RUTGERS

THE STATE UNIVERSITY
OF NEW JERSEY

Adversarial Search

Games

Edgar Granados

Games

Properties

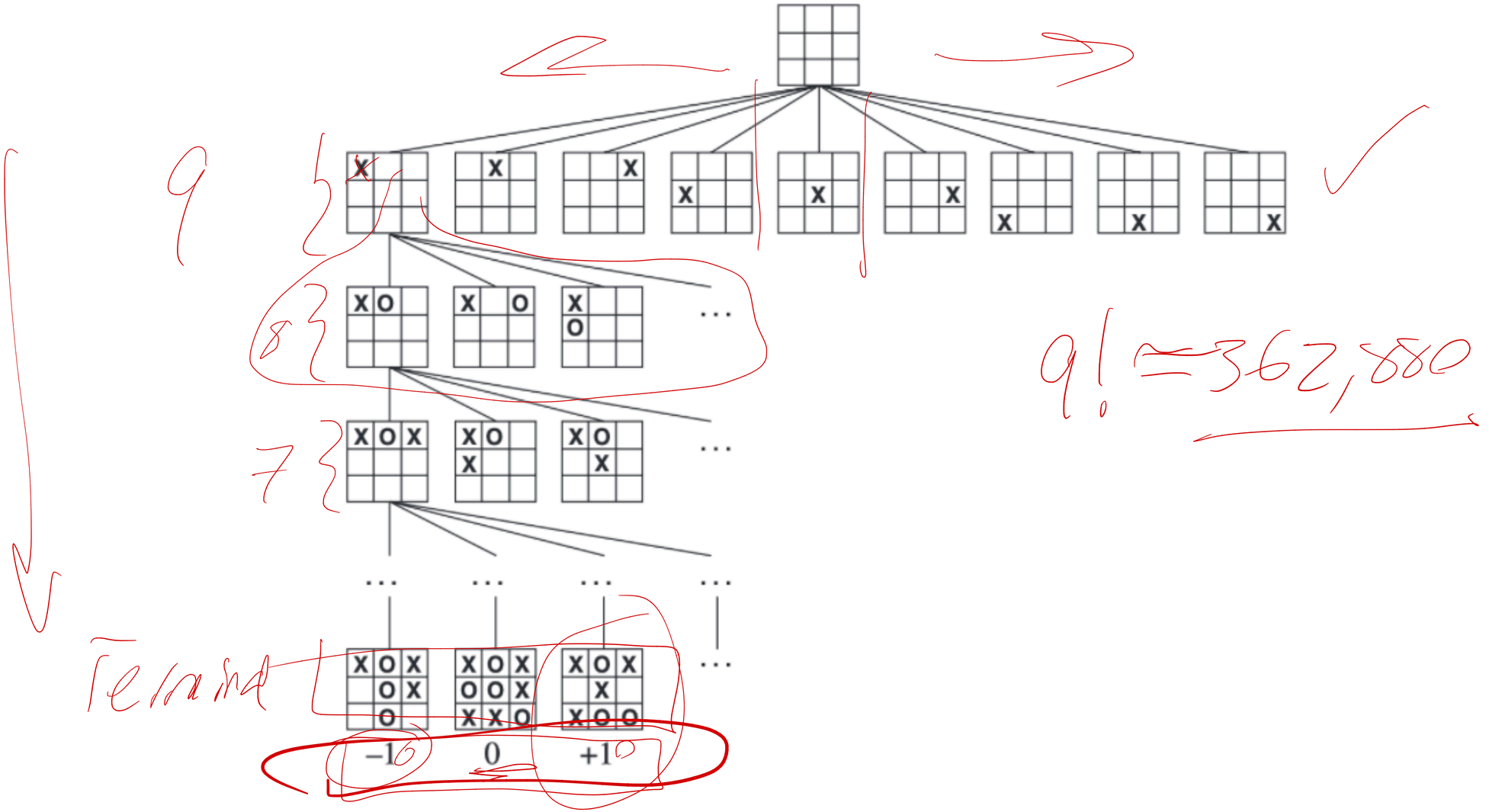
- Full Observability
- Multi-agent
- Deterministic
- Sequential
- Discrete
- Known
- **Zero-sum**

Game

- **Initial State (S_0)**: Where the game starts
- **To-Move(s):** The player whose turn it is to move at state s
- **Actions**: Set of legal moves
- **Transition model**: The effect of applying the action
- **Is-Terminal(s):** Returns true if the game is over
- **Utility(s,p)**: A function that assigns a numeric value to the final state

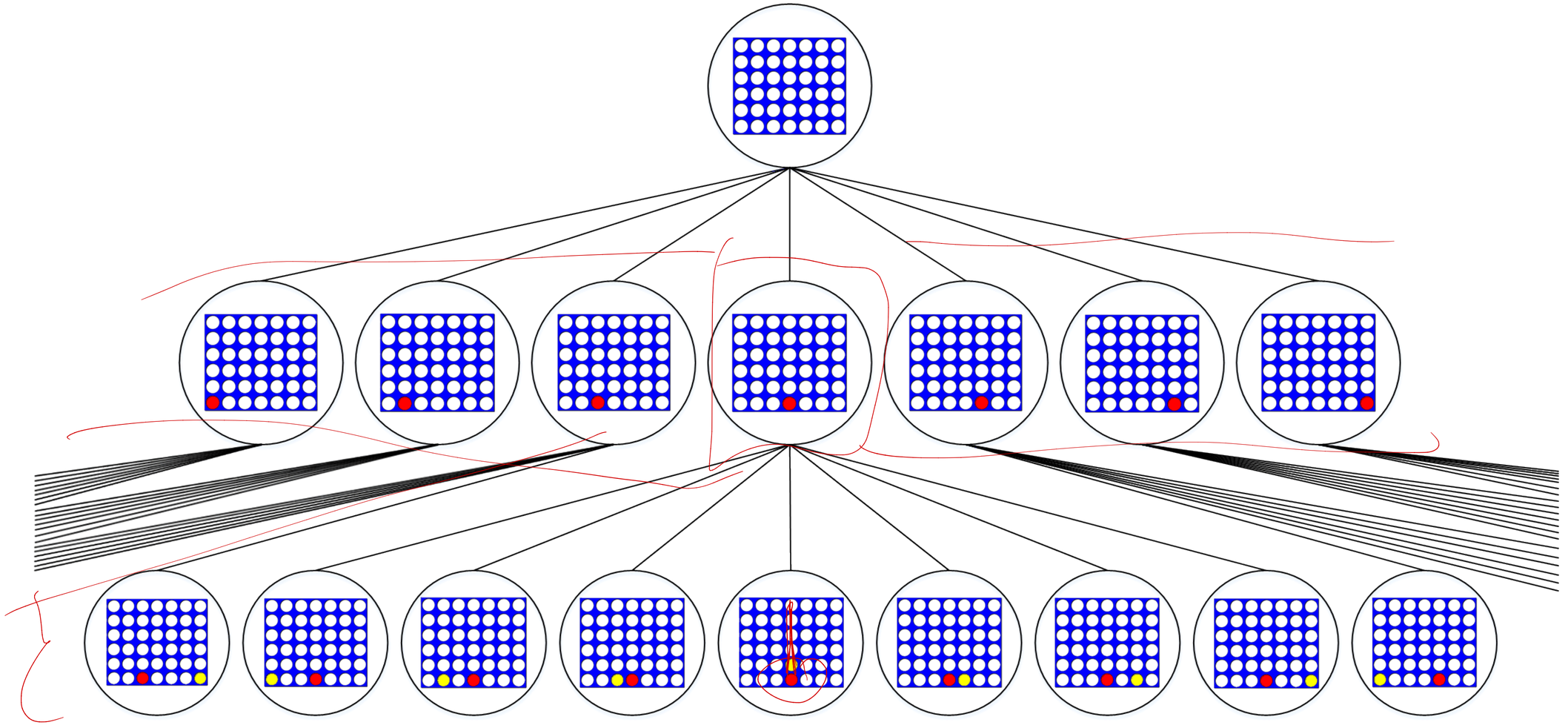
Examples

Tic-Tac-Toe



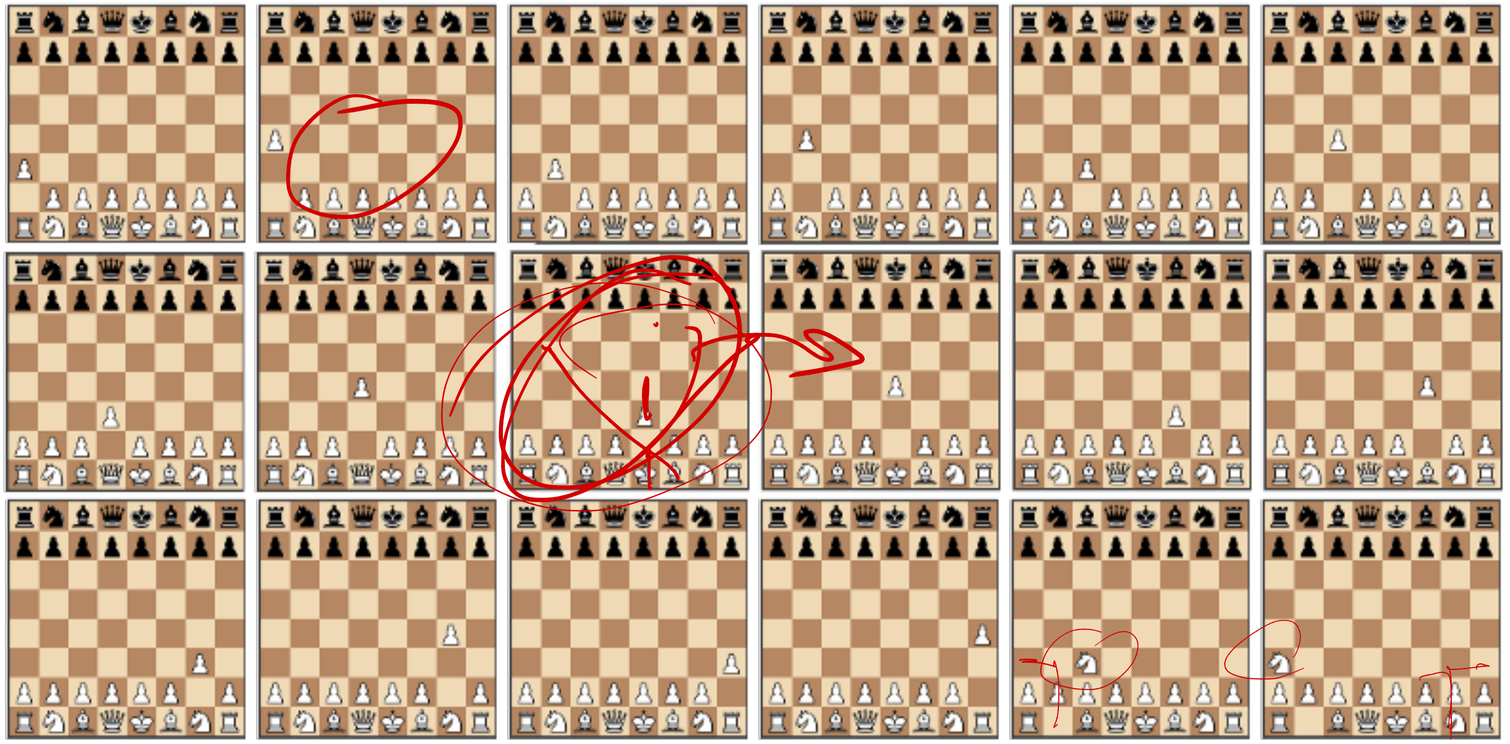
Examples

Connect 4

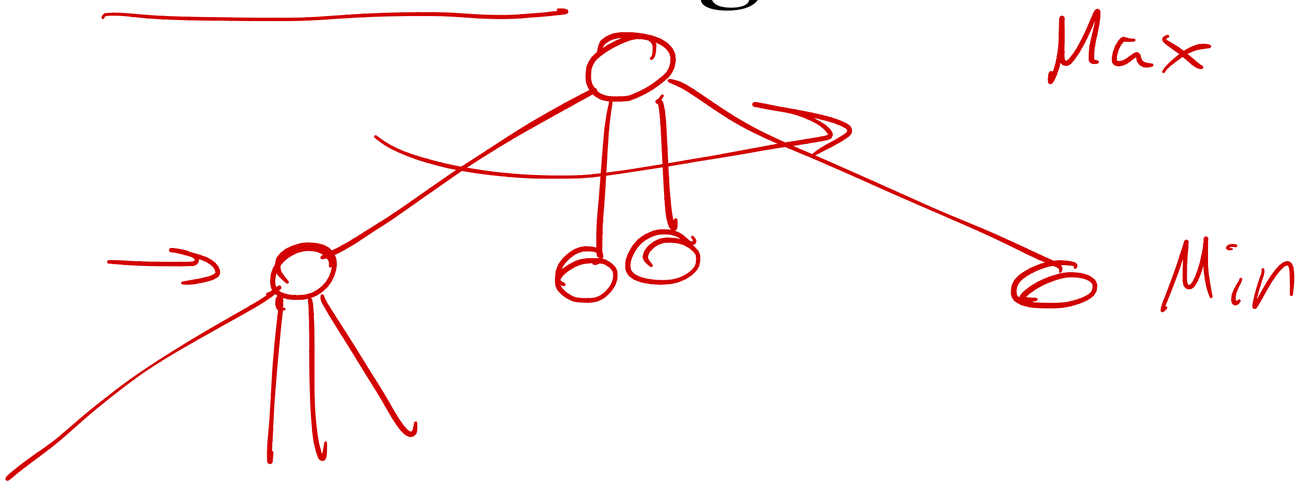


Examples

Chess



MinMax Algorithm

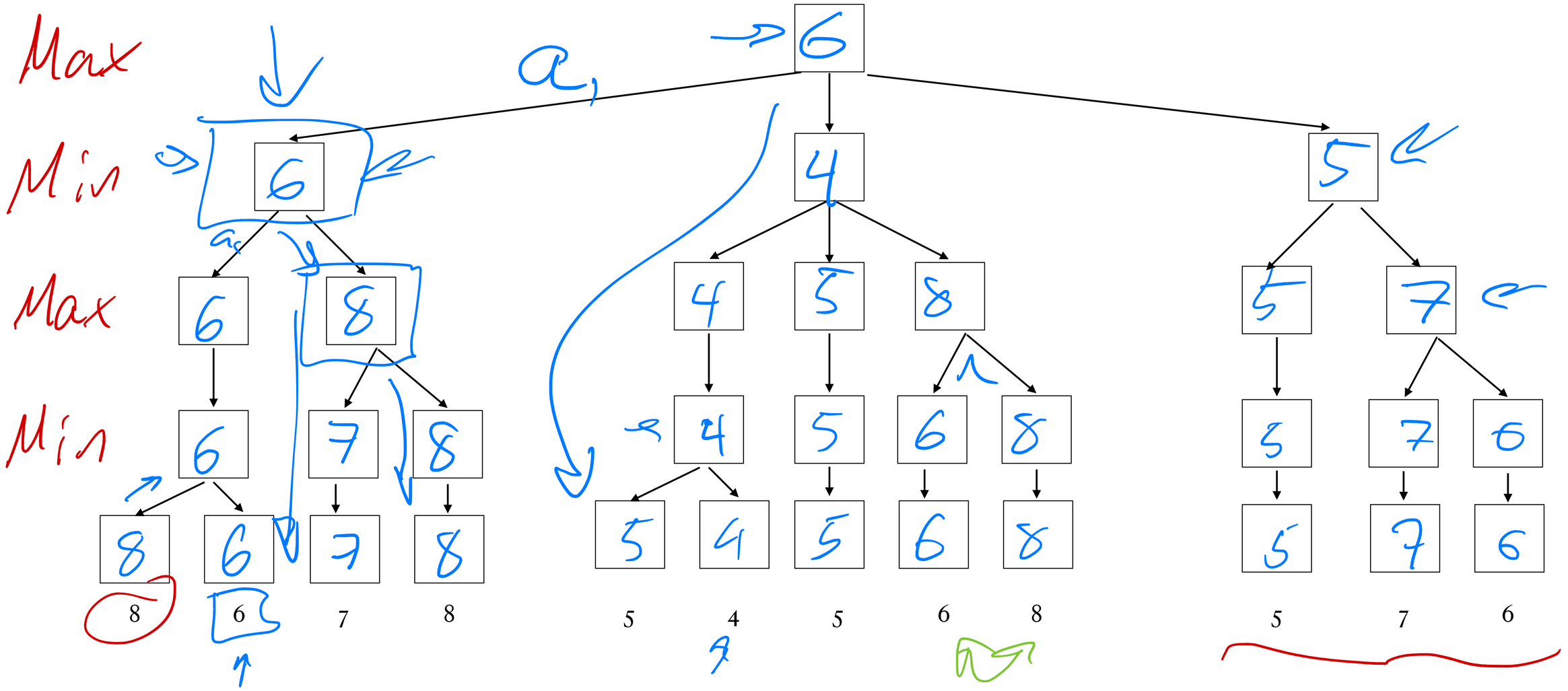


DFS

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = MAX, \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if PLAYER}(s) = MIN. \end{cases}$$

MinMax Algorithm

Example: 2 players



MinMax Algorithm

Pseudocode

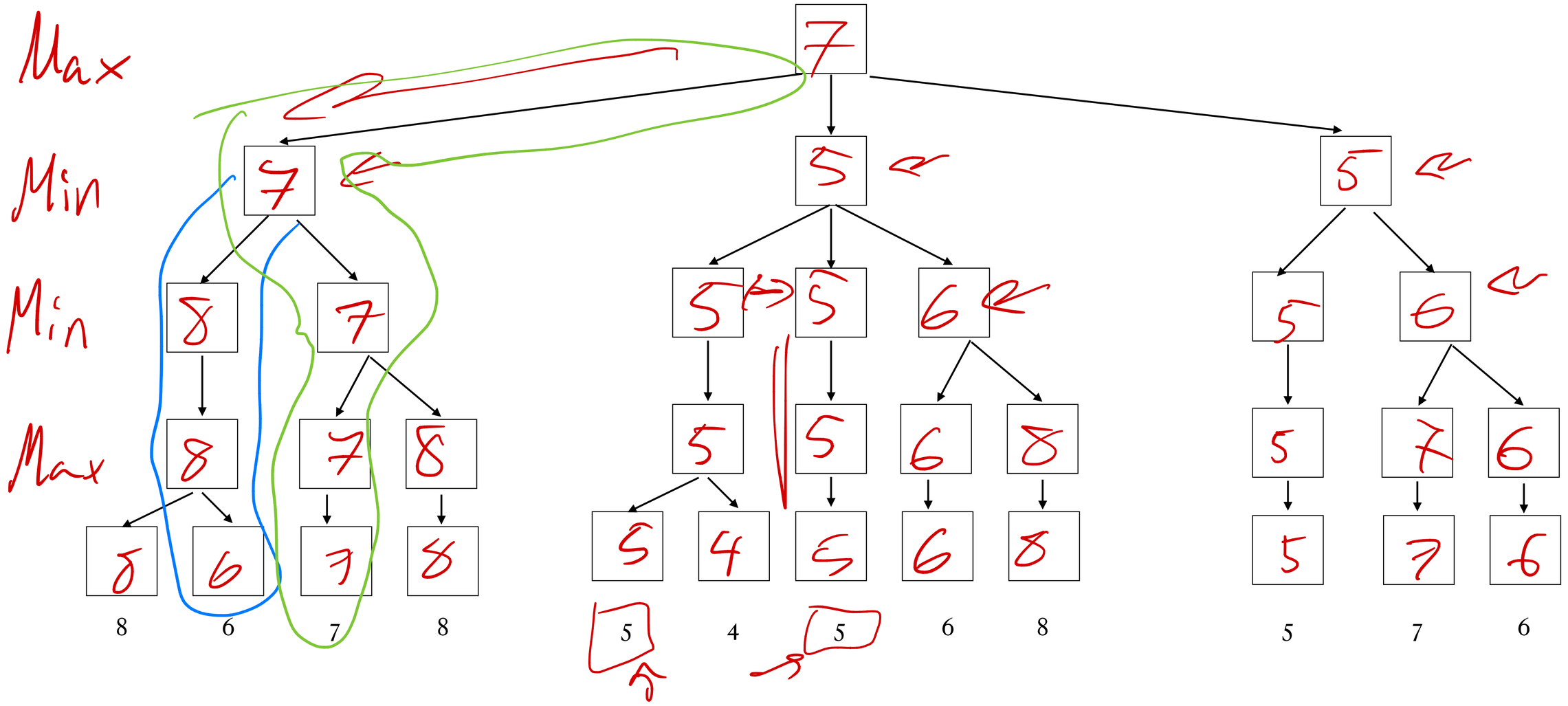
```
function MINIMAX-SEARCH(game, state) returns an action
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state)
    return move

function MAX-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v, move ← −∞, null
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
        if v2 > v then
            v, move ← v2, a
    return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v, move ← +∞, null
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
        if v2 < v then
            v, move ← v2, a
    return v, move
```

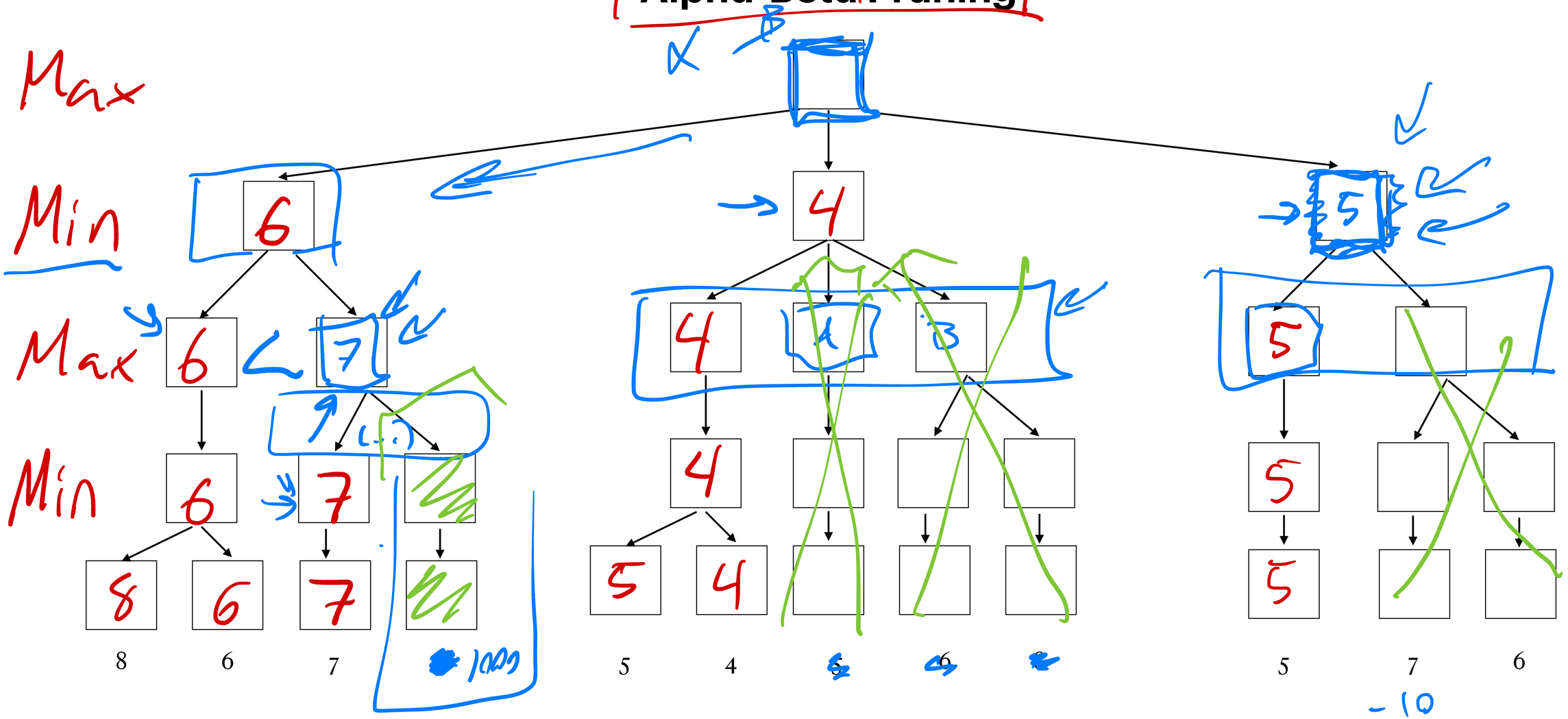
MinMax Algorithm

Example: 3 players



MinMax Algorithm

Alpha-Beta Pruning



MinMax Algorithm

Alpha-Beta Pruning: Pseudocode

Original

```
function MINIMAX-SEARCH(game, state) returns an action
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state)
    return move

function MAX-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v, move ← −∞, null
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
        if v2 > v then
            v, move ← v2, a
    return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v, move ← +∞, null
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
        if v2 < v then
            v, move ← v2, a
    return v, move
```

α, β

```
function ALPHA-BETA-SEARCH(game, state) returns an action
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state, −∞, +∞)
    return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← −∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
        if v2 > v then
            v, move ← v2, a
            α ← MAX(α, v)
        if v ≥ β then return v, move
    return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← +∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
        if v2 < v then
            v, move ← v2, a
            β ← MIN(β, v)
        if v ≤ α then return v, move
    return v, move
```

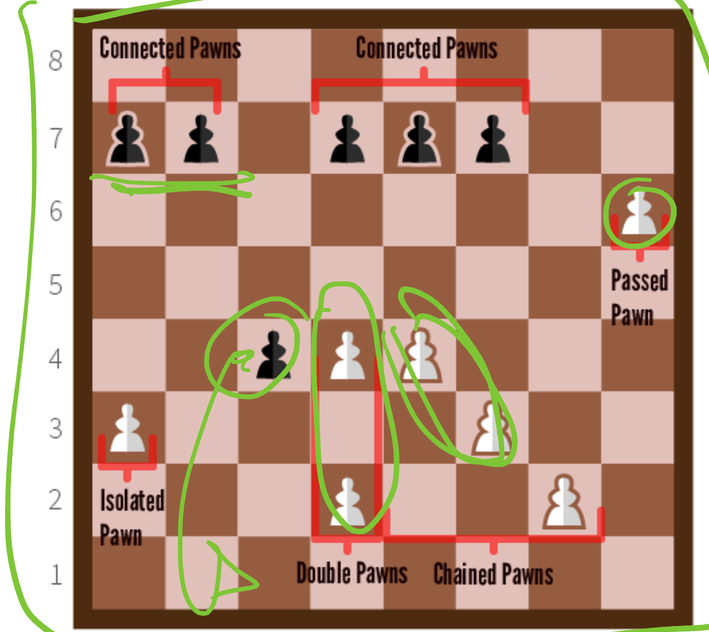
MinMax Algorithm

Possible Improvements

- Heuristic Alpha-Beta Search

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) = true, \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = MAX, \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = MIN. \end{cases}$$

Pieces and Point Value		
Pawn		1
Knight		3
Bishop		3
Rook		5
Queen		9
King		priceless



MinMax Algorithm

Possible Improvements

- Use Iterative Deepening Depth-First Search
  - Transposition Table

MinMax Algorithm

Possible Improvements

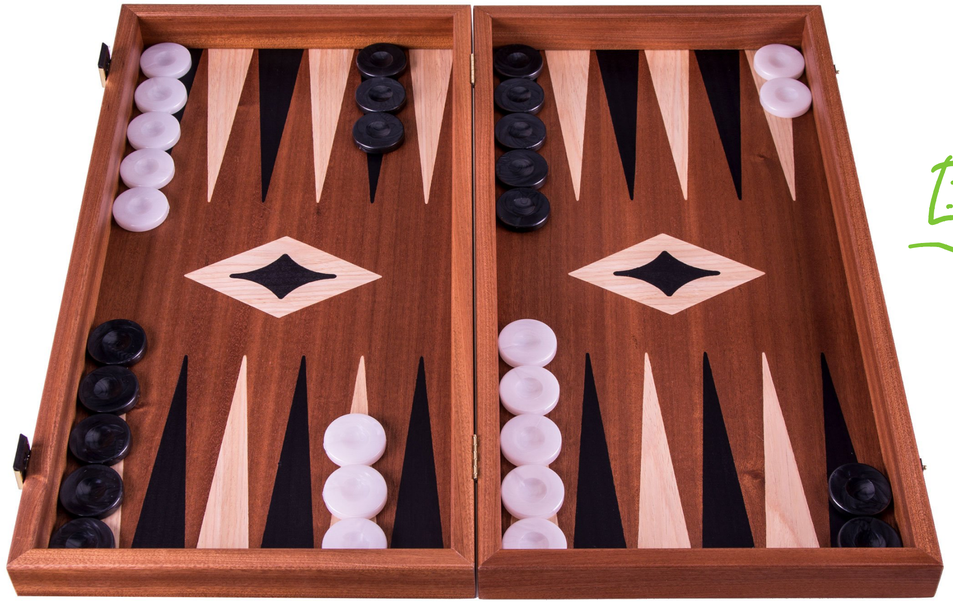
- Monte Carlo Search

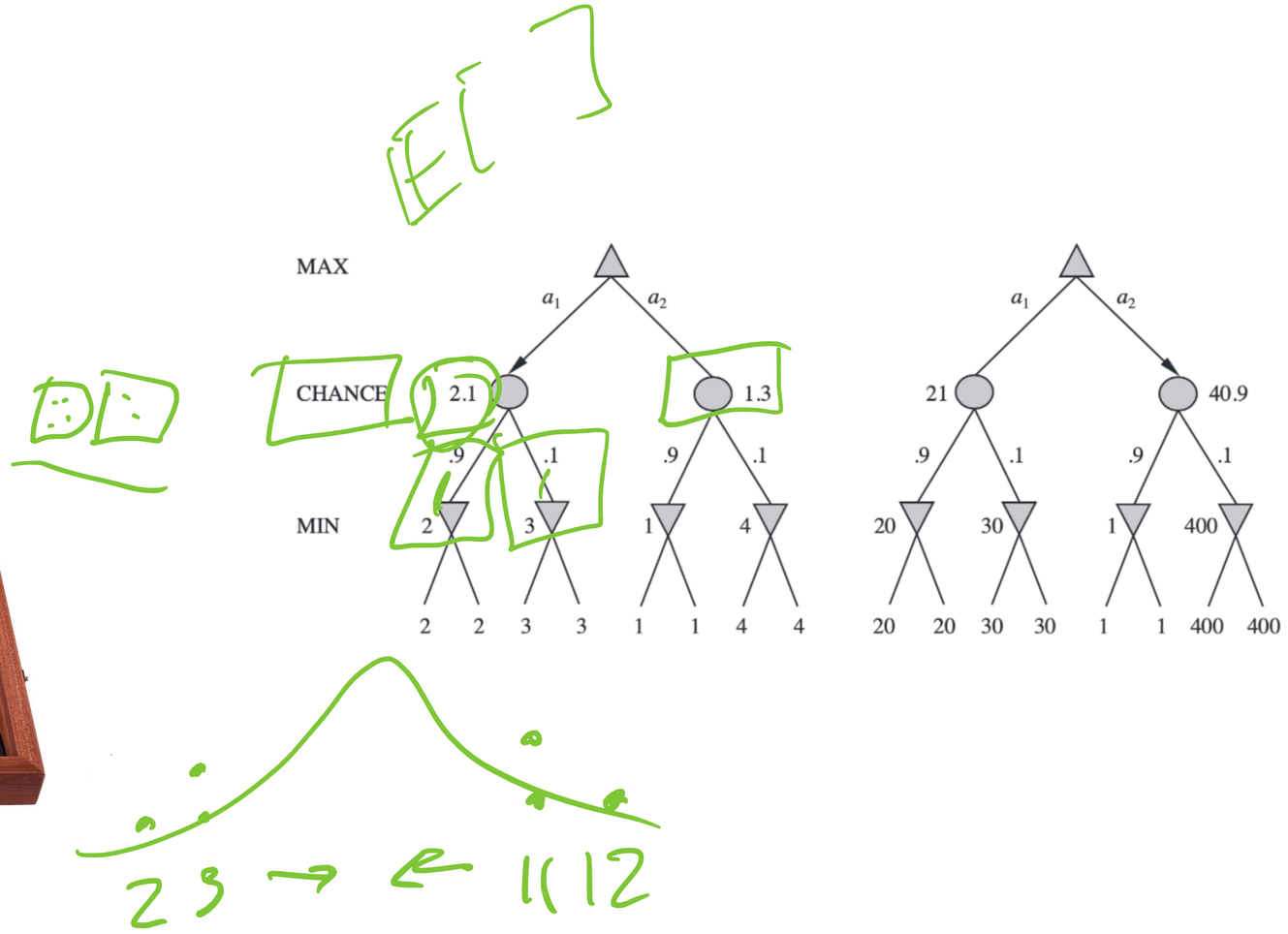
```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
    tree ← NODE(state)
    while IS-TIME-REMAINING() do
        leaf ← SELECT(tree)
        child ← EXPAND(leaf)
        result ← SIMULATE(child)
        BACK-PROPAGATE(result, child)
    return the move in ACTIONS(state) whose node has highest number of playouts
```

MinMax Algorithm

Possible Improvements

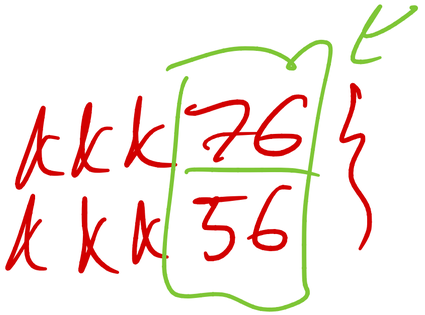
- Stochastic Games





MinMax Algorithm

Possible Improvements



- Stochastic Games

SHARE

RESEARCH ARTICLE

Superhuman AI for heads-up no-limit poker: Libratus beats top professionals

Noam Brown, Tuomas Sandholm*

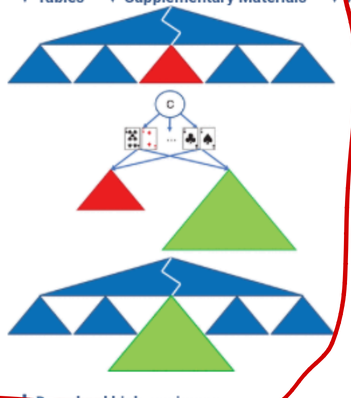
+ See all authors and affiliations

Science 26 Jan 2018:
Vol. 359, Issue 6374, pp. 418-424
DOI: 10.1126/science.aao1733

Article | Figures & Data | Info & Metrics | eLetters | PDF

Figures

▼ Tables ▼ Supplementary Materials ▼ Additional Files



Download high-res image

Open in new tab

Download Powerpoint

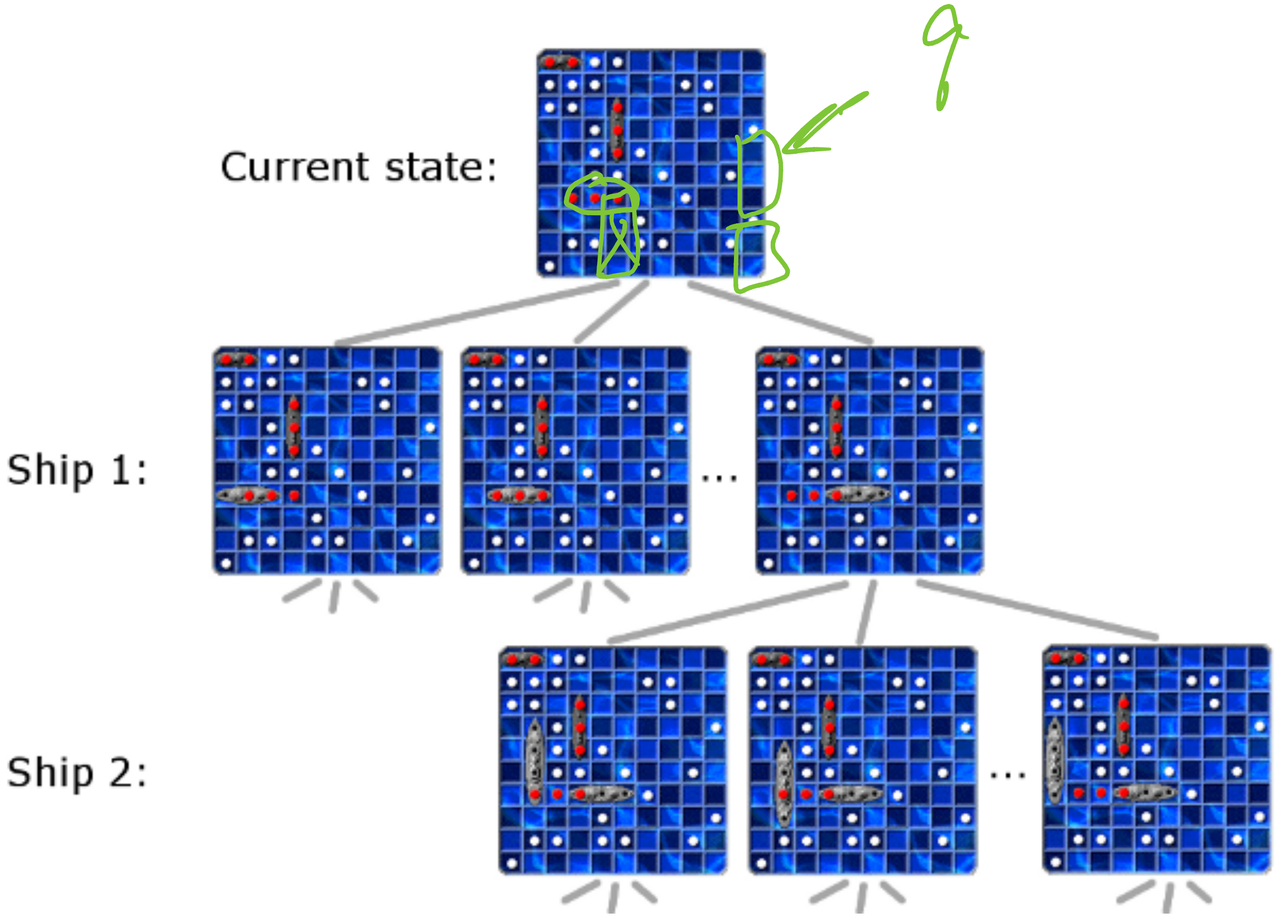
Fig. 1 Subgame solving.

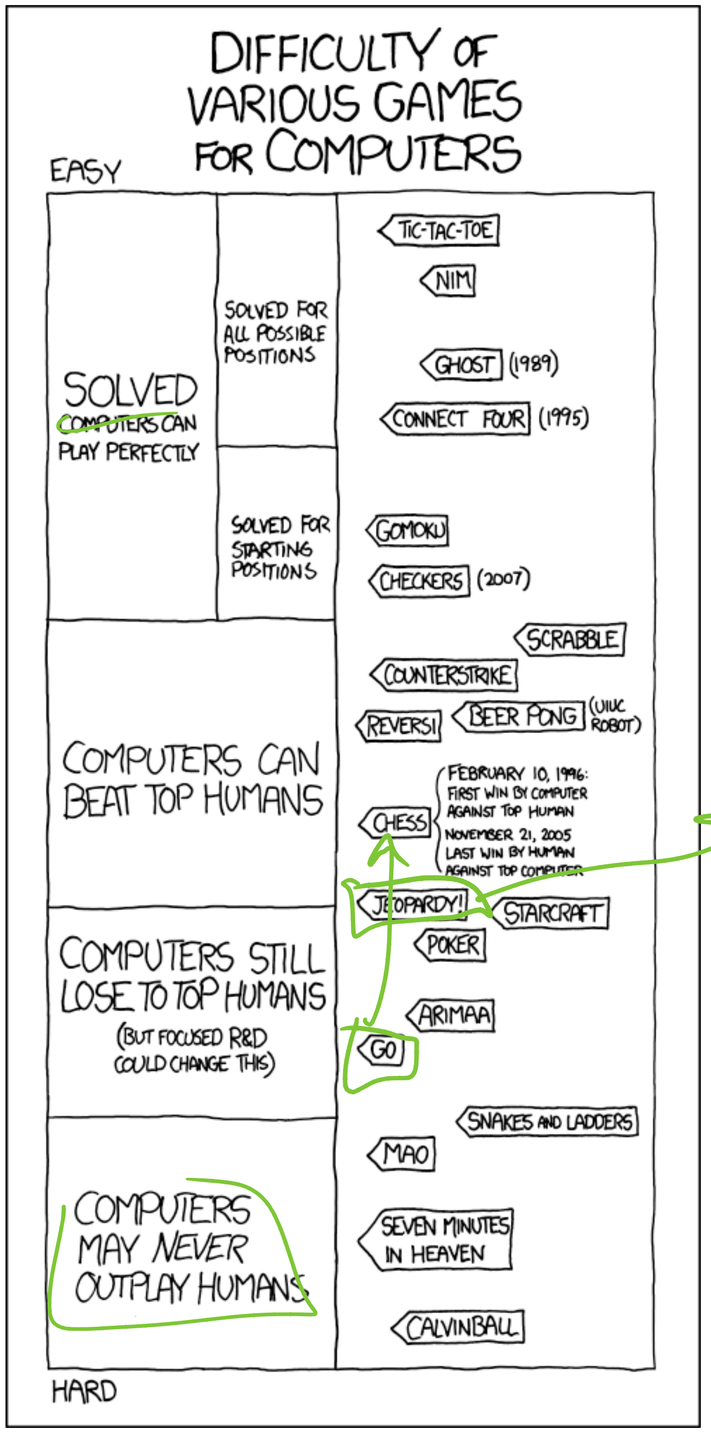
(**Top**) A subgame (red) is reached during play. Blue and red indicate the blueprint strategy. The white path indicates the action sequence before the reached subgame. (**Middle**) A more-detailed strategy for that subgame is determined by solving an augmented subgame in which, on each iteration, the opponent is dealt a random poker hand and given the choice of taking the expected value of the old abstraction (red) or of playing in the new, finer-grained abstraction (green), where the strategy for both players can change. This forces Libratus to make the finer-grained strategy at least as good as that in the original abstraction against every opponent poker hand. (**Bottom**) The new strategy is substituted in place of the old one.

MinMax Algorithm

Possible Improvements

- Partial Observability





State of the art

2012

IBM Watson