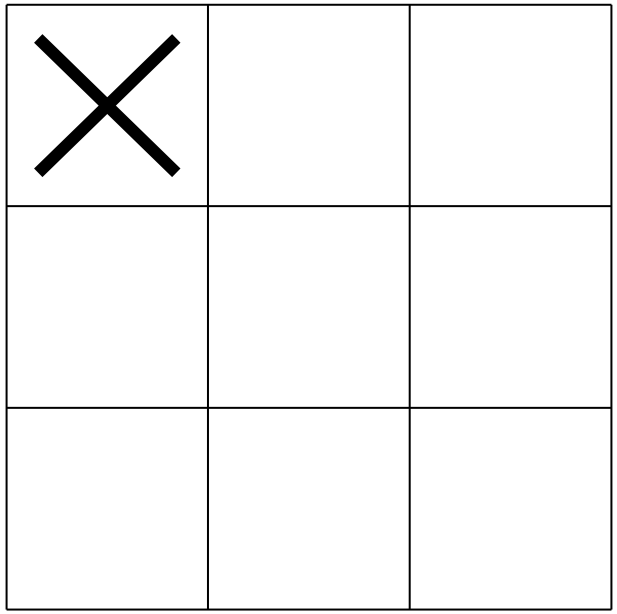
MAX plays with Xs



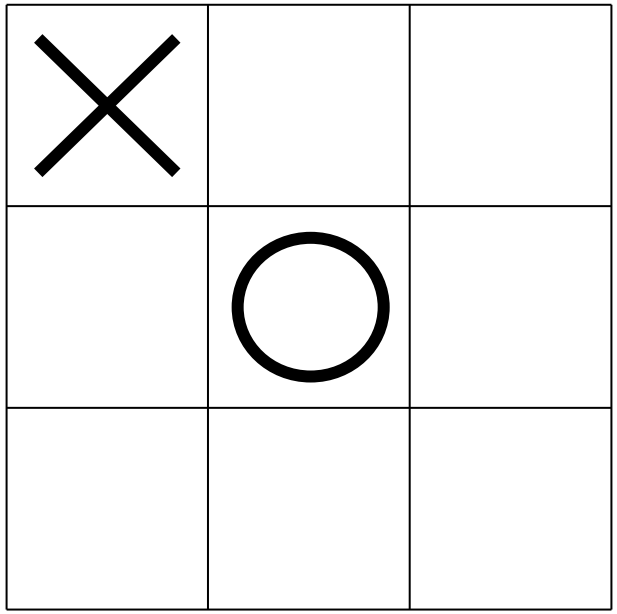
MIN plays with Os

The tic-tac-toe game

MAX



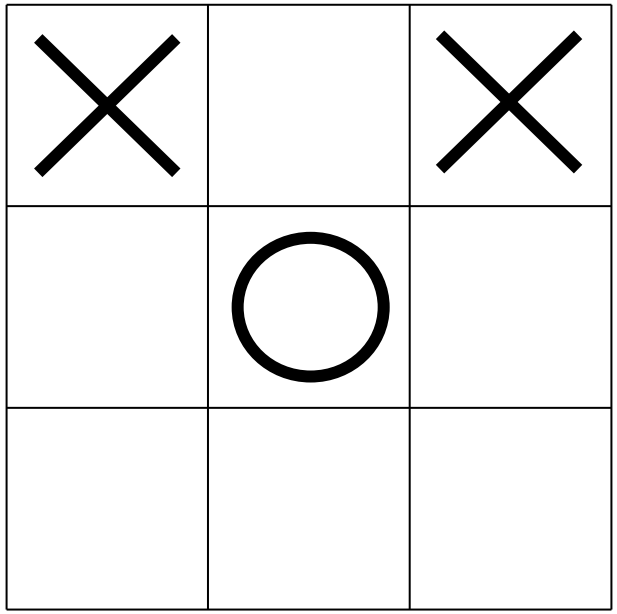
The tic-tac-toe game



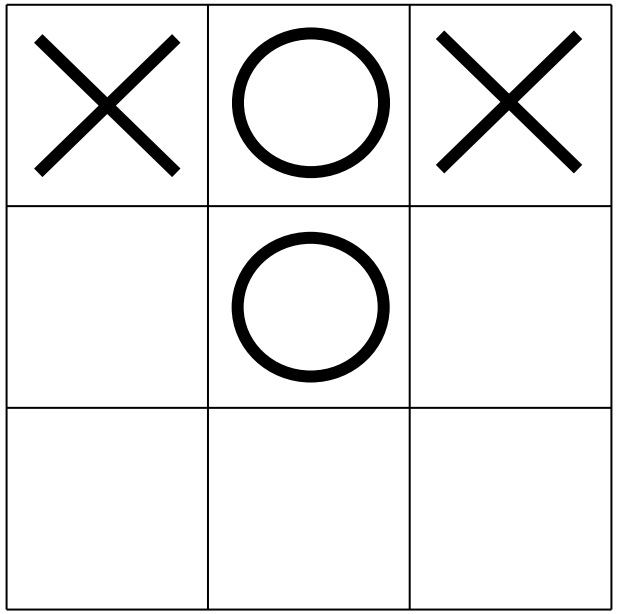
MIN

The tic-tac-toe game

MAX



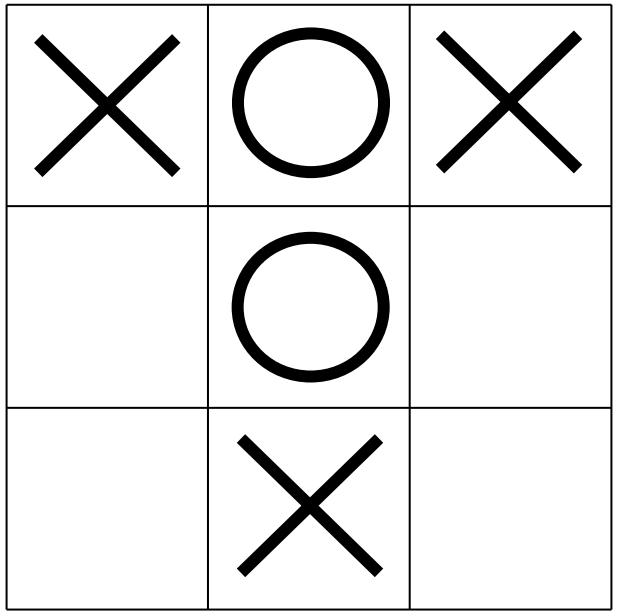
The tic-tac-toe game



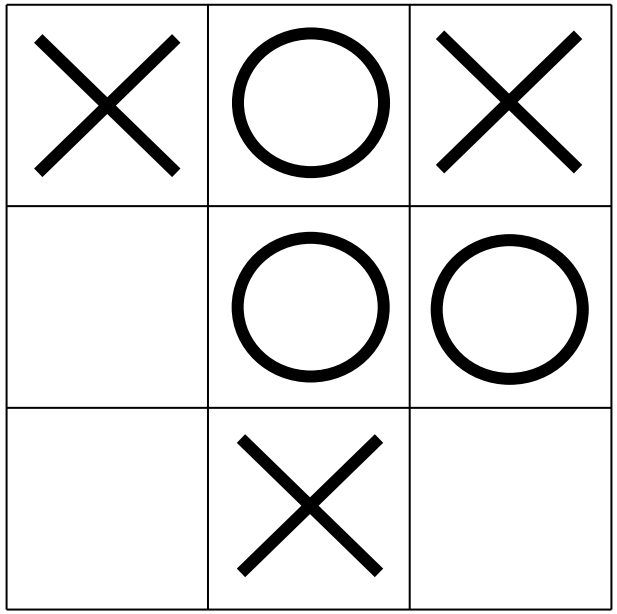
MIN

The tic-tac-toe game

MAX



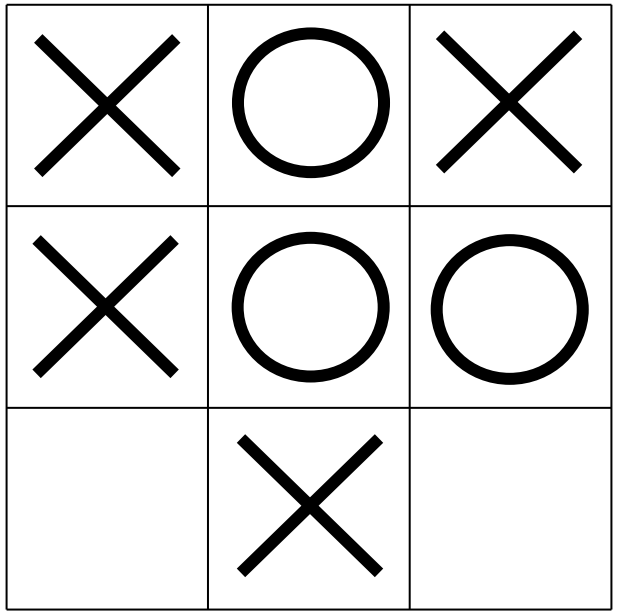
The tic-tac-toe game



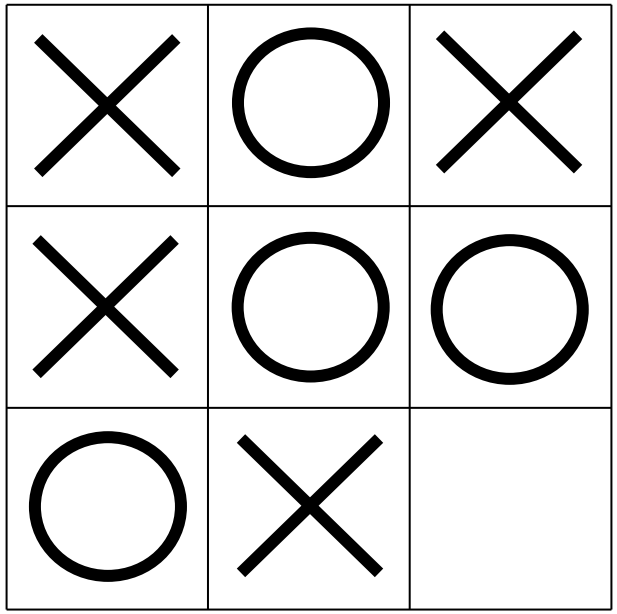
MIN

The tic-tac-toe game

MAX



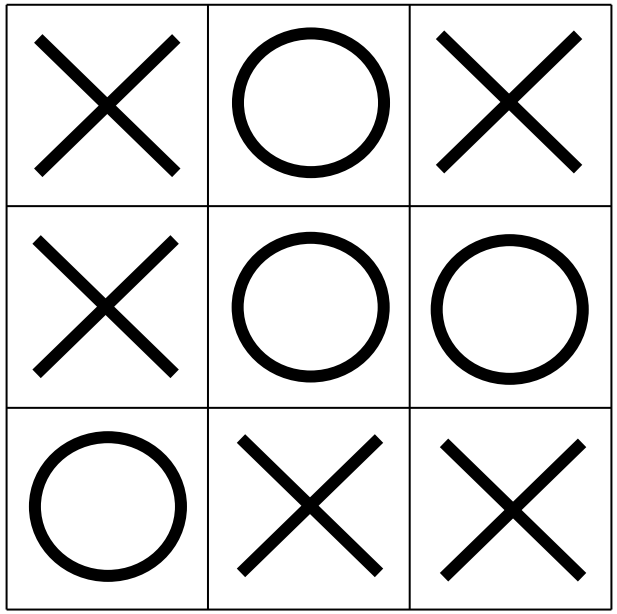
The tic-tac-toe game



MIN

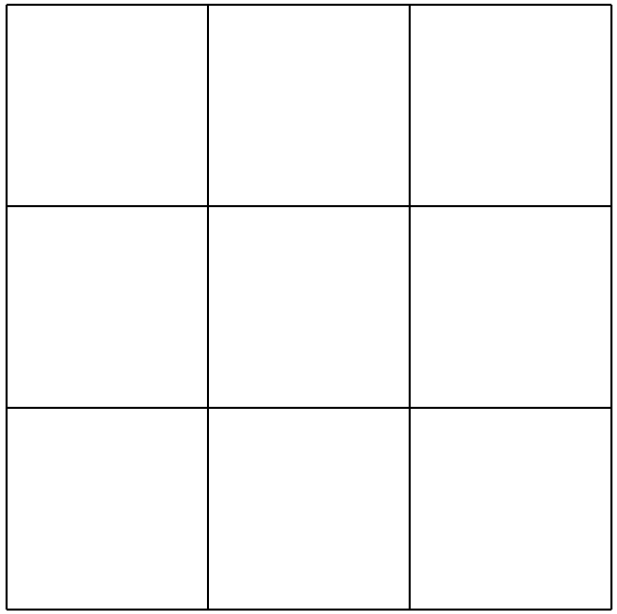
The tic-tac-toe game

MAX



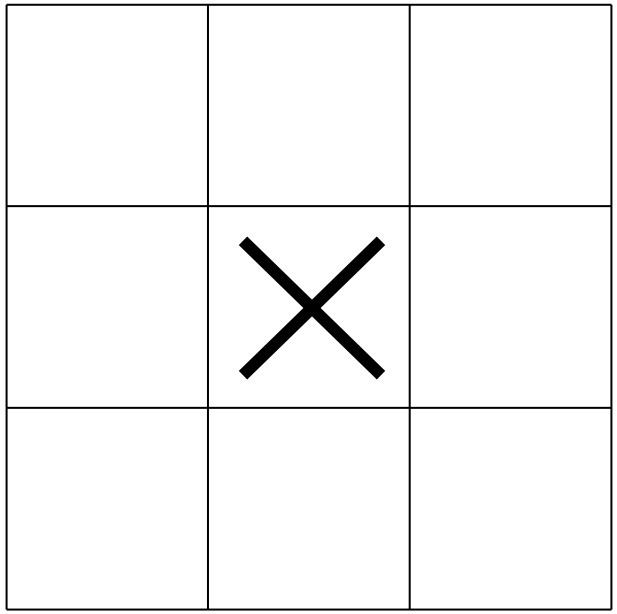
Draw!

The tic-tac-toe game

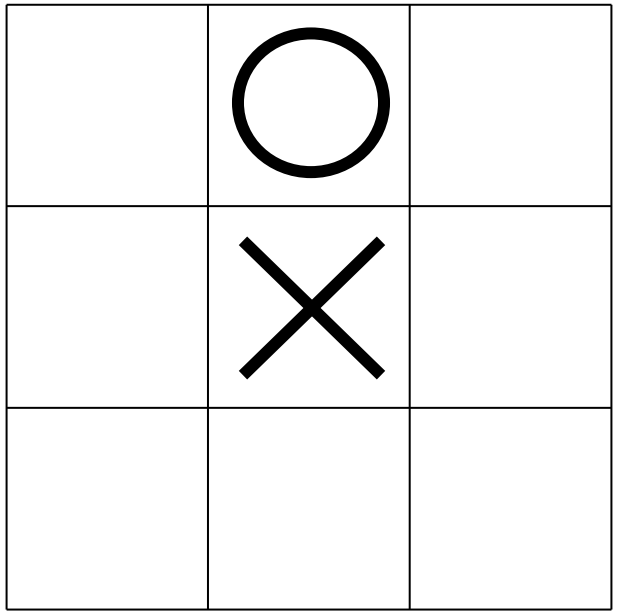


The tic-tac-toe game

MAX



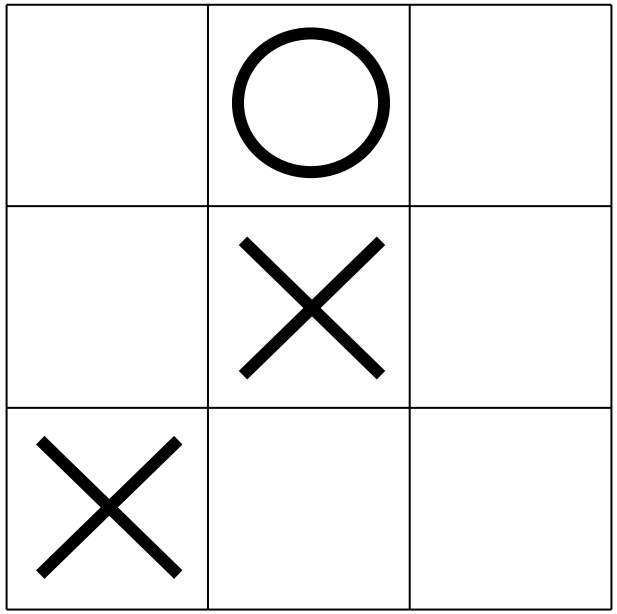
The tic-tac-toe game



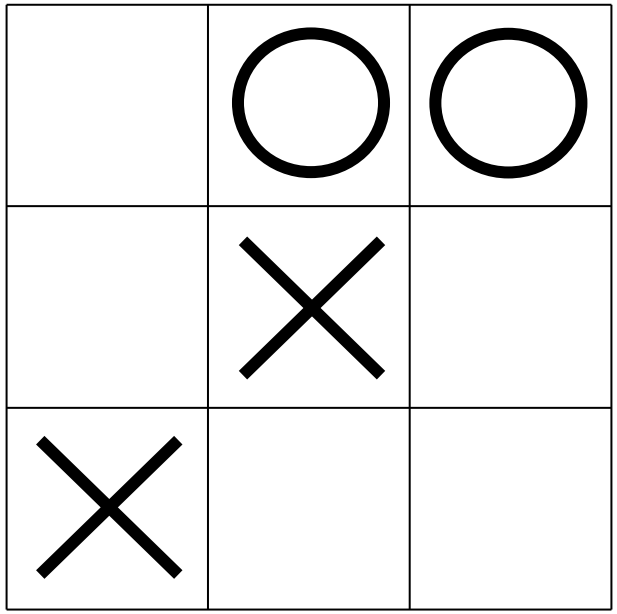
MIN

The tic-tac-toe game

MAX



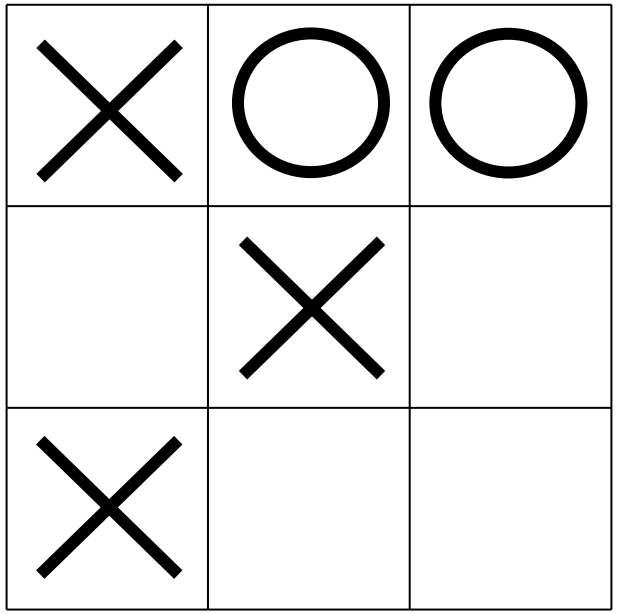
The tic-tac-toe game



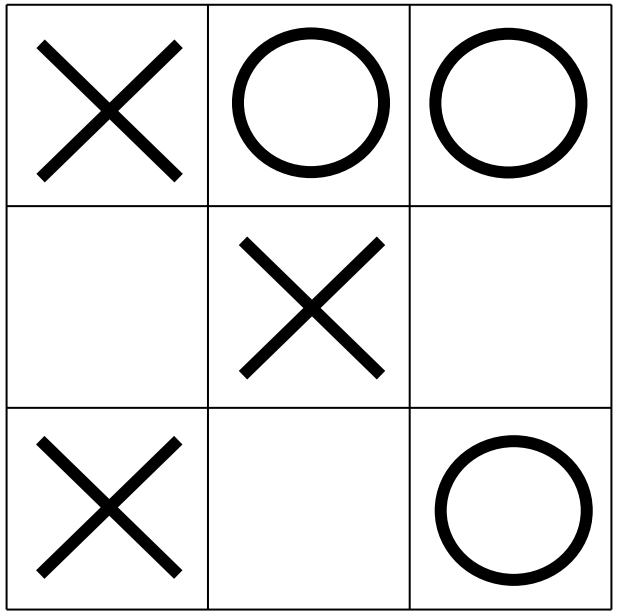
MIN

The tic-tac-toe game

MAX



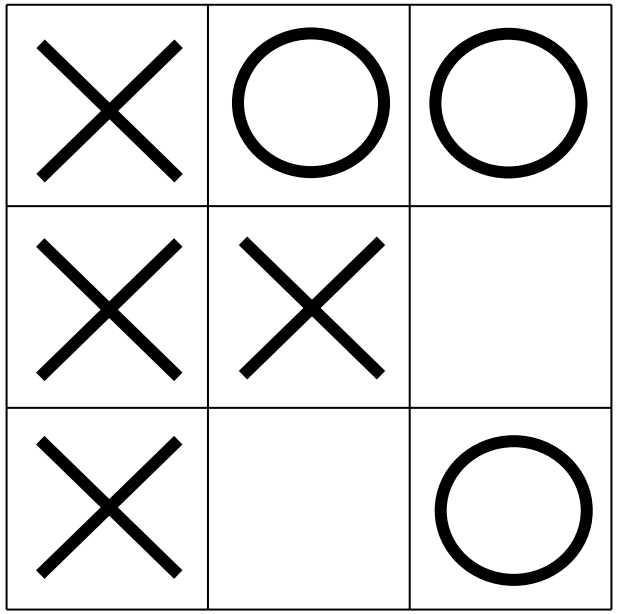
The tic-tac-toe game



MIN

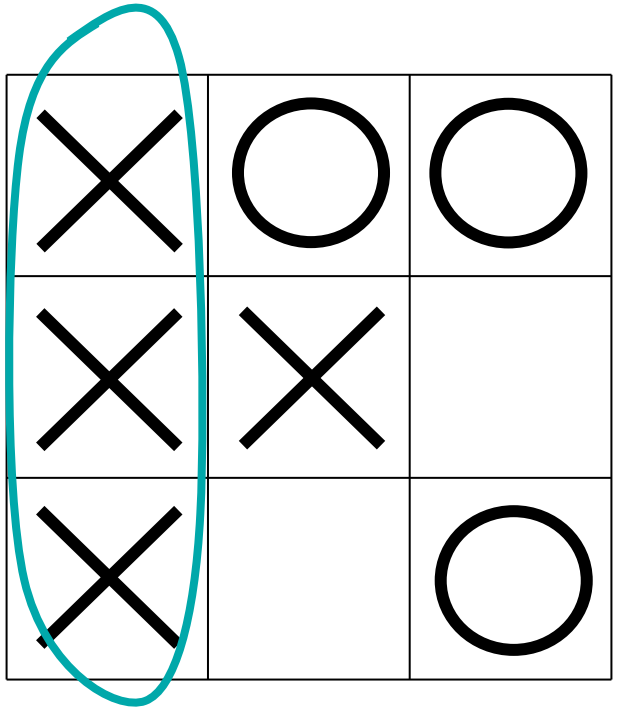
The tic-tac-toe game

MAX



The tic-tac-toe game

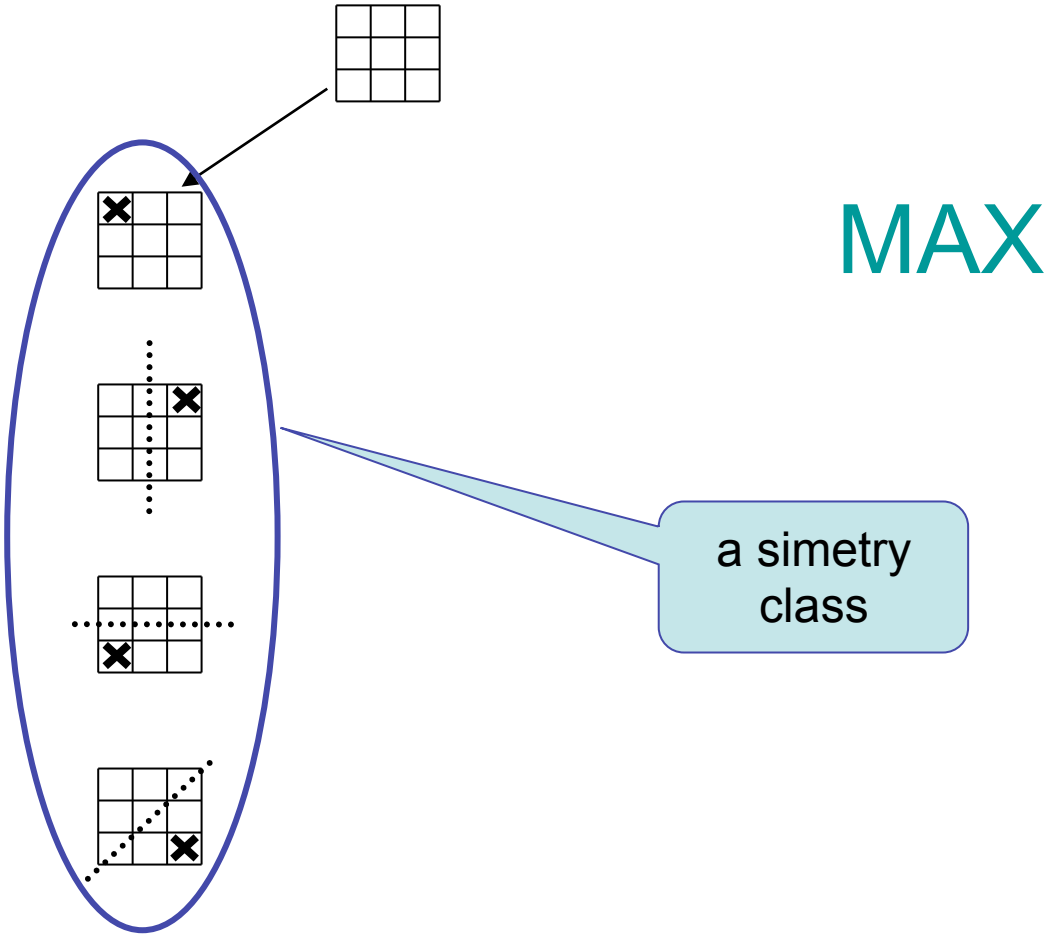
MAX wins



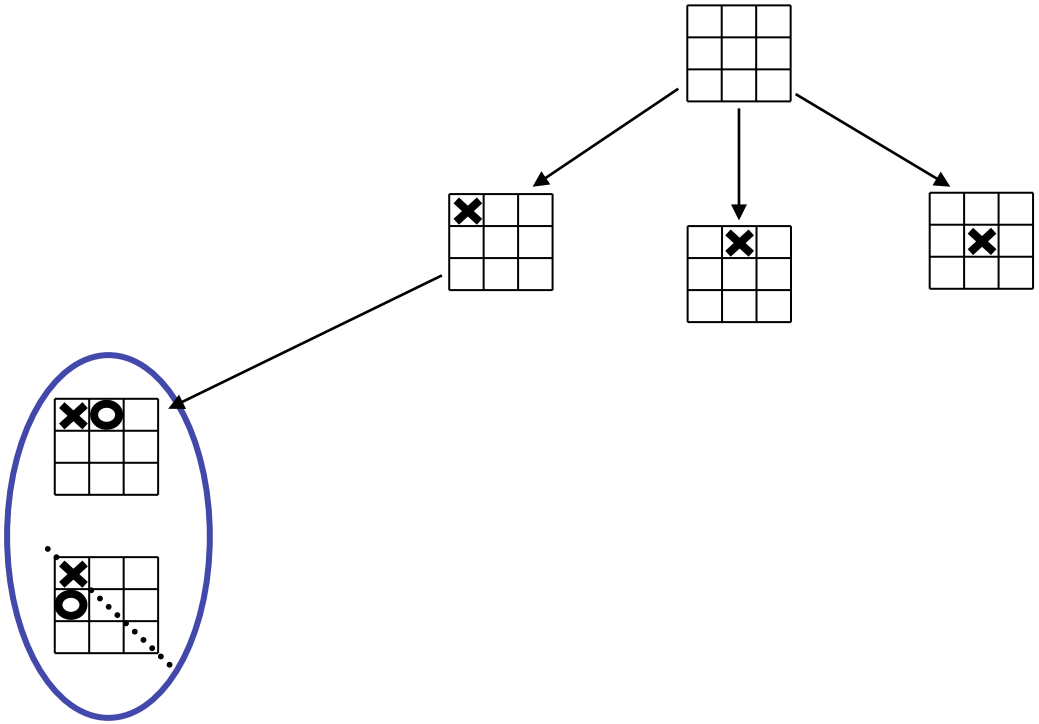
Representation as an AI problem

1. Problem versus instance
2. The state space:
   - a state: the position on the board of the signs between two moves
   - the size of the space: 3^9
3. Representing a state:
   - a 3x3 matrix
4. Representing a transition
   - algorithmical (in the present approach)
5. How is it controled the evolution of the game?
   - the MIN-MAX method
   - the ALPHA-BETA method

The game tree



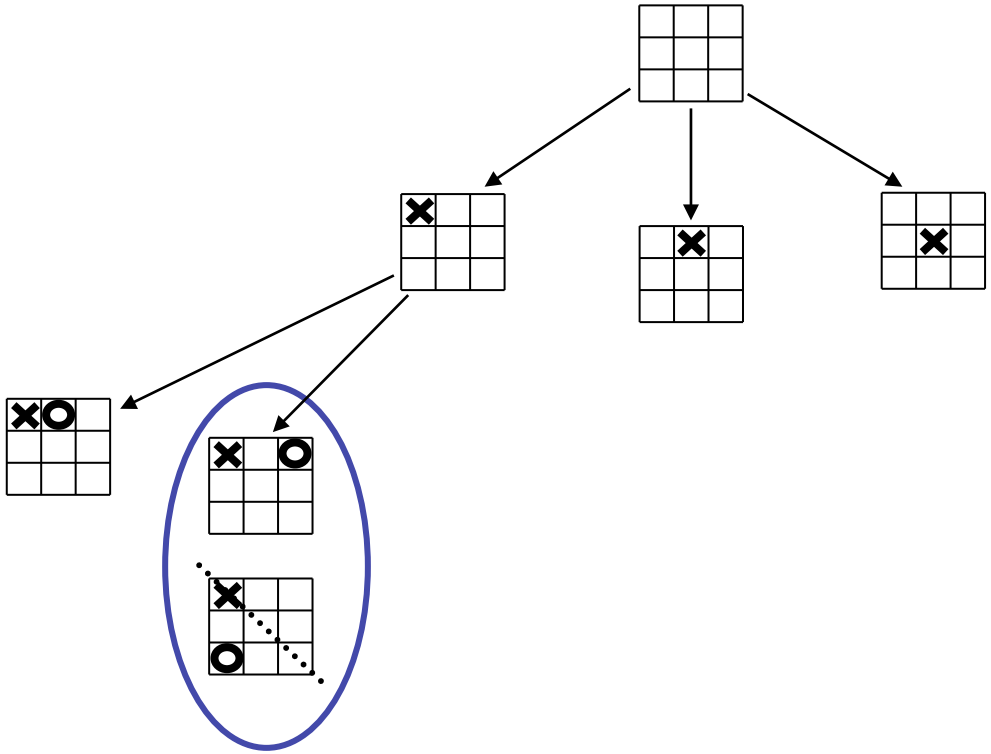
The game tree



MAX

MIN

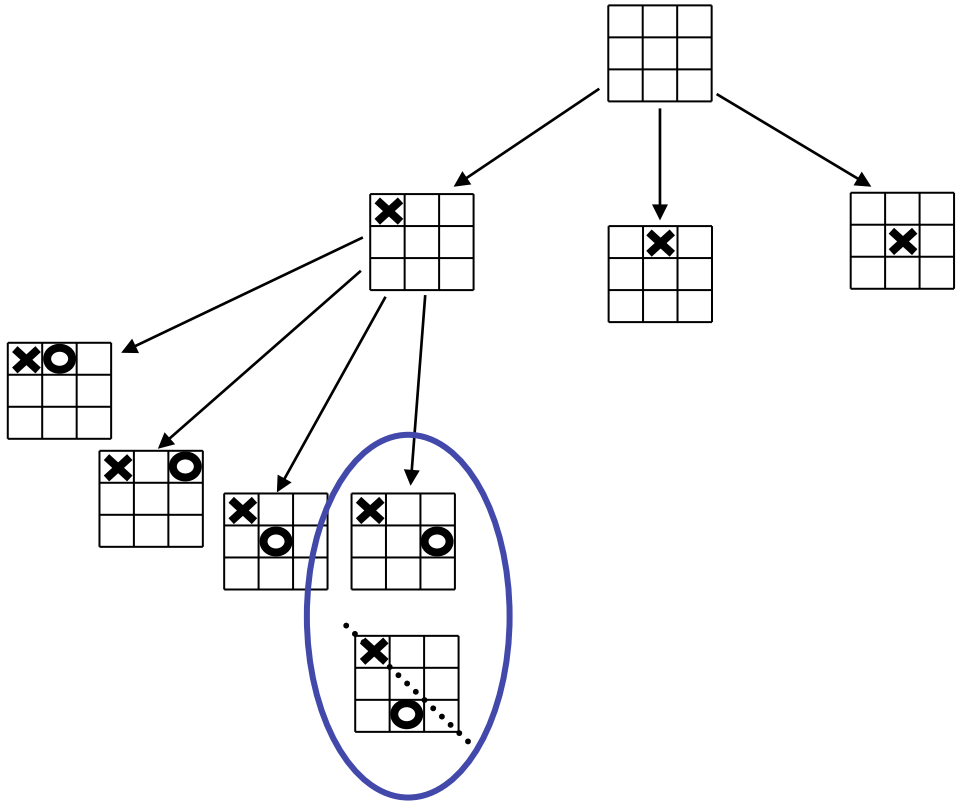
The game tree



MAX

MIN

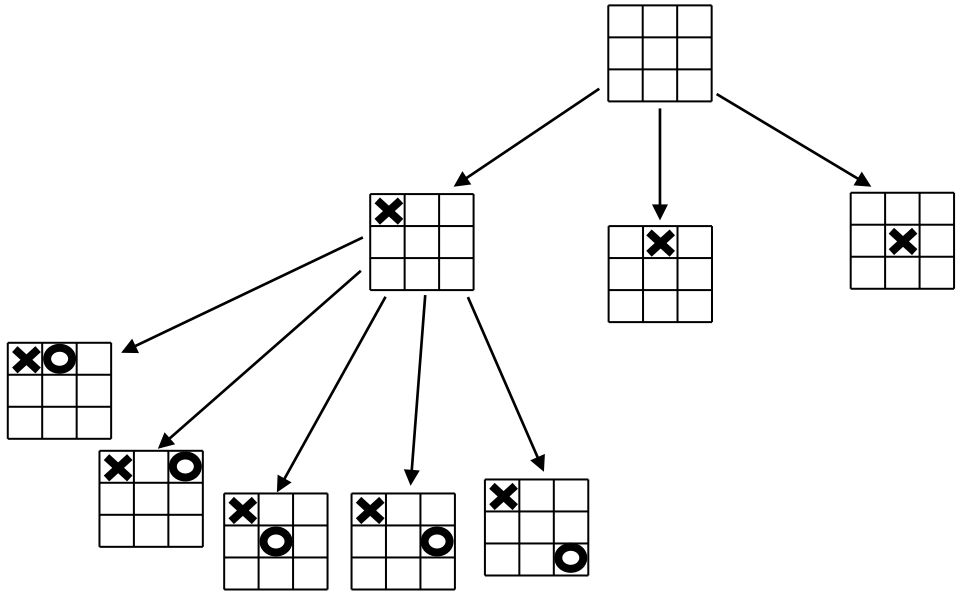
The game tree



MAX

MIN

The game tree

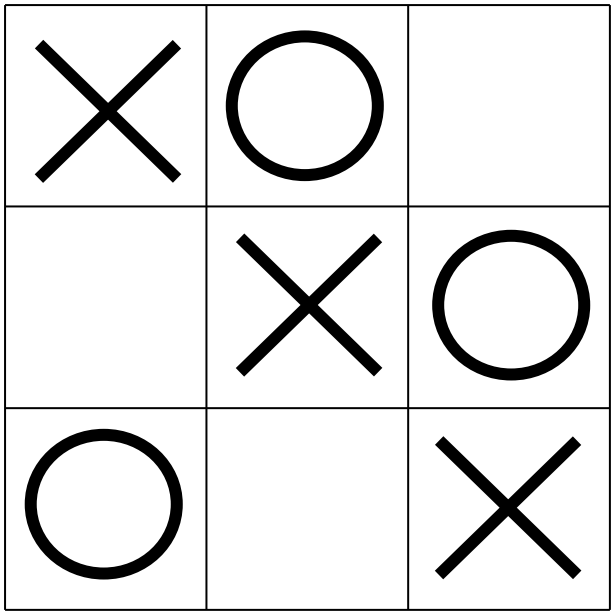


MAX

MIN

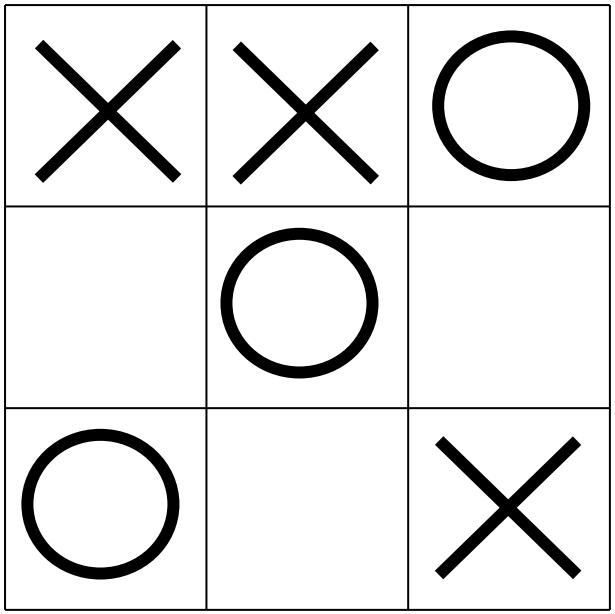
The value of a state

MAX wins: +∞



The value of a state

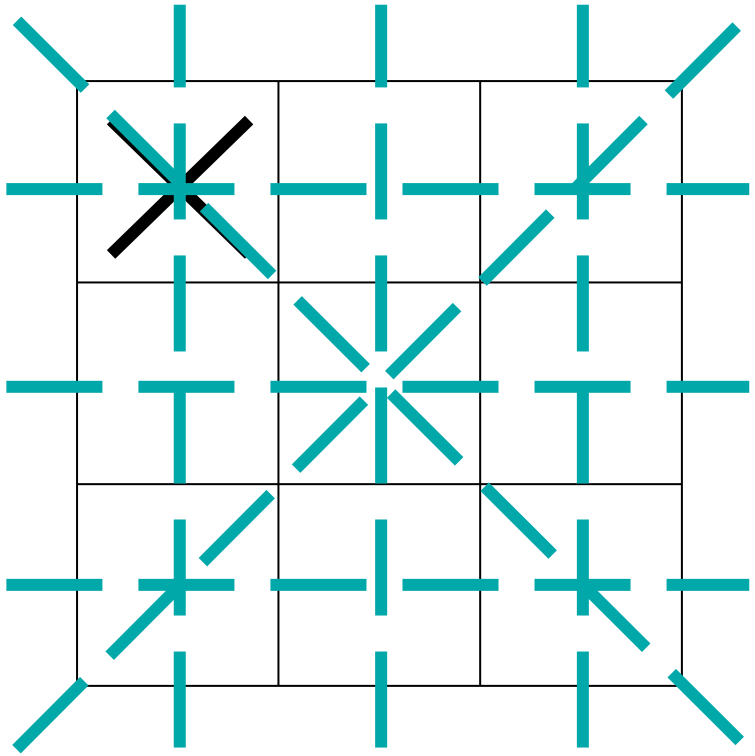
MIN wins: -∞



Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

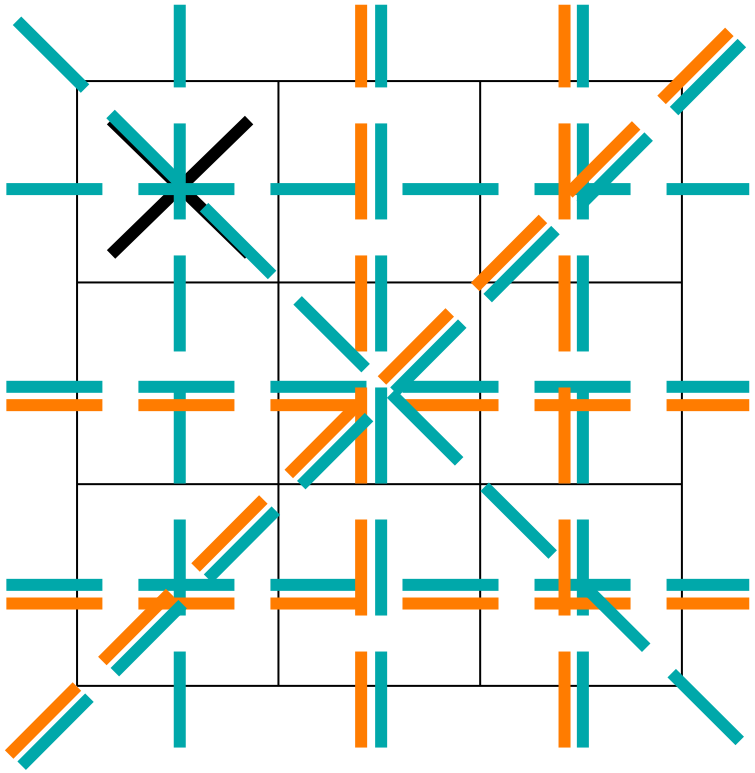
An example of an evaluation function: *the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*



Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:
the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.

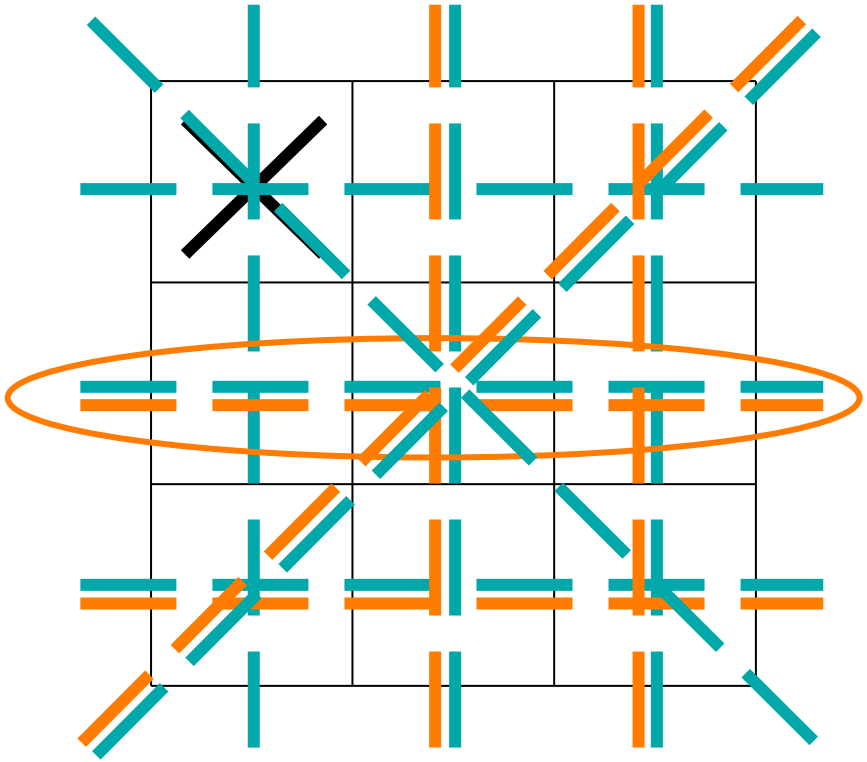


8 - 5 = 3

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function: *the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

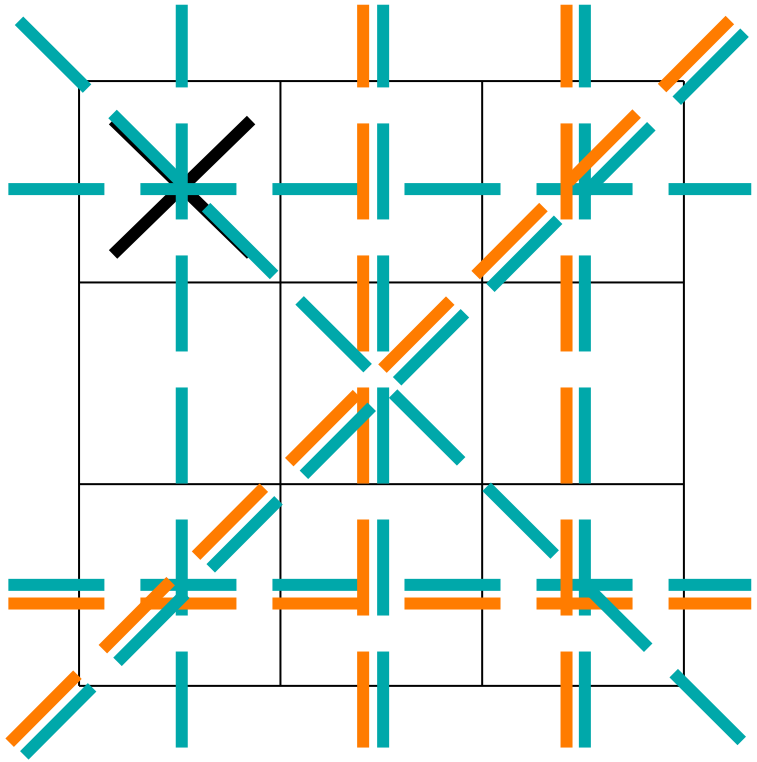


Lines having no sign could be taken by any of the players...

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function: *the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

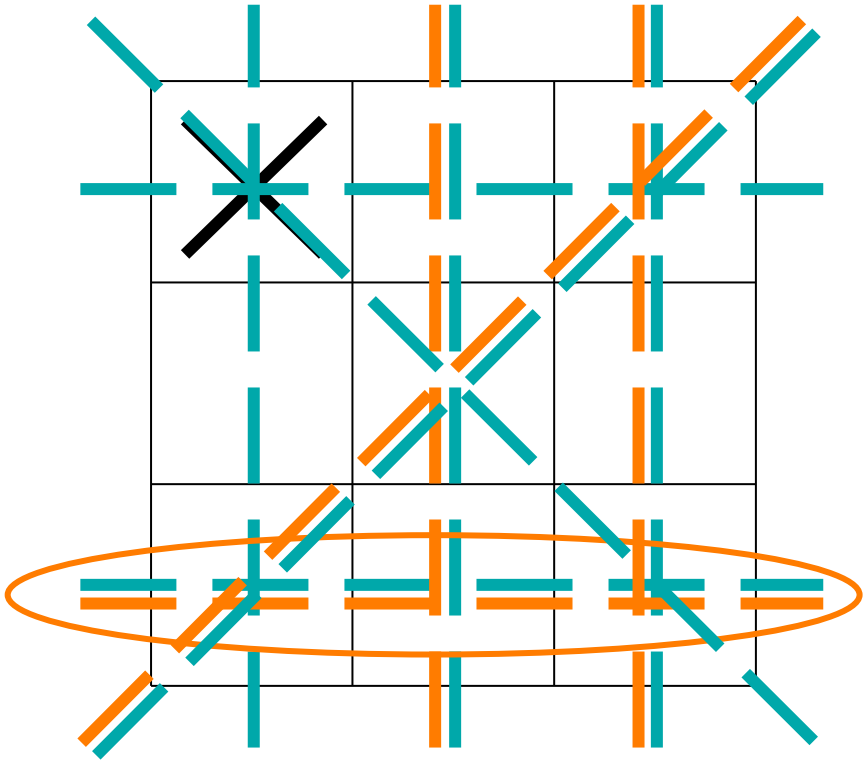


Lines having no sign could be taken by any of the players...

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:
the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.

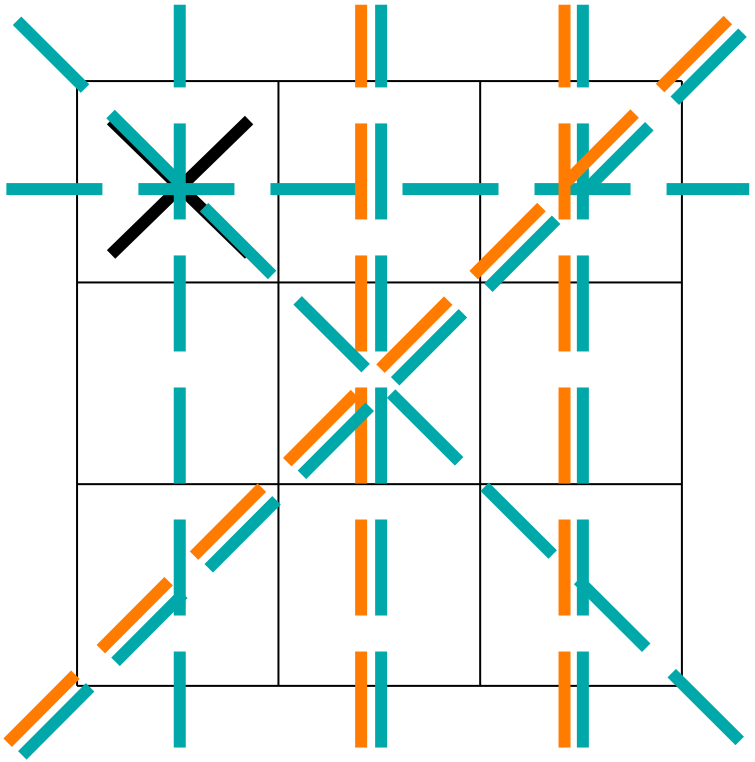


Lines having no sign could be taken by any of the players...

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function: *the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*

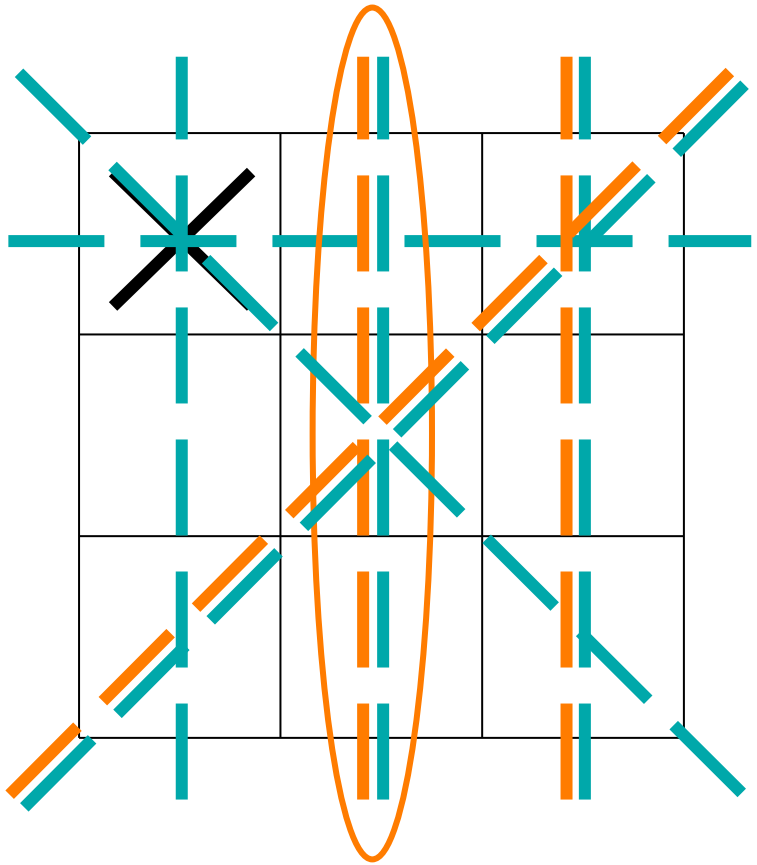


Lines having no sign could be taken by any of the players...

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:
the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.

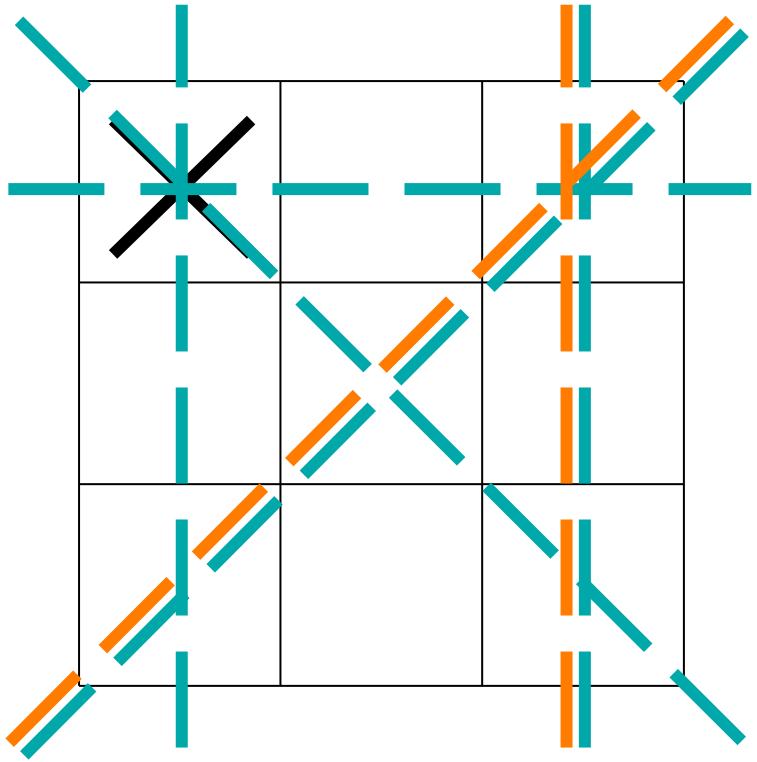


Lines having no sign could be taken by any of the players...

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:
the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.

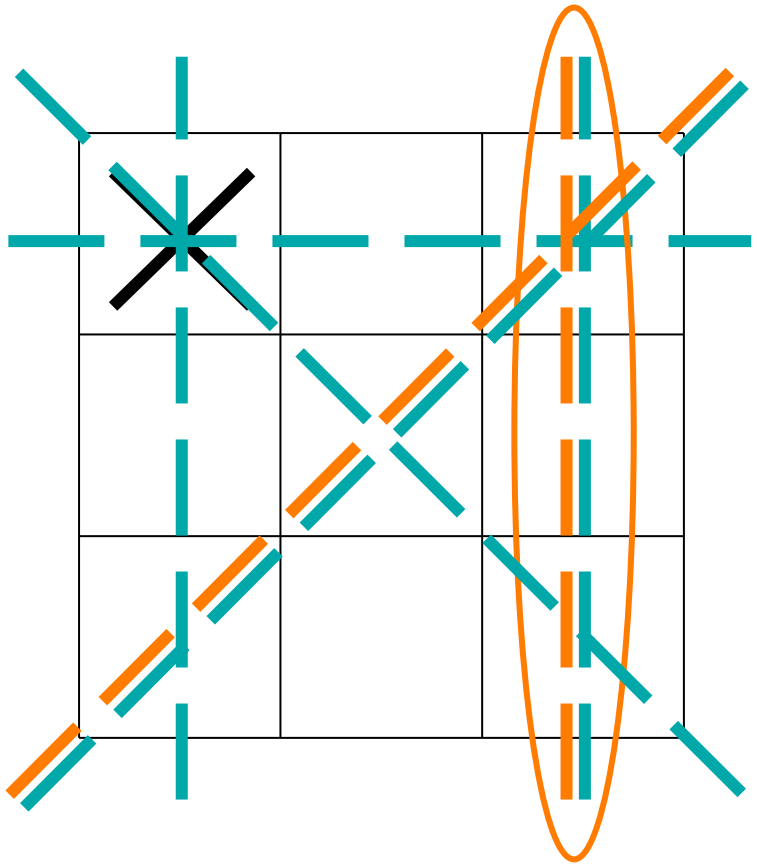


Lines having no sign could be taken by any of the players...

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:
the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.

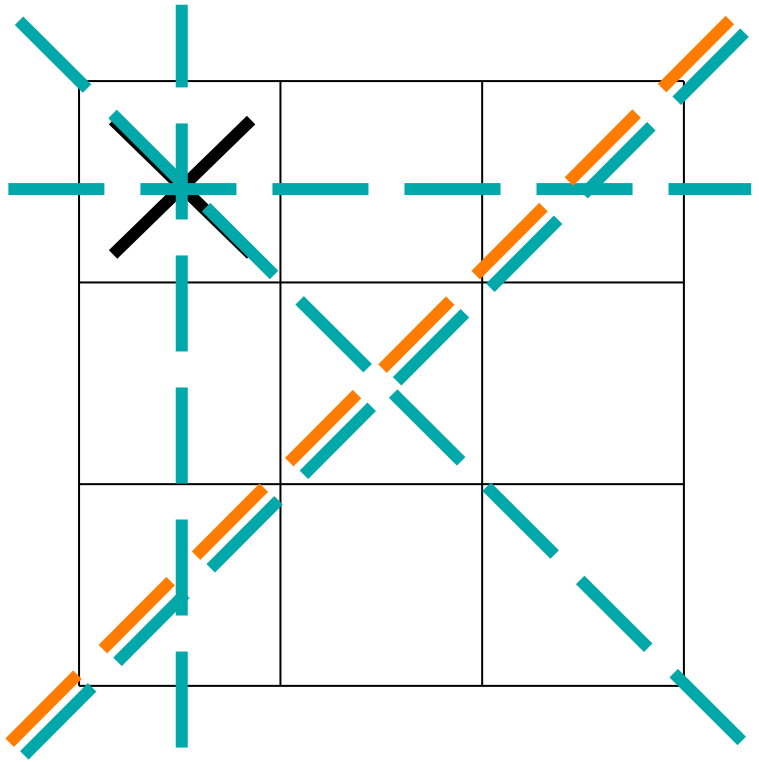


Lines having no sign could be taken by any of the players...

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:
the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.

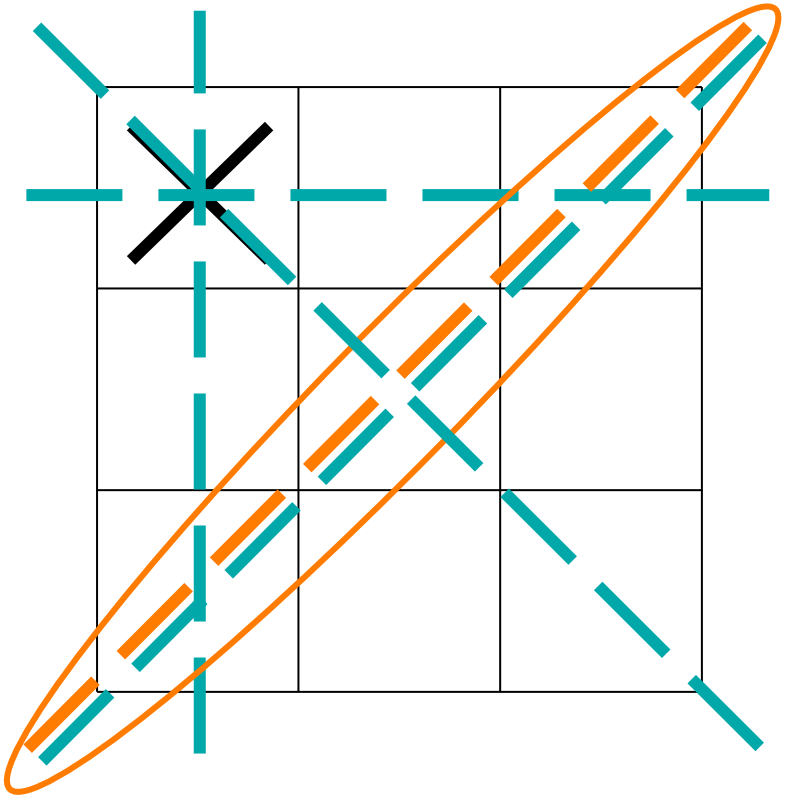


Lines having no sign could be taken by any of the players...

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:
the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.

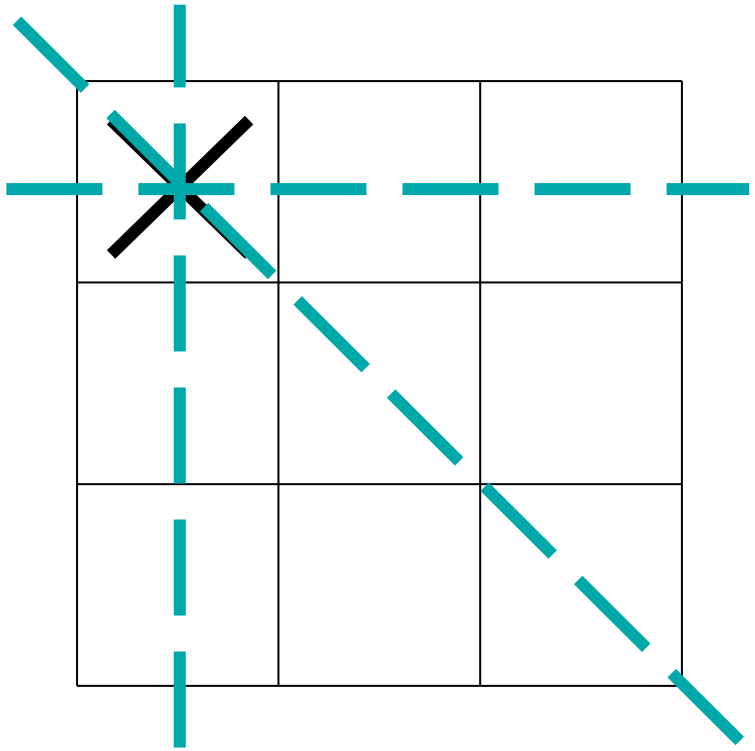


Lines having no sign could be taken by any of the players...

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

An example of an evaluation function:
the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.

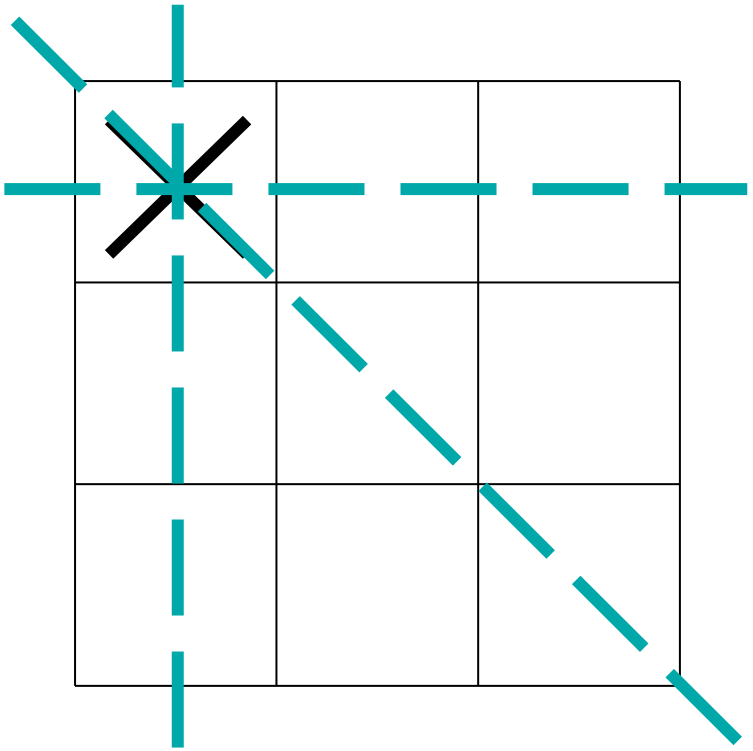


Lines having no sign could be taken by any of the players...

Evaluating a state

A state is better if it opens up more win possibilities until the end of the game.

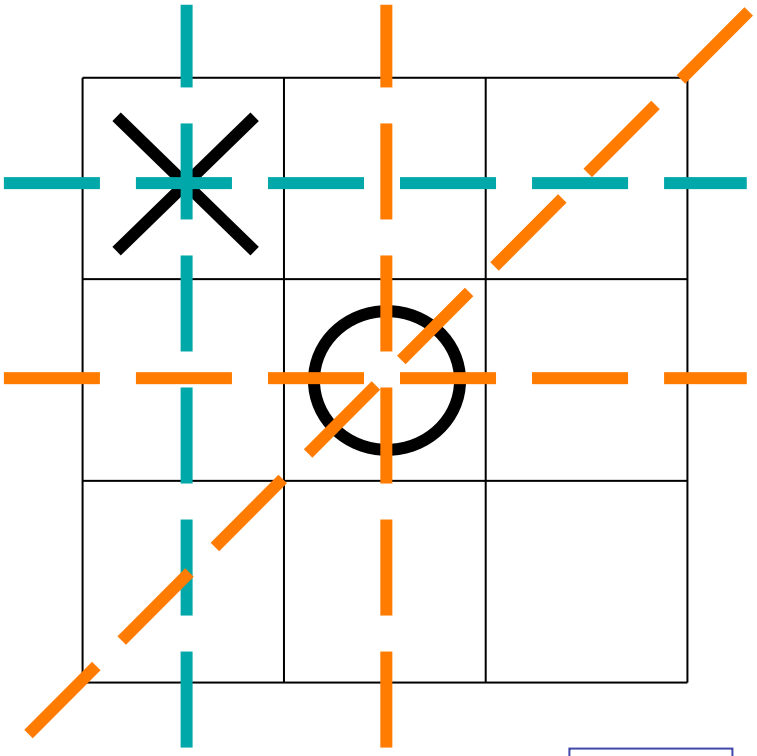
An example of an evaluation function: *the value of the state is the difference between the number of lines that MAX could fill in till the end of the game and those that MIN could fill in till the end of the game.*



Lines having no sign could be taken by any of the players...

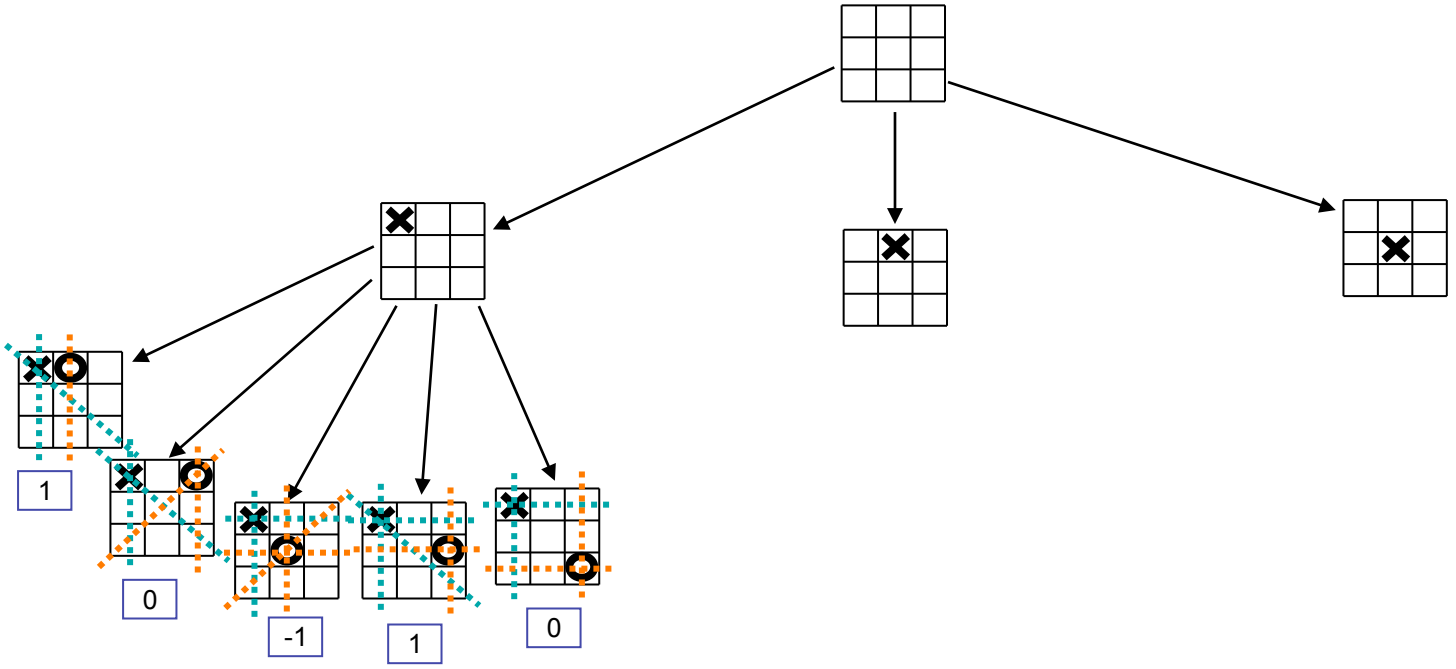
3 - 0 = 3

Evaluating a state



2 - 3 = -1

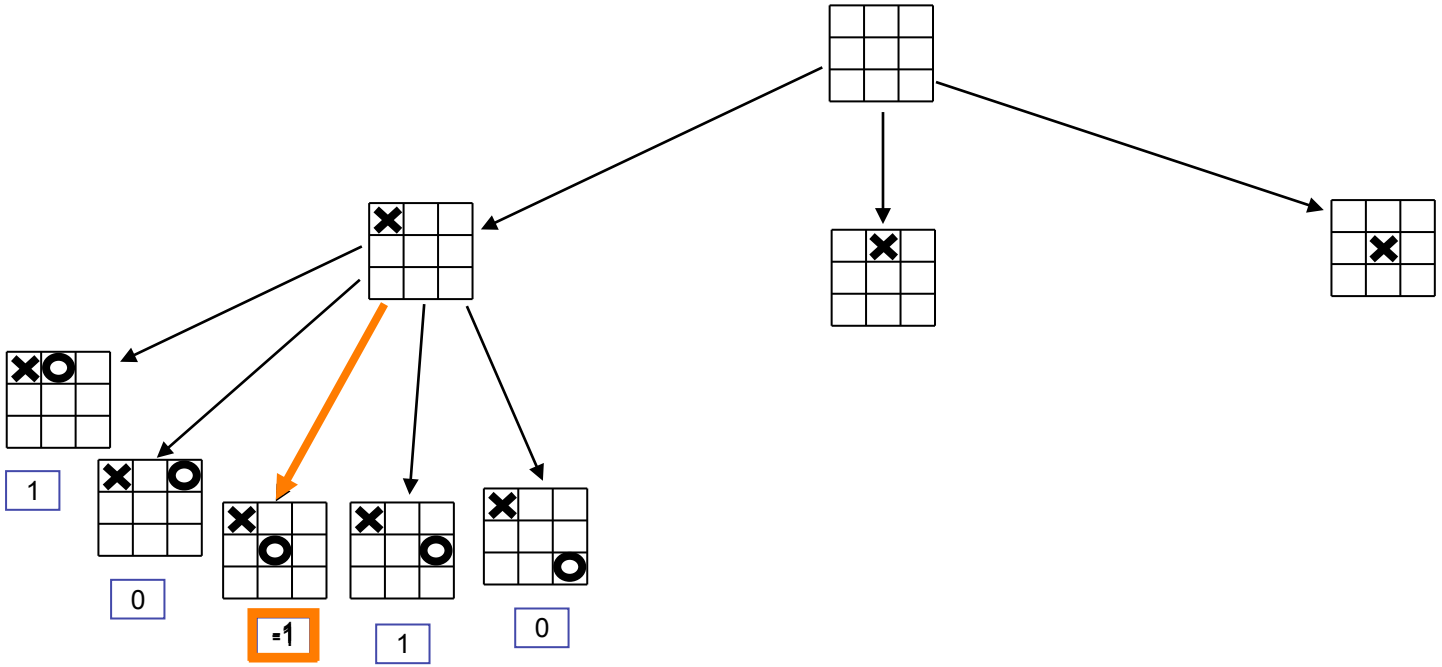
Evaluation: bottom-up



MAX

MIN

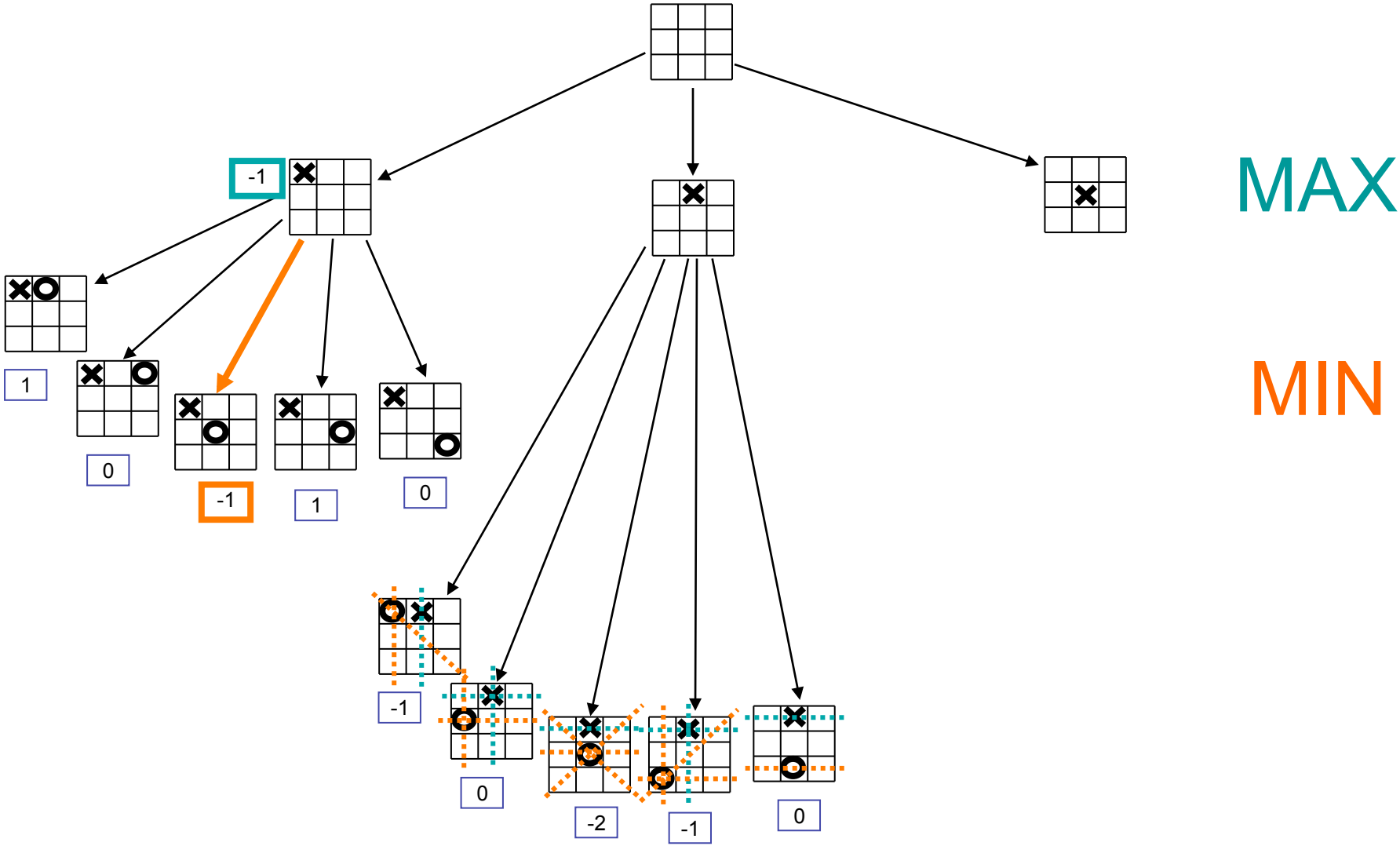
Evaluation: bottom-up



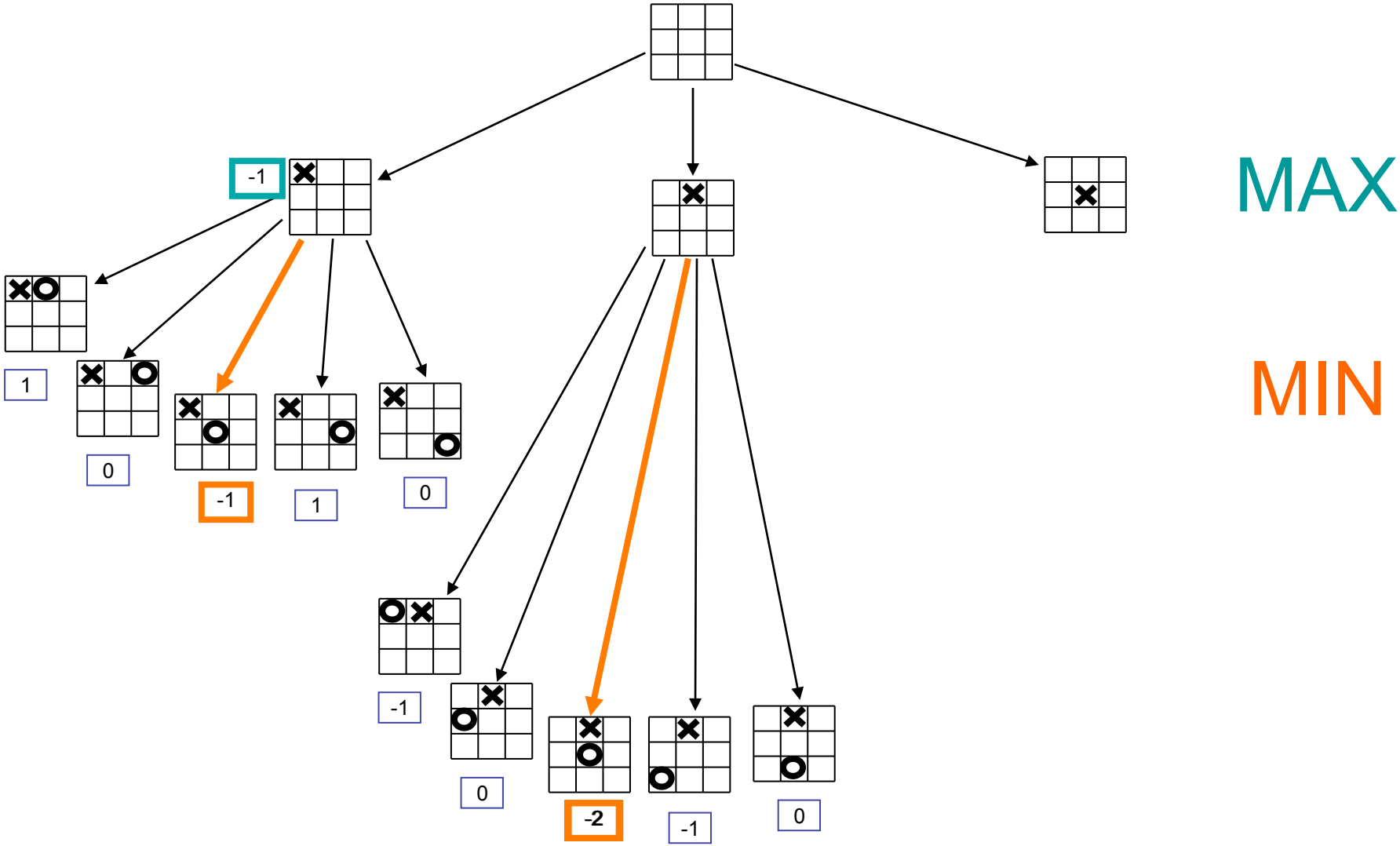
MAX

MIN

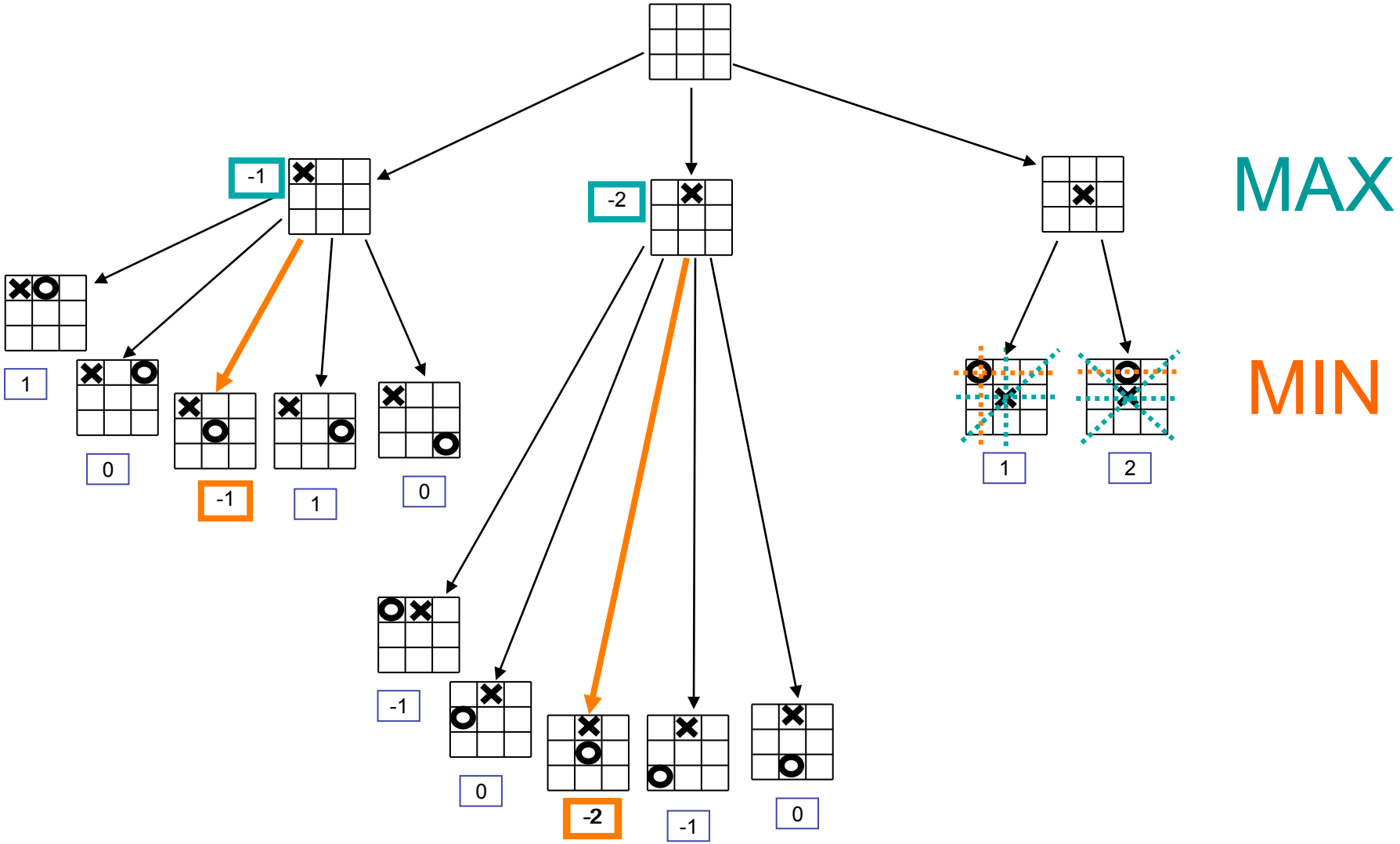
Evaluation: bottom-up



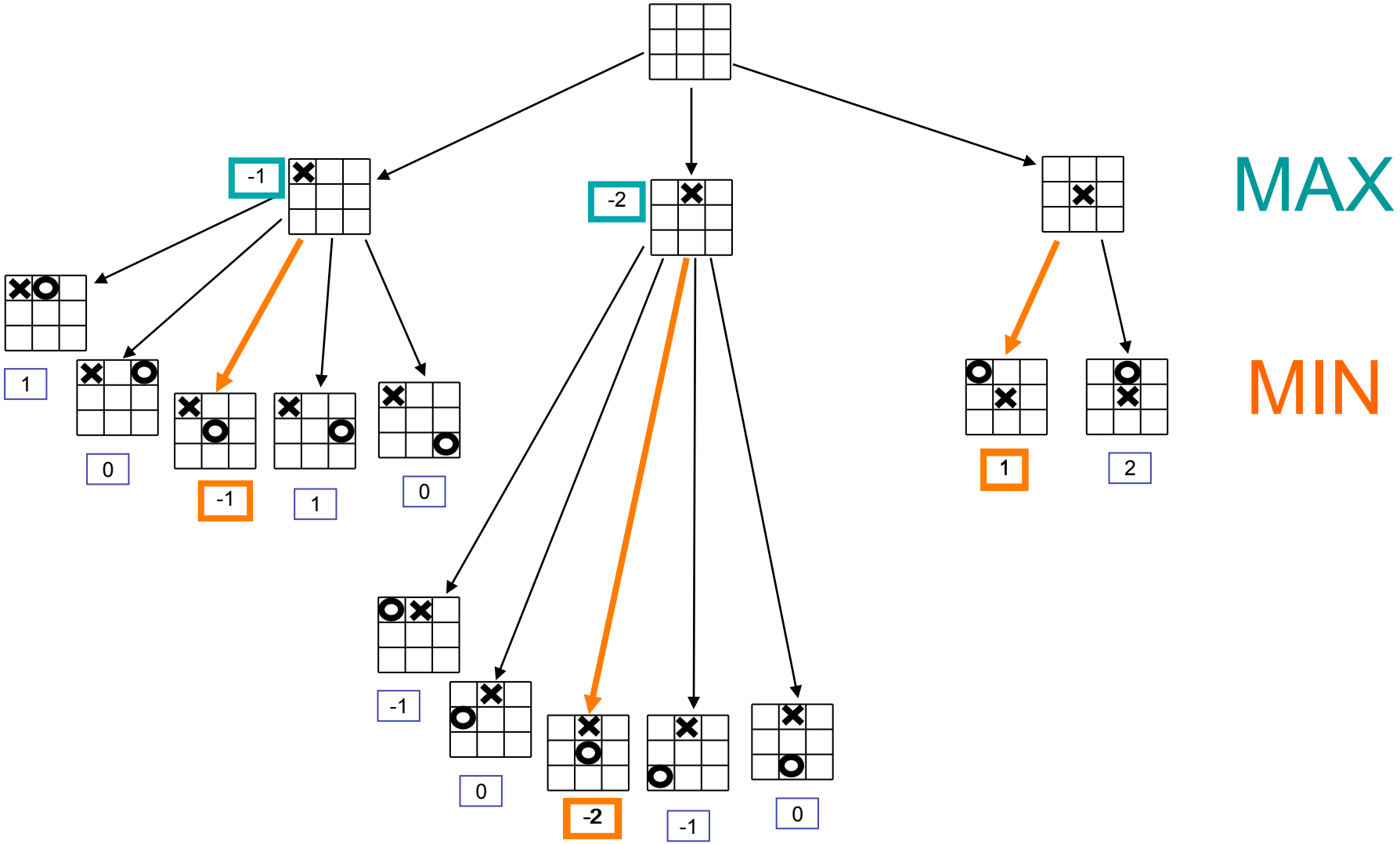
Evaluation: bottom-up



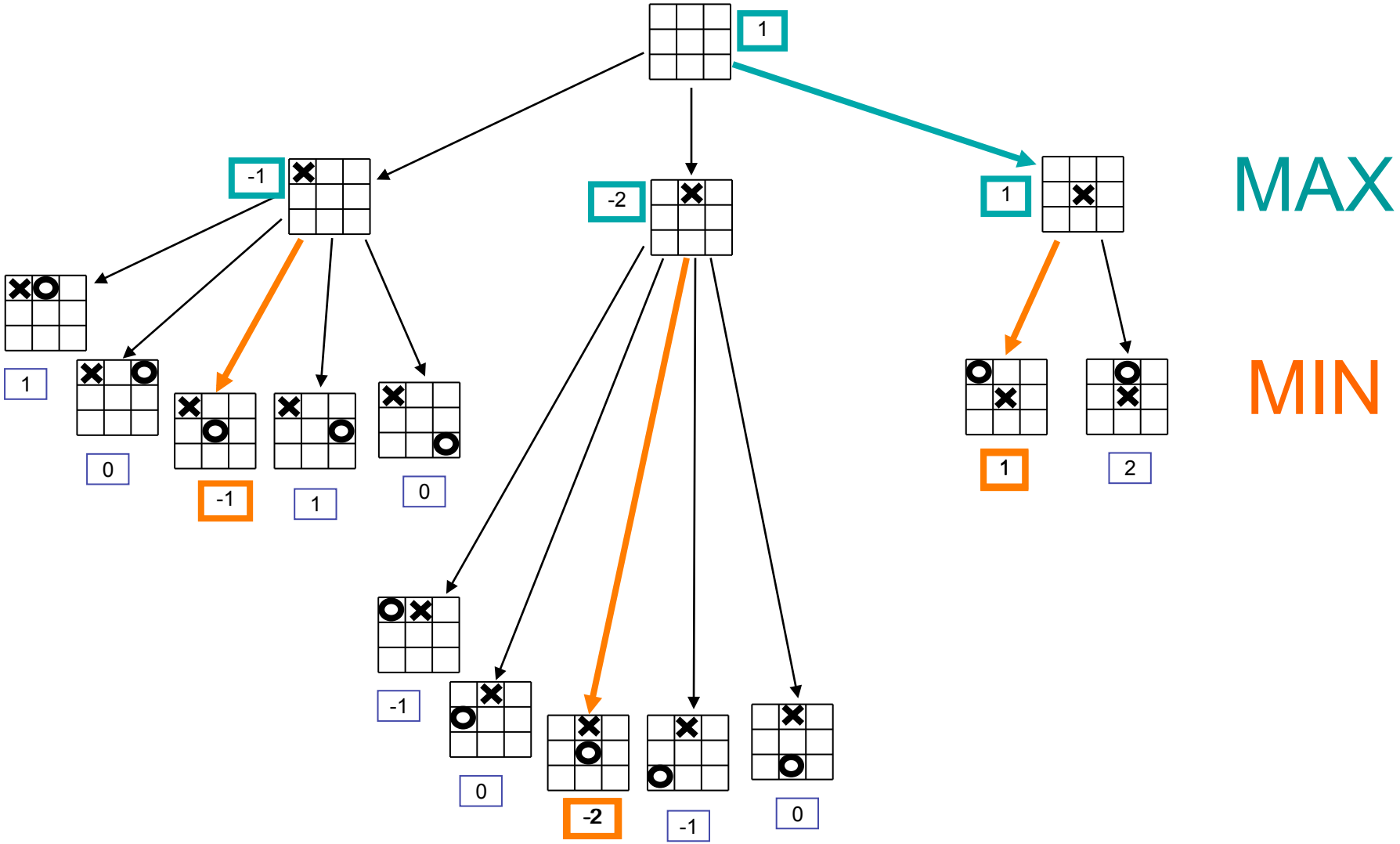
Evaluation: bottom-up



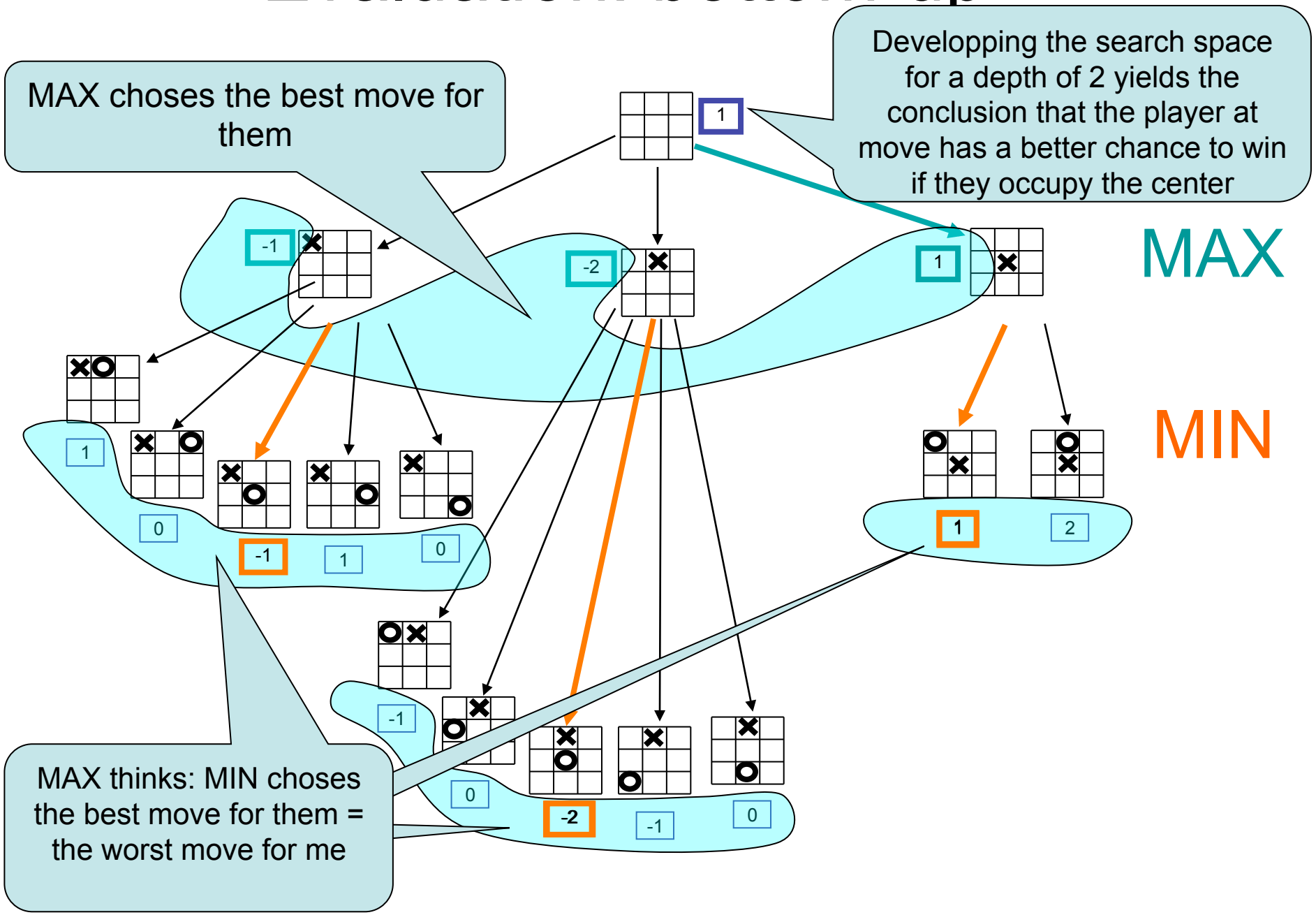
Evaluation: bottom-up



Evaluation: bottom-up



Evaluation: bottom-up



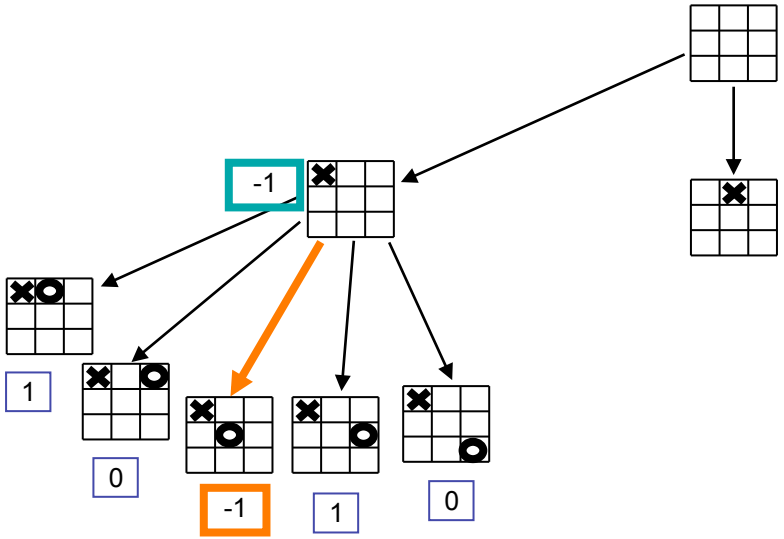
The MIN-MAX method

```
function min-max(state, player, depth)
begin
  if (depth = 0) then return score(state);
  val = worst(player);
  while (there are still states to generate) begin
    generate a state -> s;
    val <- back-up-compare(val, min-max(s, not(player), depth-1), player);
              // the following instruction reduces the search space in case a win is reached in one of the generated states:
    if (val = -worst(player)) return(val);
  end
  return(val);
end

function worst(player)
begin
  if player = MAX then return -∞;
  else return +∞;
end

funtion back-up-compare(val1, val2, player)
begin
  if player = MAX then return max(val1, val2);
  else return min(val1, val2);
end
```

The initial call:

min-max( ,MAX,2)

Evaluation: bottom-up



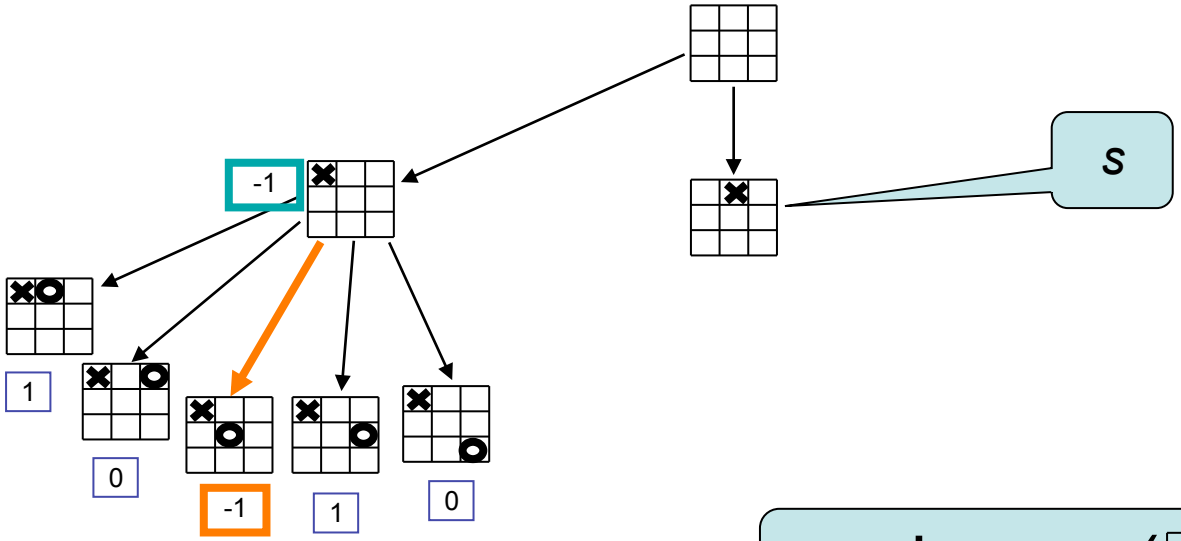
MAX

MIN

val=-1; player = MAX; depth=1;

while (there are still states to generate) **begin**
    generate a state -> *s*;
    ...
end

Evaluation: bottom-up



MAX

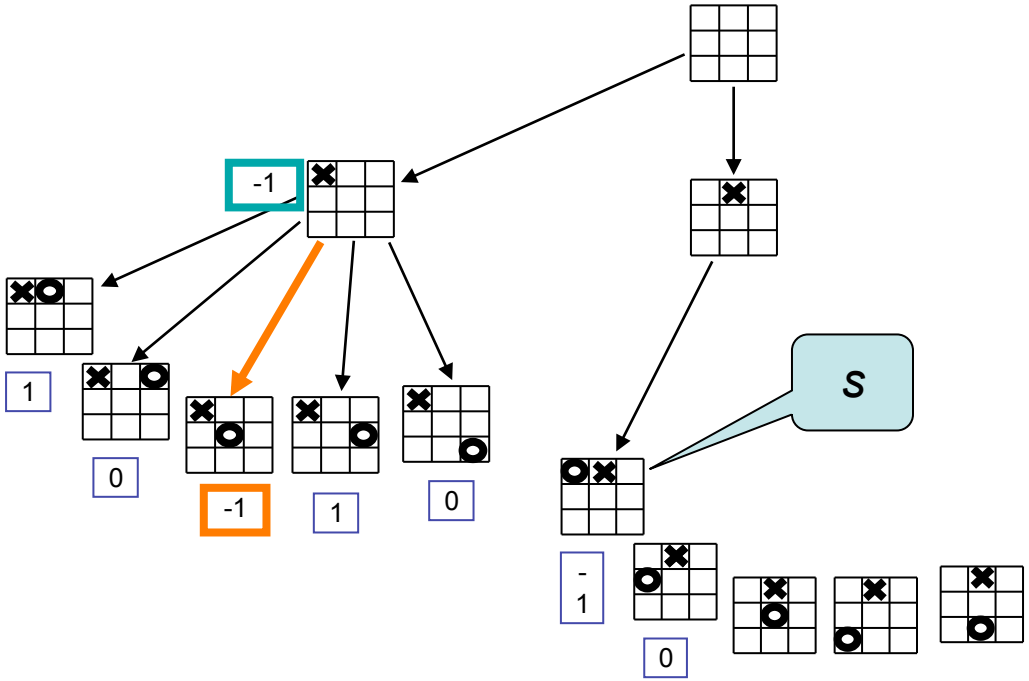
MIN

min-max(,MIN,1)

val=-1; player = MAX; depth=2;

```
while (there are still states to generate) begin
    generate a state -> s;
    val <- back-up-compare(val, min-max(s, not(player), depth-1), player);
    if (val = -worst(player)) return(val);
  end
```

Evaluation: bottom-up



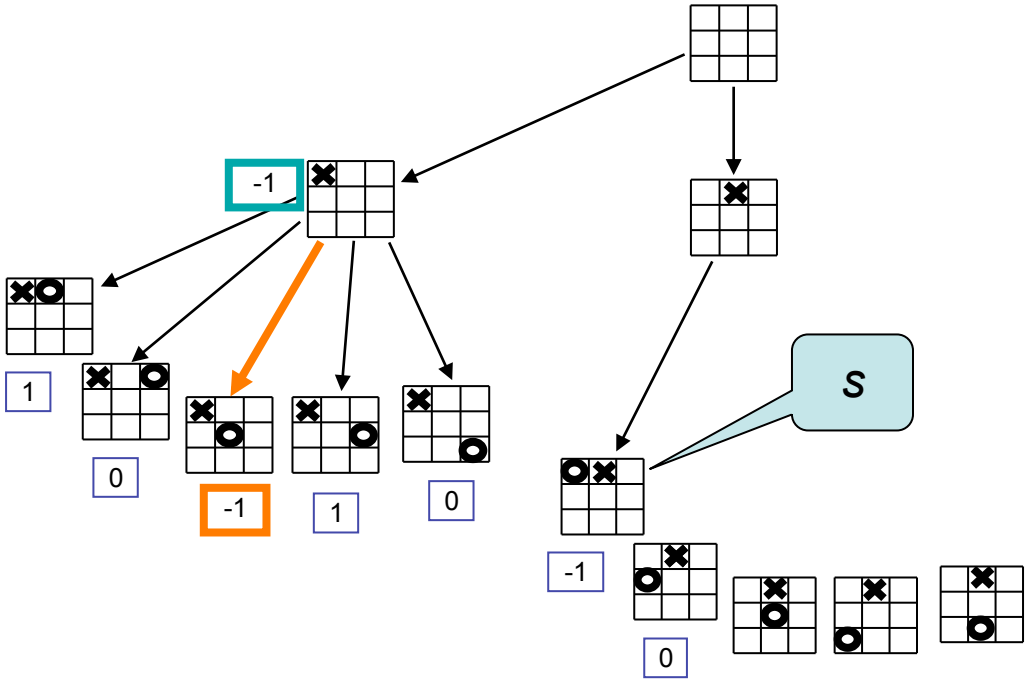
MAX

MIN

min-max( ,MIN,1)

```
val = worst(player);                                          val=∞
  while (there are still states to generate) begin
    generate a state -> s;
    val <- back-up-compare(val, min-max(s, not(player), depth-1), player);
    if (val = -worst(player)) return(val);
  end
```

Evaluation: bottom-up



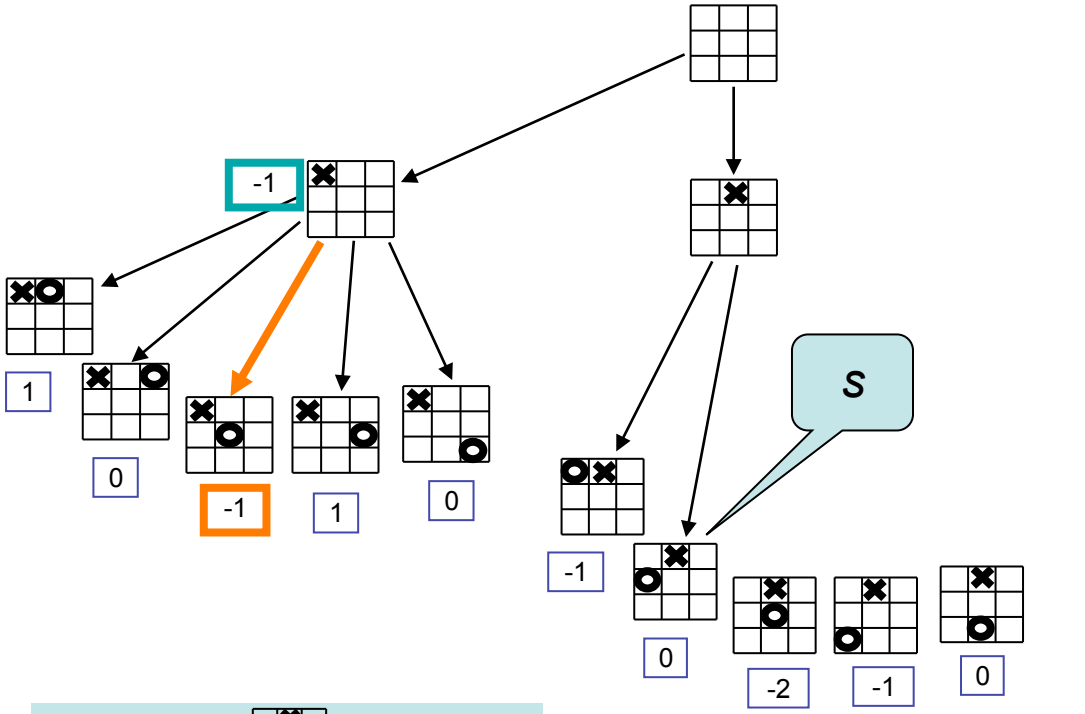
MAX

MIN

min-max( ,MAX,0)

if (*depth* = 0) **then return** score(*state*); → -1

Evaluation: bottom-up



MAX

MIN

min-max(,MIN,1)

val=∞; player=MIN;

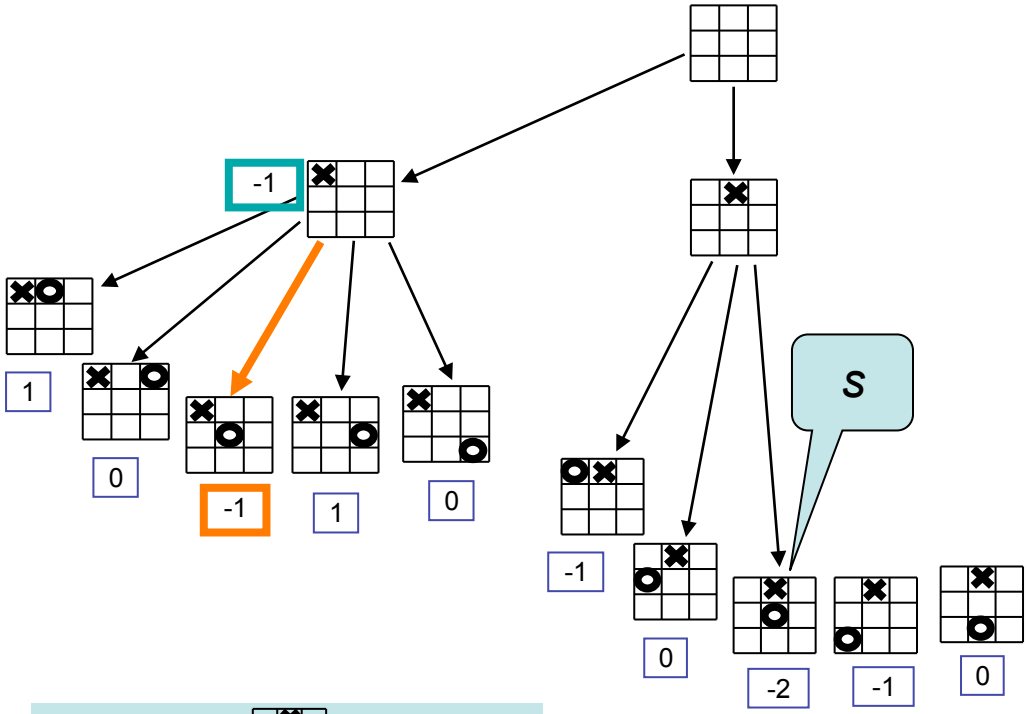
val <- back-up-compare(*val*, -1, *player*);

if (*val* = -worst(*player*)) **return**(*val*);

end

-1

Evaluation: bottom-up



MAX

MIN

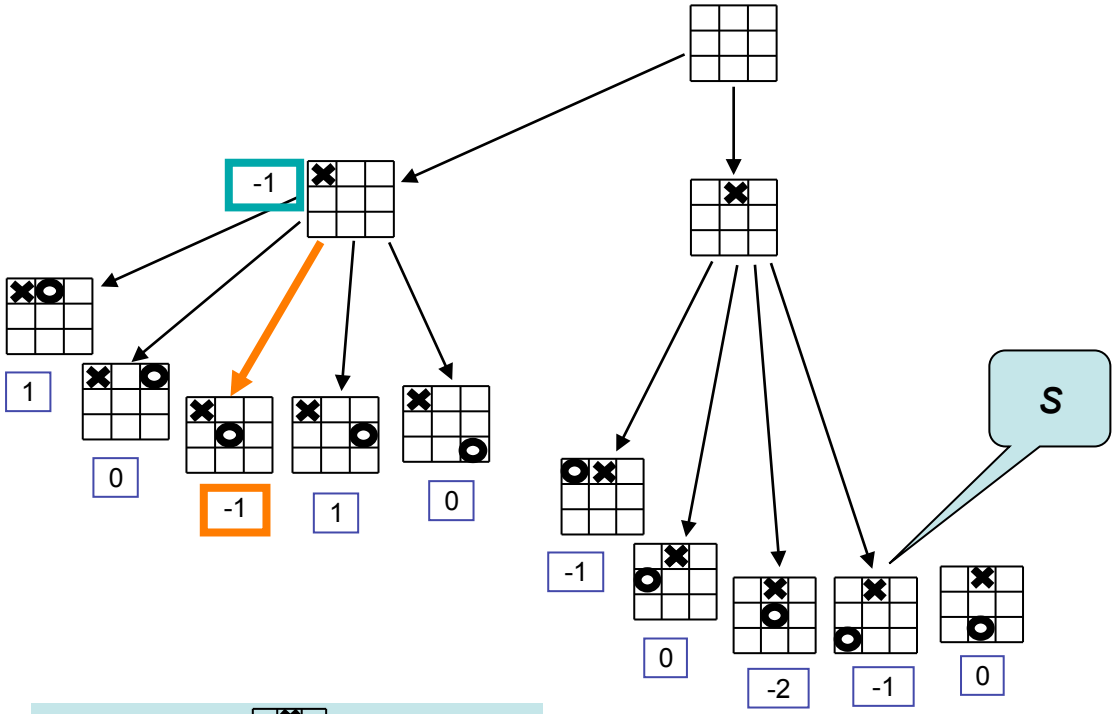
min-max(,MIN,1)

val=∞; player=MIN;

val <- back-up-compare(*val*, -1, *player*);
if (*val* = -worst(*player*)) **return**(*val*);
end

-2

Evaluation: bottom-up



MAX

MIN

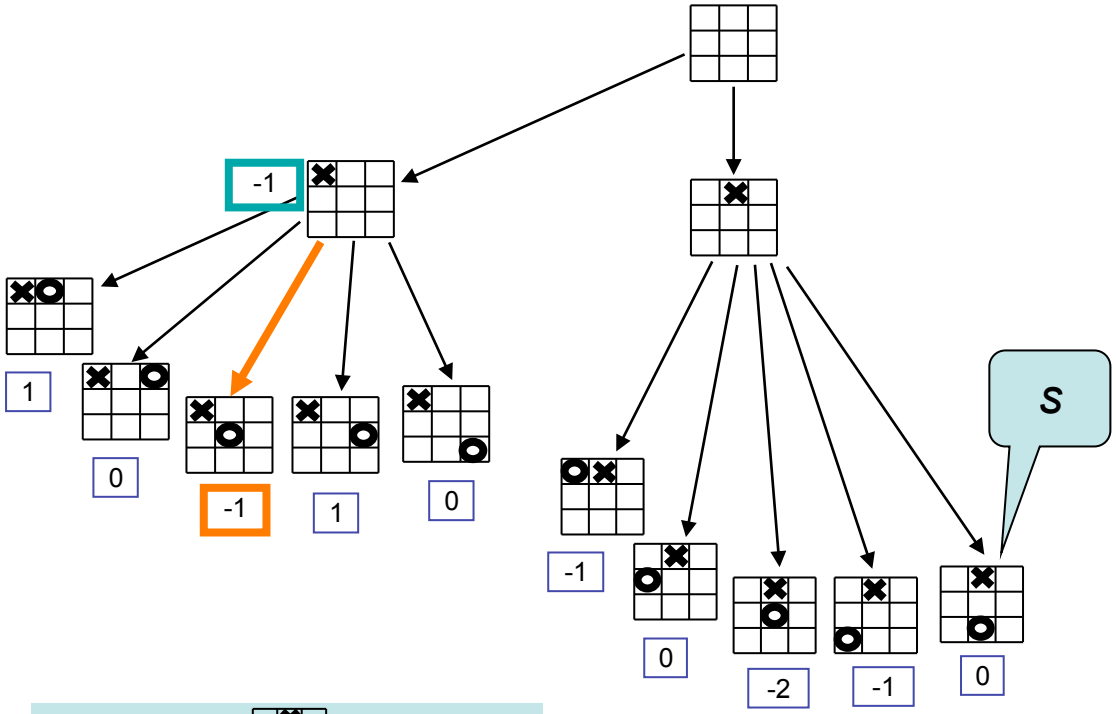
min-max(,MIN,1)

val=∞; player=MIN;

val <- back-up-compare(*val*, -1, *player*);
if (*val* = -worst(*player*)) **return**(*val*);
end

-2

Evaluation: bottom-up



MAX

MIN

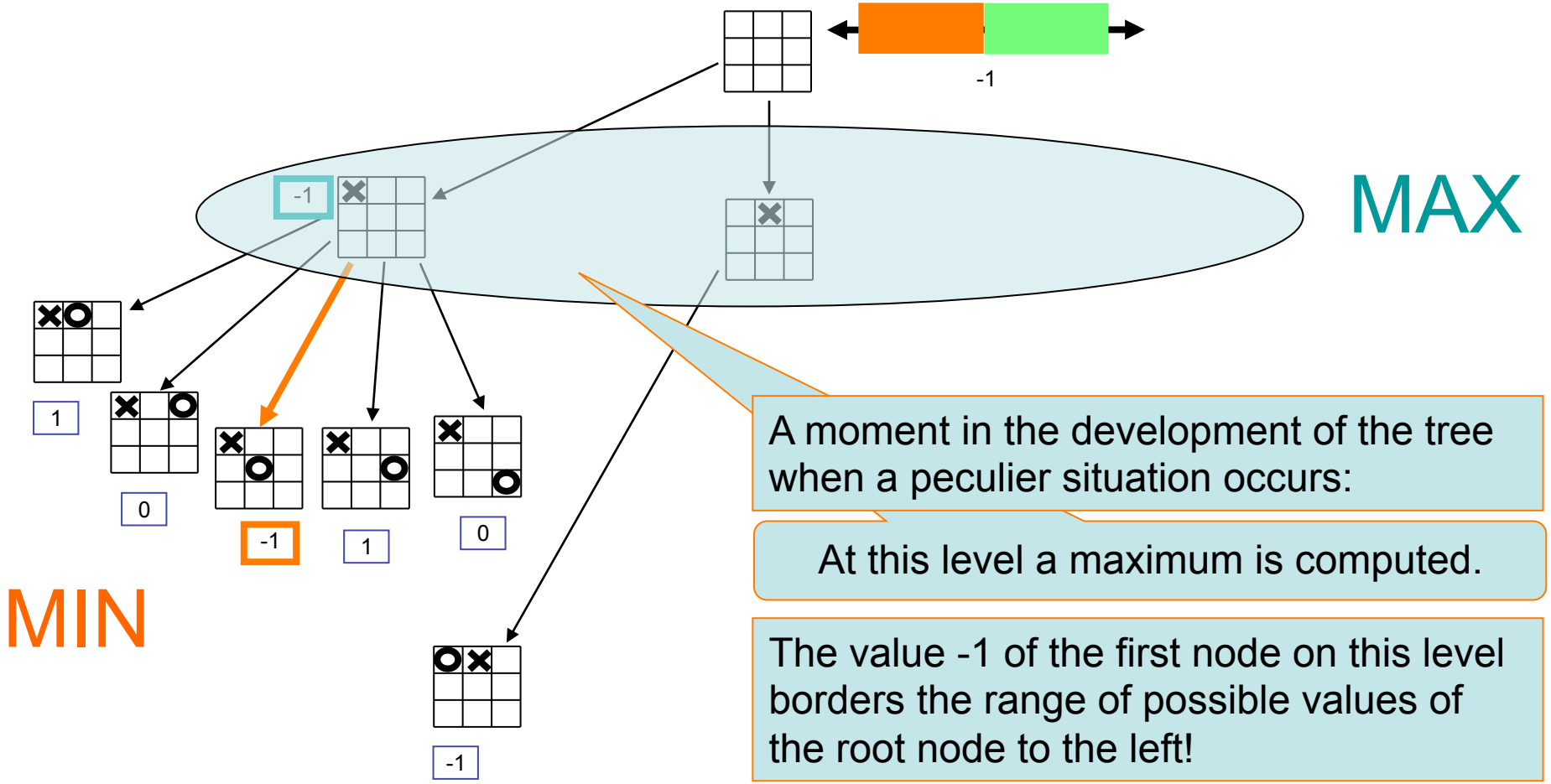
min-max(,MIN,1)

val=∞; player=MIN;

val <- back-up-compare(*val*, -1, *player*);
if (*val* = -worst(*player*)) **return**(*val*);
end

-2

The Alpha-Beta method

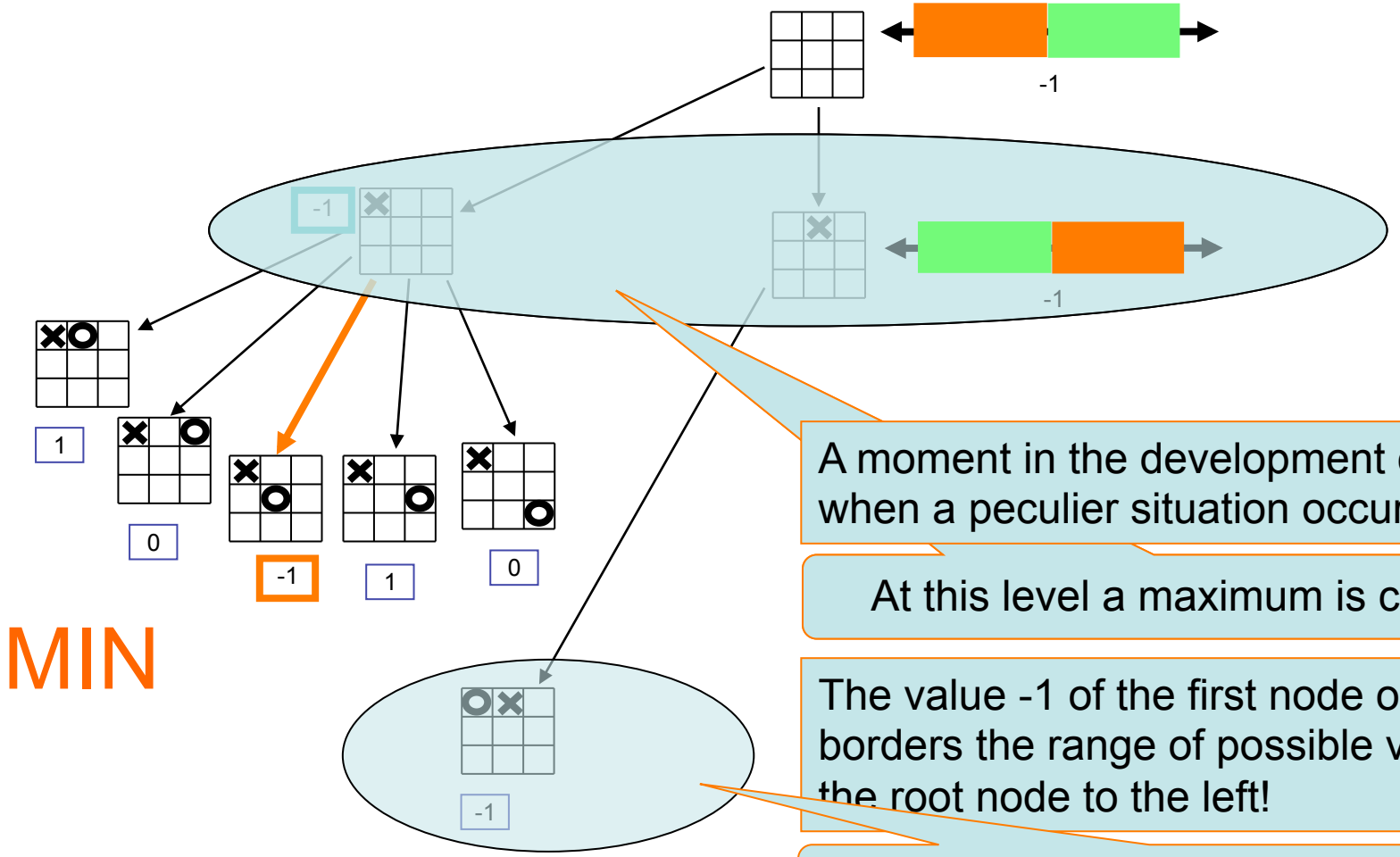


A moment in the development of the tree when a peculier situation occurs:

At this level a maximum is computed.

The value -1 of the first node on this level borders the range of possible values of the root node to the left!

The Alpha-Beta method



A moment in the development of the tree when a peculier situation occurs:

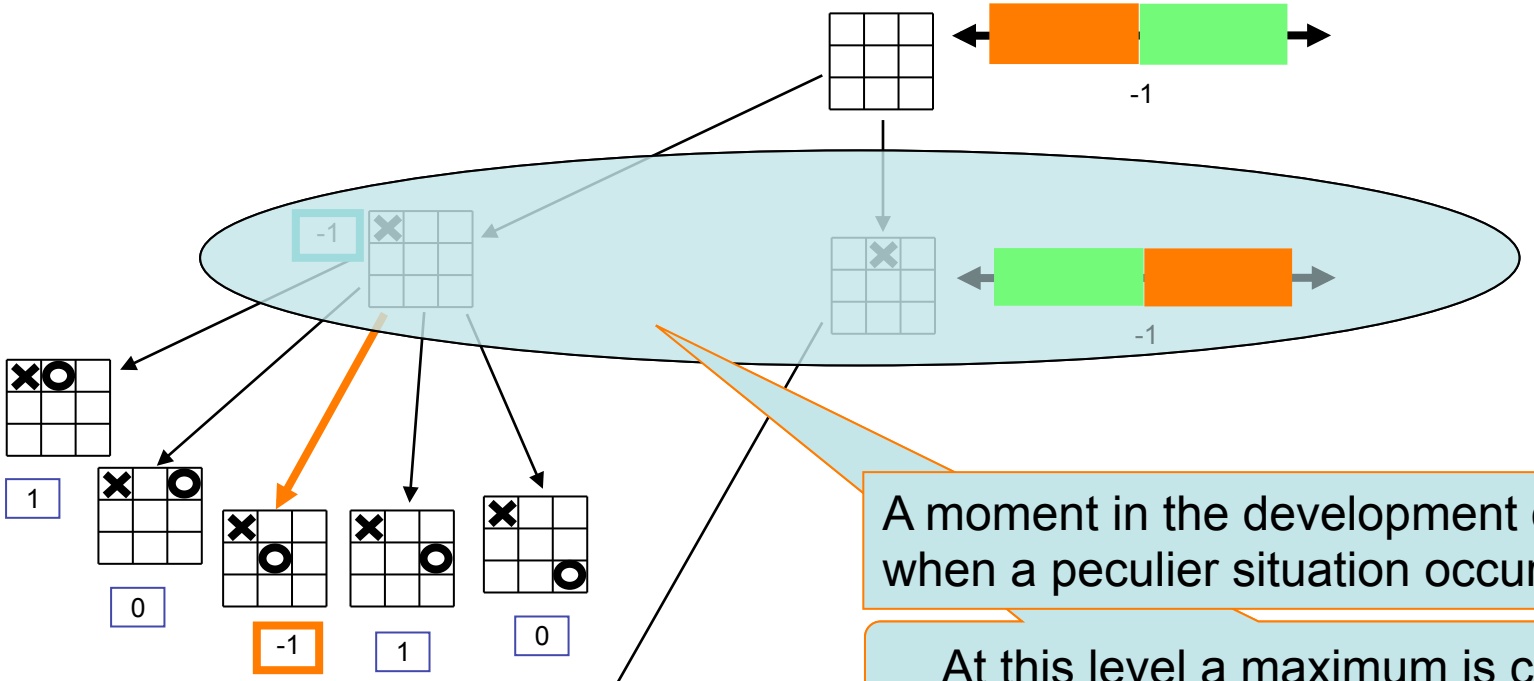
At this level a maximum is computed.

The value -1 of the first node on this level borders the range of possible values of the root node to the left!

At this level a minimum is computed.

Any value of the parent node could be less then or equal to -1.

The Alpha-Beta method



MAX

MIN

A moment in the development of the tree when a peculier situation occurs:

At this level a maximum is computed.

The value -1 of the first node on this level borders the range of possible values of the root node to the left!

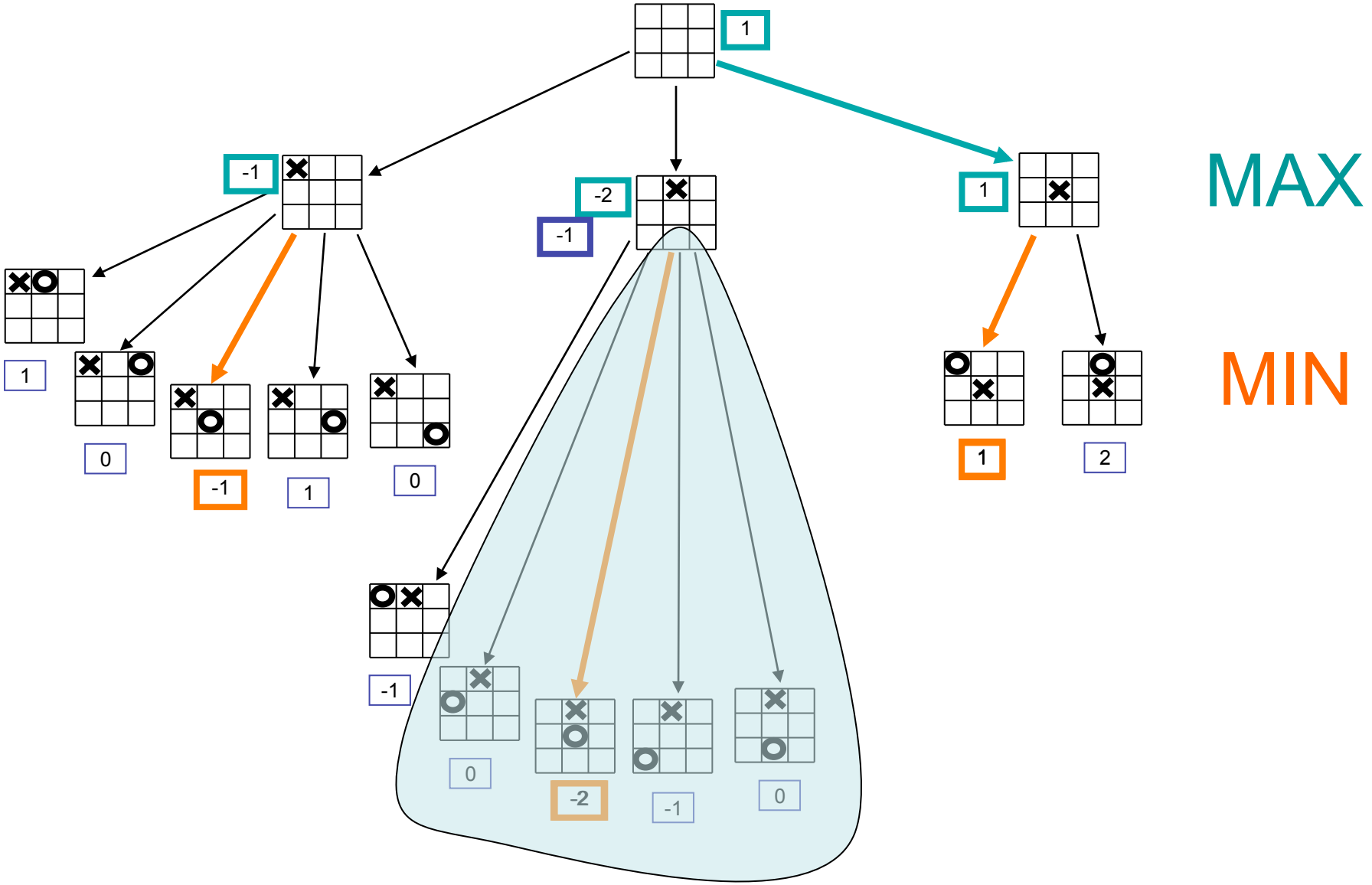
At this level a minimum is computed.

Any value of the parent node could be less then or equal to -1.

It can not influence eny longer the value of the root node.

The generation can now be stopped!

The Alpha-Beta method



The Alpha-Beta method

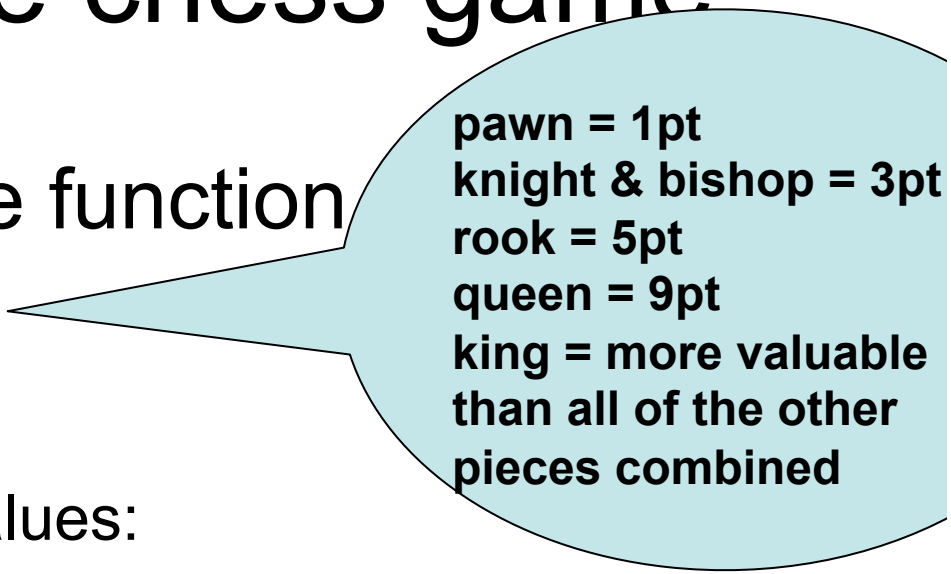
```
function alpha-beta(state, player, depth)
begin
  if (depth = 0) then return score(state);
  val = worst(player);
  while (there are still states to generate) begin
    generate a state -> s;
    newval <- alpha-beta(s, not(player), depth-1);
        if player=MAX & newval ≤ val then return(newval);
        else if player=MIN & newval ≥ val then return(newval);
        else val ← back-up-compare(val, min-max(s, not(player), depth-1), player);
                // the following instruction reduces the search space in case a win is reached in one of the generated states:
    if (val = -worst(player)) return(val);
  end
  return(val);
end

function worst(player)
begin
  if player = MAX then return -∞;
  else return +∞;
end

function back-up-compare(val1, val2, player)
begin
  if player = MAX then return max(val1, val2);
  else return min(val1, val2);
end
```

The initial call:

alpha-beta ( ,MAX,2)

Solving P8: the chess game

- Choose a state-score function
  - give values to pieces
  - evaluate a state
    - dynamic change of values:
      - in the endgame the king is more powerful than a bishop or knight but less powerful than a rook.
      - advanced pawns are more valuable than those on their initial squares,
      - coordination between pieces (e.g. a pair of bishops usually coordinate better than a bishop and a knight),
      - type of position (e.g. knights are generally better in closed positions with many pawns while bishops are more powerful in open positions)



Develop instantaneous game trees

- Compute short-term actions
  - develop the game tree from the current position for a given depth
  - use MIN-MAX and alpha-beta to choose the best move

Tactics

- Simple one-move or two-move tactical actions:
  - threats, exchanges of material, double attacks, etc.
  - implement tactical moves (see https://en.wikipedia.org/wiki/Chess#Fundamentals_of_tactics)

Strategies

- Setting up goals and long-term plans
- Control the center and centralization
- The pawn structure (https://en.wikipedia.org/wiki/Pawn_structure), king safety, control of key squares or groups of squares

Interface

- Explain why a move is better than another
  - by visually showing short-term engagements
  - by generating explanations in natural language
    - invent a controlled language and parameterize it
    - generate sentences that “read” the sequence of movements of a game