Game Ai- Project #2

Group Members

- Kamyar Manshaei
- Ali Mohamed Fatouh Ahmed
- Mojtaba Taghizadeh
- Omid Najaee Nezhad

Prof. Christian Bauckhage

BONN

TicTacToe Game Tree

➔ The ***tic tac toe*** game:

- Number of game states
    - $3^9 = 19683$
- Number of nodes in complete game tree
    - $9! = 362880$
- Some nodes are identical
- Different sequences of actions to achieve a specific game state

TicTacToe Game Tree

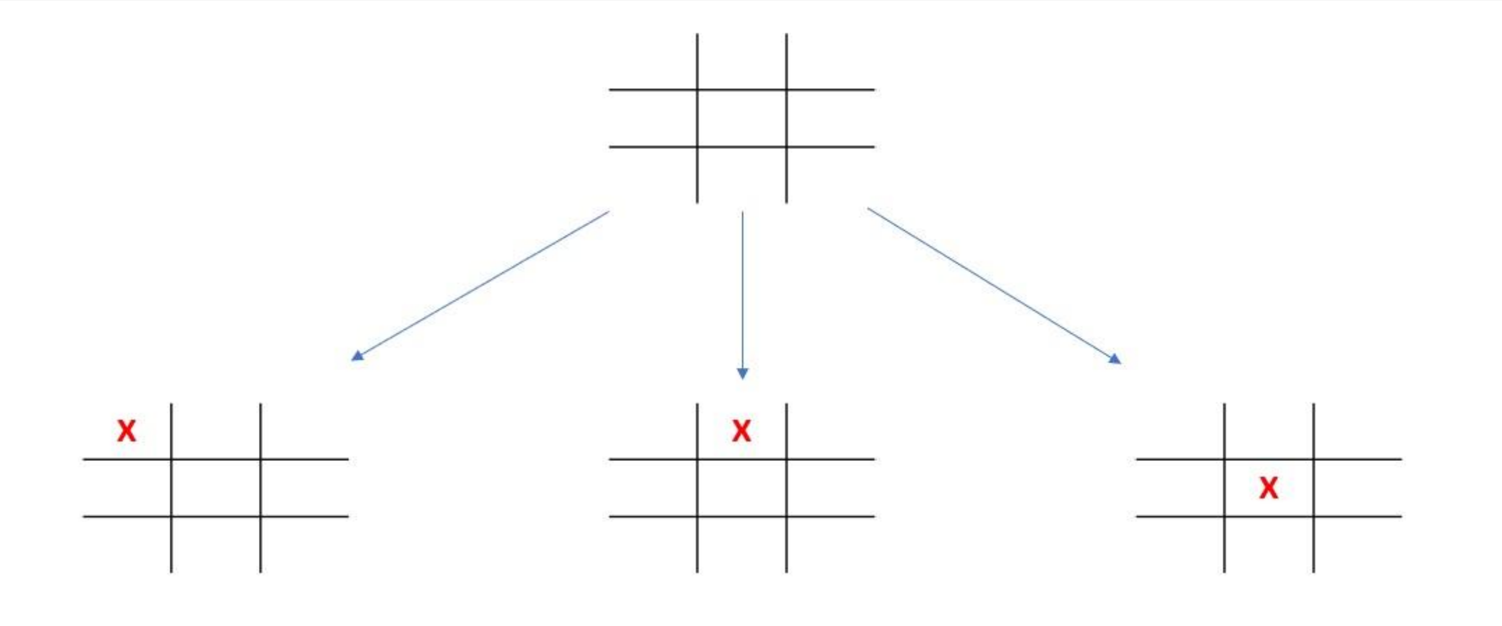
- ➔ Symmetry in ***tic tac toe*** game tree

- ➔ States below look similar:

- ➔ For each node remove:
  - ◆ States which are exactly the same
  - ◆ Rotation: 90°,180°,270°
  - ◆ Reflection across: X axis, Y axis, main diagonal, antidiagonal

TicTacToe Game Tree

➔ Children of the first node



TicTacToe Game Tree

➔ ***Tic tac toe*** game tree (Ignoring symmetric nodes)

- 765 nodes
- Player X wins 91 times
- Player O wins 44 times
- A draw situation occurs 3 times
- Average branching factor is 0.9986928104575163

➔ ***Tic tac toe*** game tree (Including symmetric nodes)

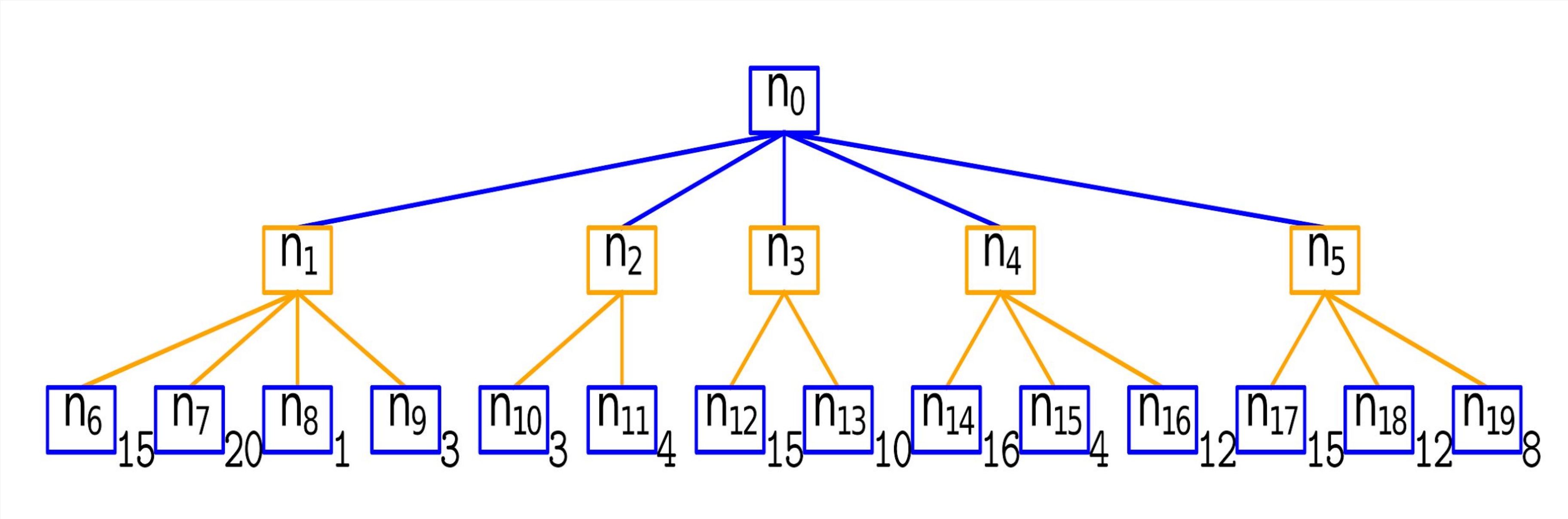
- 549946 nodes
- Player X wins 131184 times
- Player O wins 77904 times
- A draw situation occurs 46080 times
- Average branching factor is 0.9999981816396519

Minimax Computations

Minimax Algorithm:

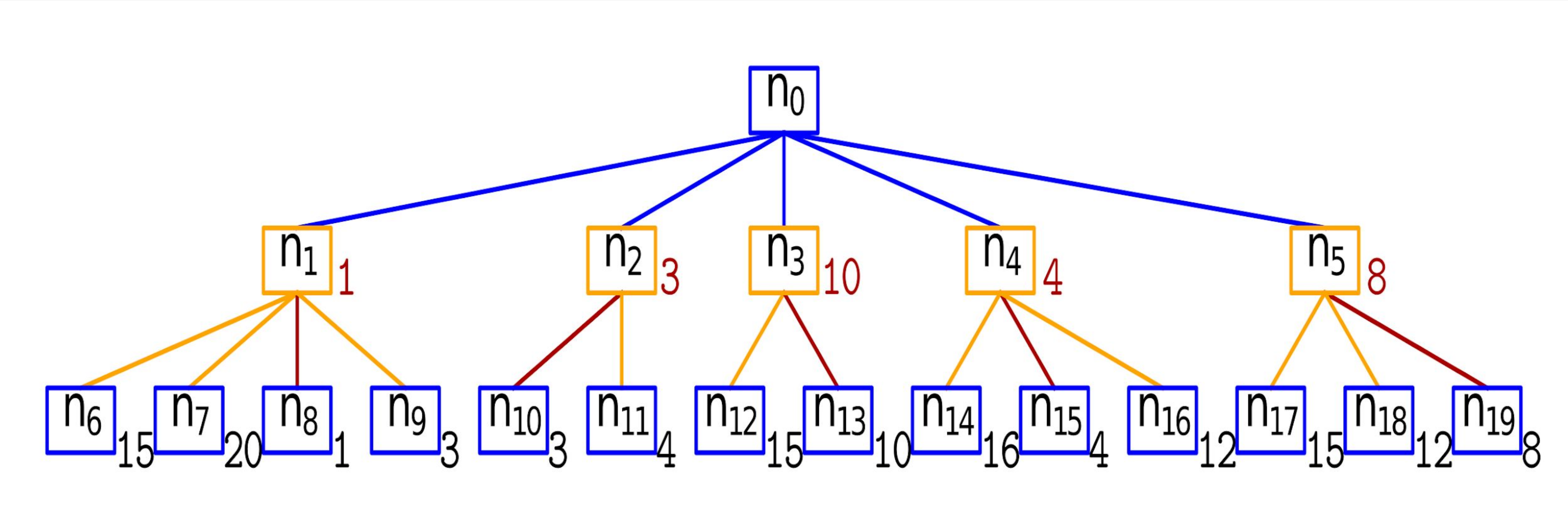
$$
mmv(n,p) = \begin{cases} u(n) & \text{if } n \text{ is a terminal node} \\ \max\limits_{s \in Succ(n)} mmv(n, \mathrm{MIN}) & \text{if } p \text{ is MAX} \\ \min\limits_{s \in Succ(n)} mmv(n, \mathrm{MAX}) & \text{if } p \text{ is MIN} \end{cases}
$$

Minimax Computations



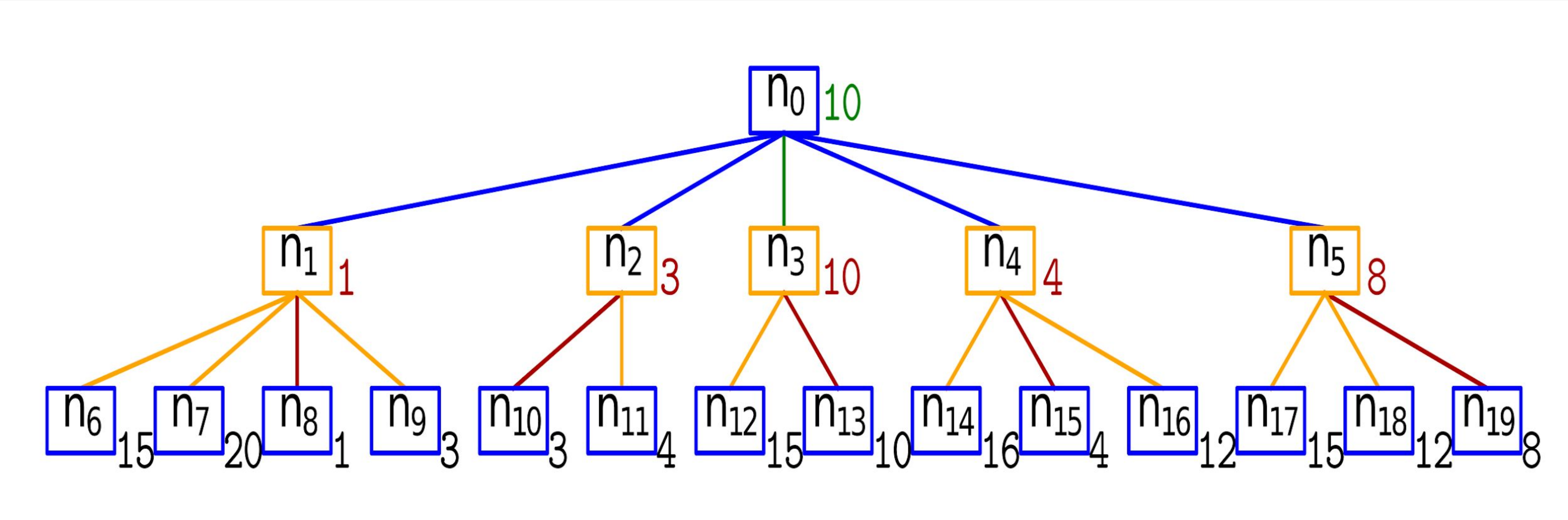
Minimax Evaluation (utility value) at leaf nodes

Minimax Computations



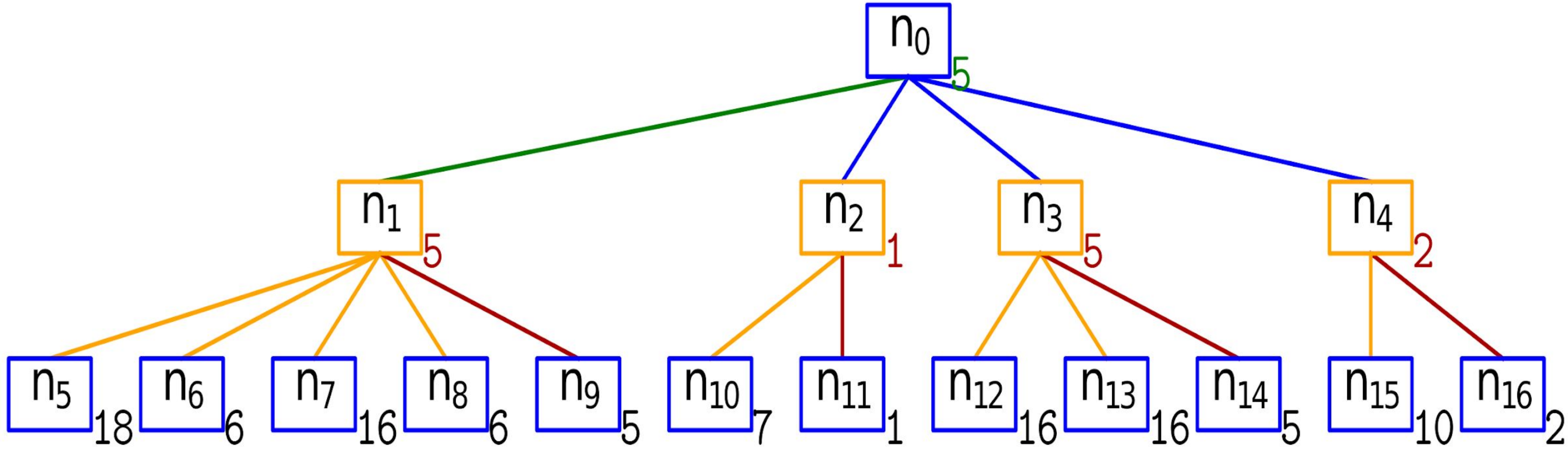
Minimax Evaluation at level 1 nodes (n1, n2, n3, n4 and n5)

Minimax Computations



Minimax Evaluation at root node

Minimax Computations: Better Alternatives



Minimax Evaluation: Naive Minimax algorithm shows the evaluation in optimal game playing but does not provide a solution for better alternatives in case of ties

Minimax Computations: Better Alternatives

Better Alternatives Algorithm:

- ➔ Run minimax algorithm
  - ◆ Mark Nodes with highest value.
- ➔ If len(Nodes) > 1
  - ◆ Run maximax algorithm on Nodes (nodes with highest value) only.
  - ◆ Mark Nodes with highest value.
- ➔ If len(Nodes) > 1
  - ◆ Calculate average of the utilities for every node in Nodes.
- ➔ Go to node with the highest value.

BONN

Minimax Computations: Better Alternatives

Maximax Algorithm:

➔ Finding the maximum payoff in the tree

$$maxi(n) = \begin{cases} u(n) & \text{if } n \text{ is a terminal node} \\ \max_{s \in Succ(n)} maxi(n) & \text{otherwise} \end{cases}$$

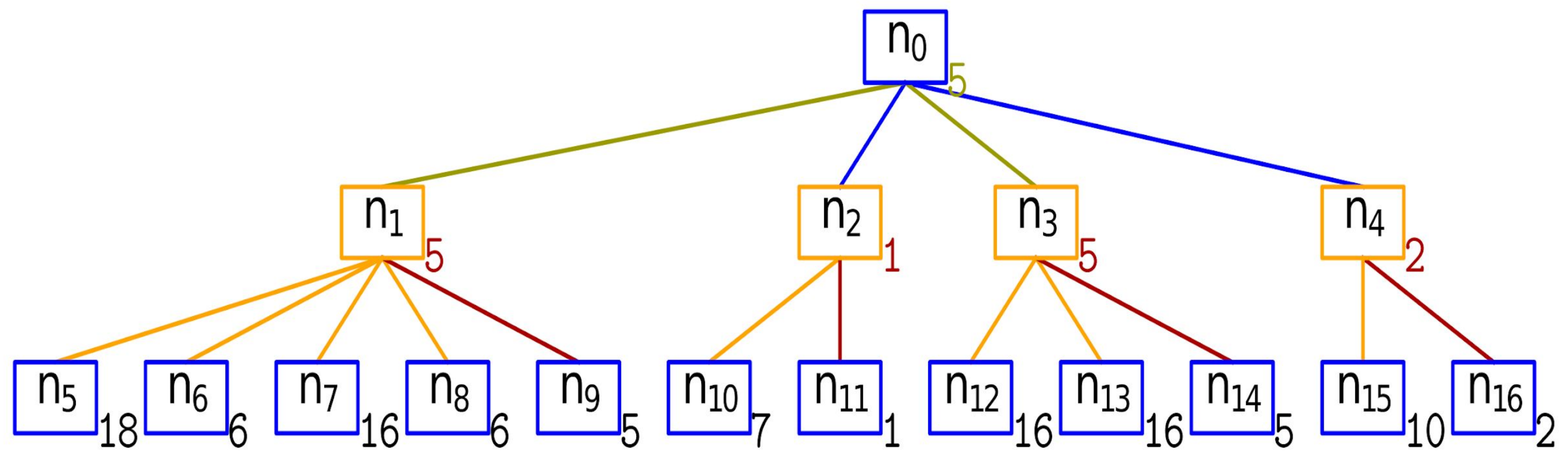
Minimax Computations: Better Alternatives

Average Algorithm:

➔ Finding the average of utilities for every node to give the most opportunistic node

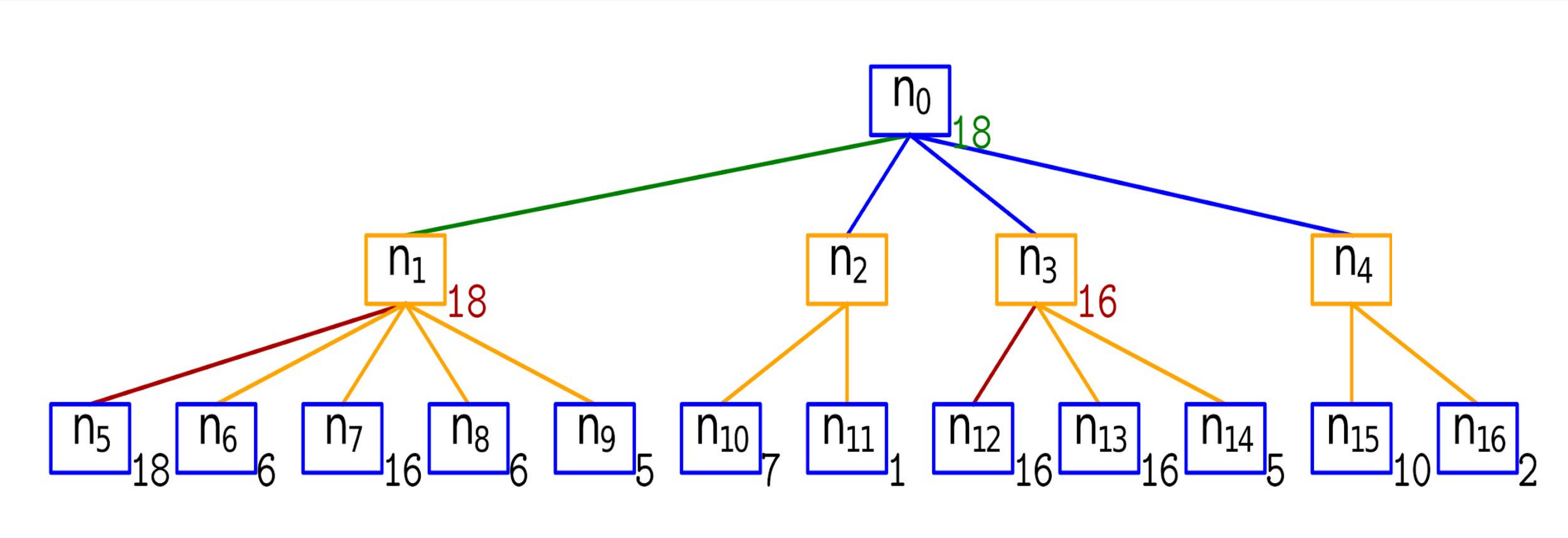
$$avgi(n) = \begin{cases} u(n) & \text{if } n \text{ is a terminal node} \\ \underset{s \in Succ(n)}{avg} \; avgi(n) & \text{otherwise} \end{cases}$$

Minimax Computations: Better Alternatives



Minimax Evaluation - Marking nodes with highest values

Minimax Computations: Better Alternatives



Maximax Evaluation

Minimax Computations: Better Alternatives

Would we have to do this in practice:

✗ Playing optimally

✓ Playing non-optimally (Human opponent)

Minimax search for Connect Four

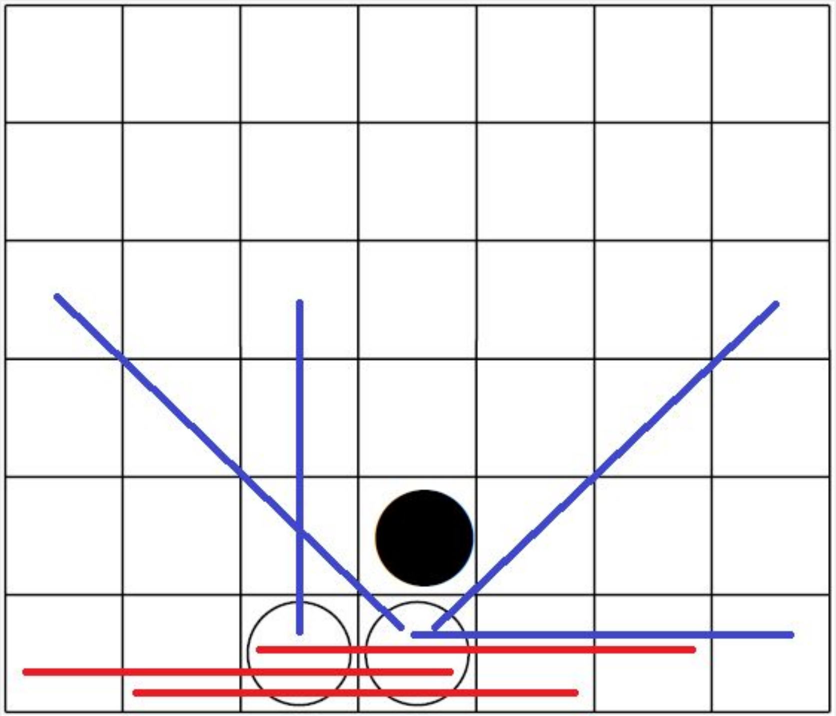
First strategy:

Calculating a score based on the following heuristic:

A 'possible winning line of magnitude i' to be i discs of one player in a row/column/diagonal, where all other positions in this row/column/diagonal are empty.
(The discs do not need to be in one contiguous block)

Minimax search for Connect Four

Example:



Minimax search for Connect Four

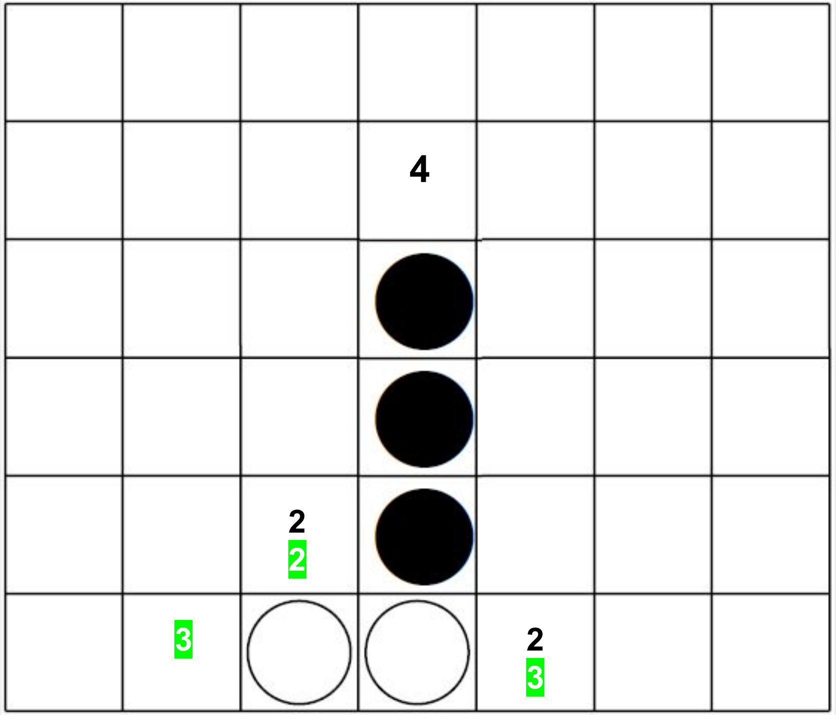
Second strategy:

Calculating a score based on the following heuristic:

A score for empty cells based on the filled cells around it.
In each move player first, checks, if the opponent has a cell with a score of more than three which means he can win in the next move he places his token in this cell. If not he places in the cell with the most score.

Minimax search for Connect Four

Example:



Minimax search for Connect Four

Statistics:

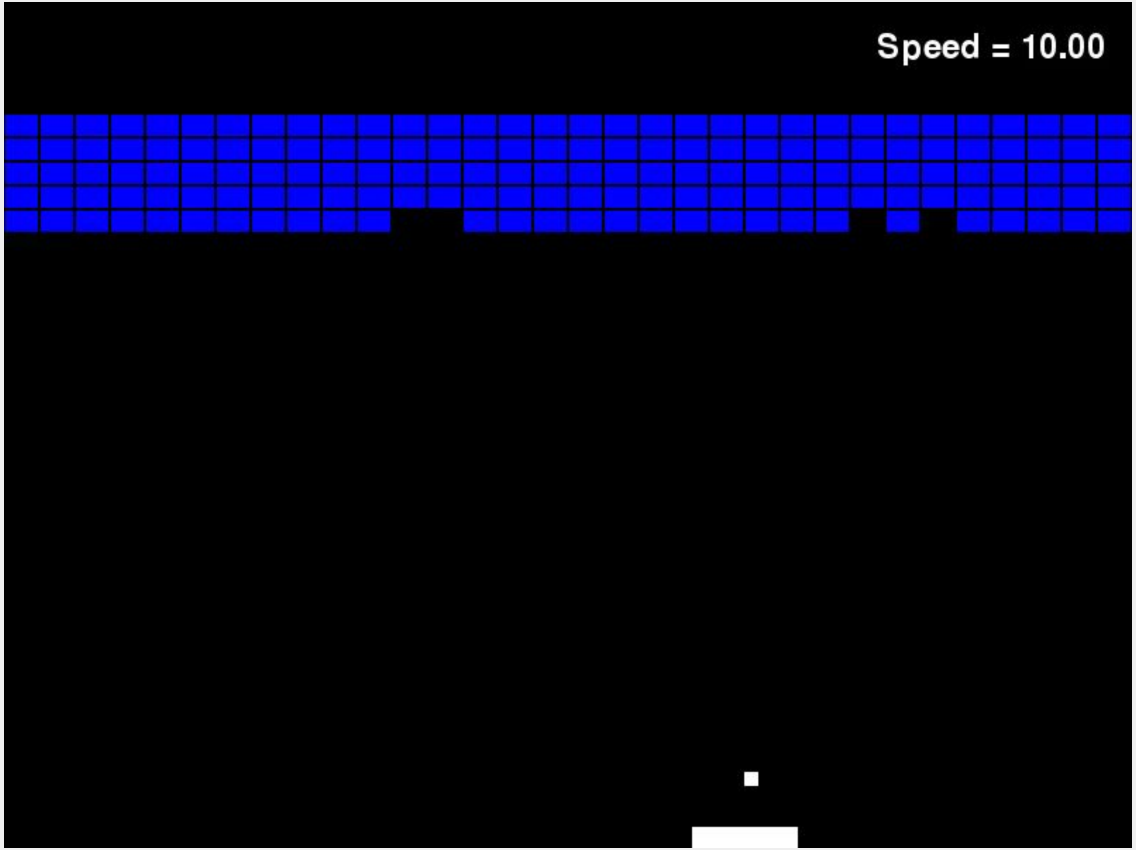
First strategy heuristic wins against random player:

997 / 1000 times (depth = 1)
1000 / 1000 times (depth $\geq$ 2)

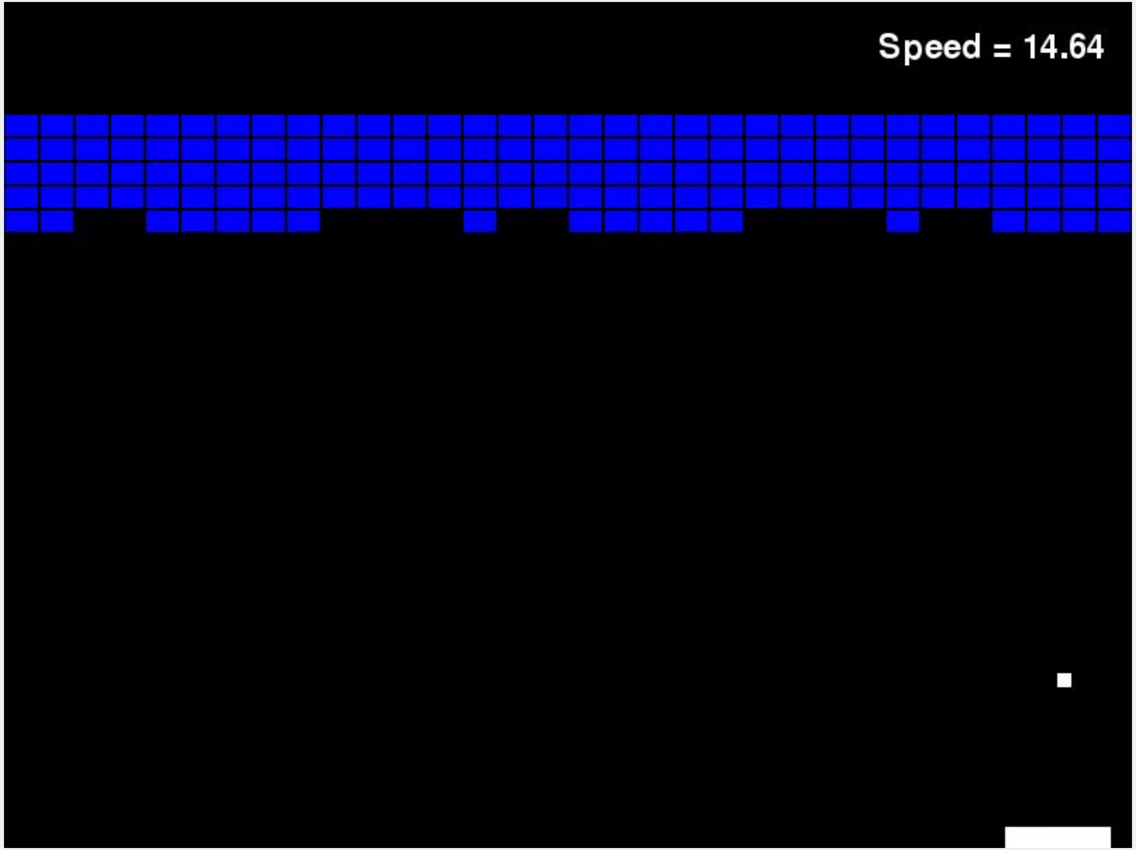
Second strategy heuristic wins against random player:

887 / 1000 times

Breakout



Breakout: Adding acceleration



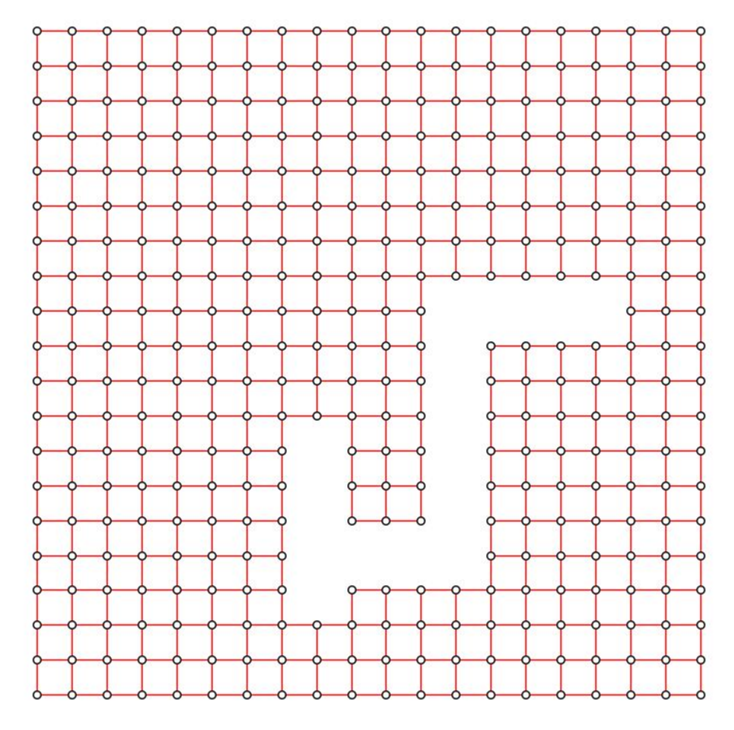
Path planning

- A matrix of a 2D game base with some obstacles in between are represented by zeros and ones
- The aim
  - Model the data using a graph
  - Find the shortest-path with between two points
    - Dijkstra Algorithm
    - A* Algorithm

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Path planning

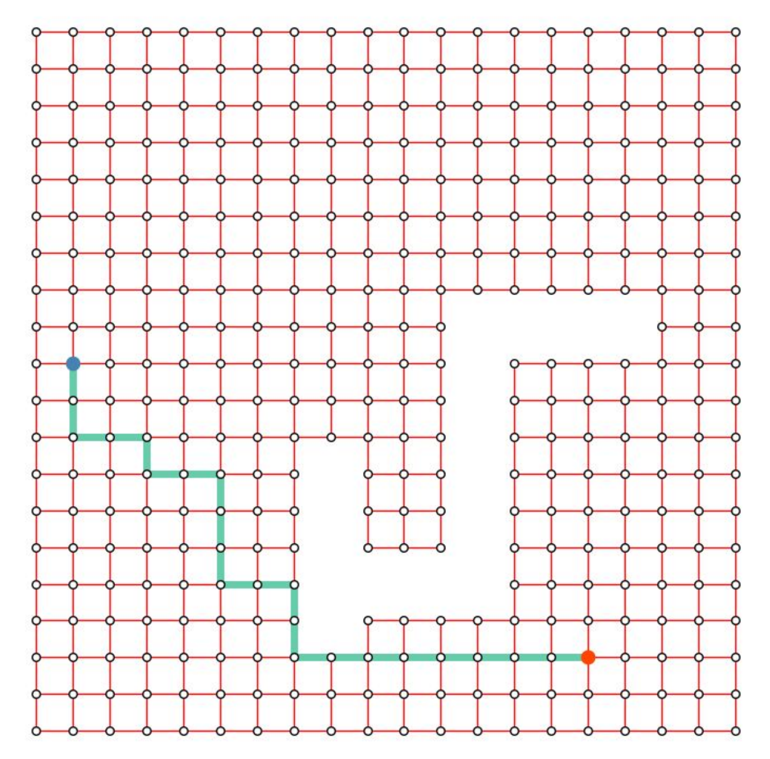
- Used *networkX* API to model the matrix with a graph.
- Used *matplotlib* to visualize the findings.



BONN

Path planning

- Dijkstra Algorithm
- Computational complexity
    - min-heap and graph structure O(log(V)*E)
    - A 2D matrix O(V^2)



Thank you for your attention.

Questions?