

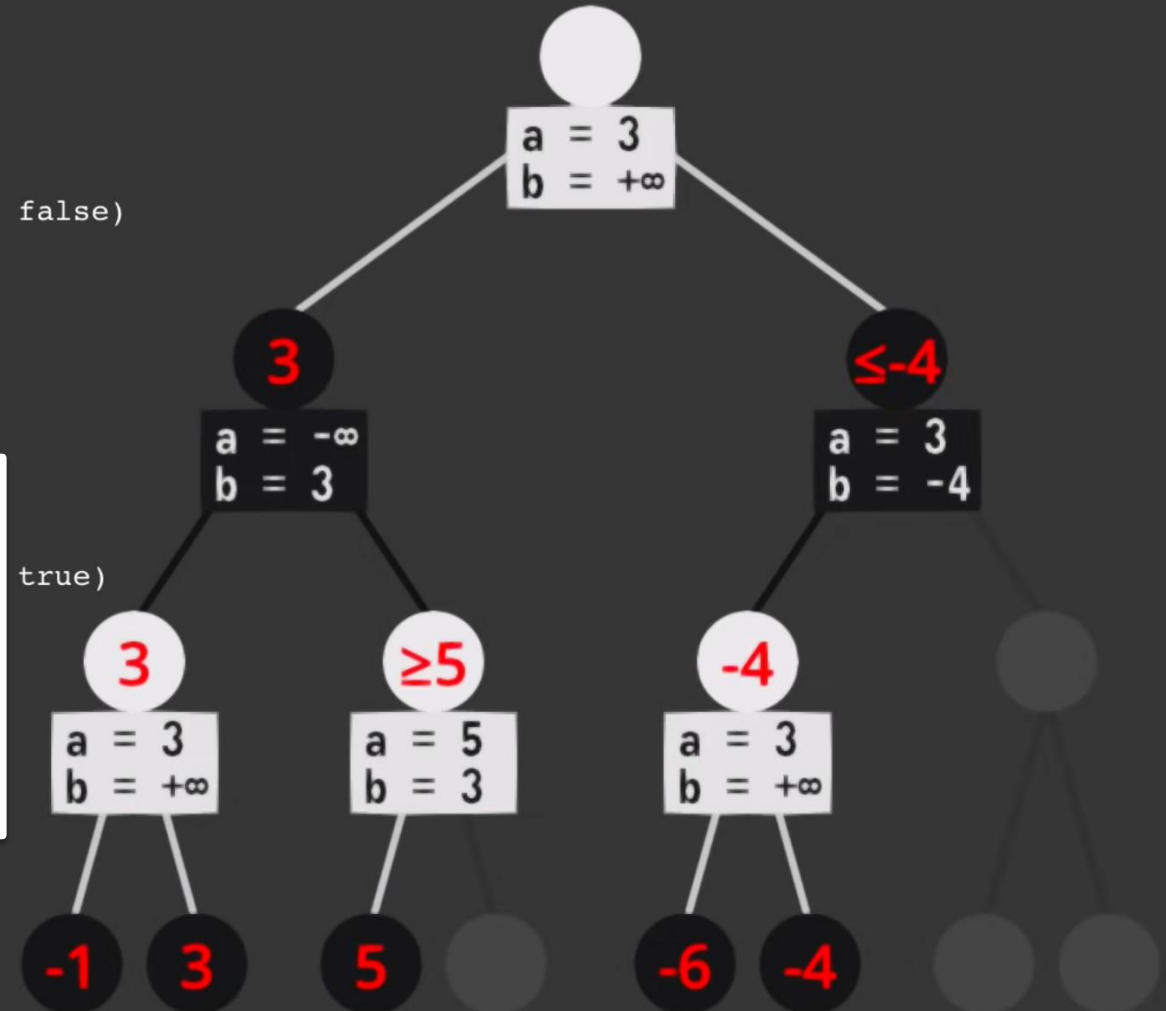
```
function minimax(position, depth, alpha, beta, maximizingPlayer)
    if depth == 0 or game over in position
        return static evaluation of position

    if maximizingPlayer
        maxEval = -infinity
        for each child of position
            eval = minimax(child, depth - 1, alpha, beta, false)
            maxEval = max(maxEval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha
                break
        return maxEval
```

LÖS WS22

Visualisierung der Algorithmen: alpha-beta (Web)

```
// initial call
minimax(currentPosition, 3, -∞, +∞, true)
```



Inhalt

- Motivation
- Inspiration
- Technologien
- Algorithmus
- Visualisierung
- Live Demo

Motivation

- TicTacToe mit entsprechendem Algorithmus bauen
 - MinMax & alpha-beta-pruning besser verstehen
- Ansprechende Visualisierung
- Spezial TicTacToe

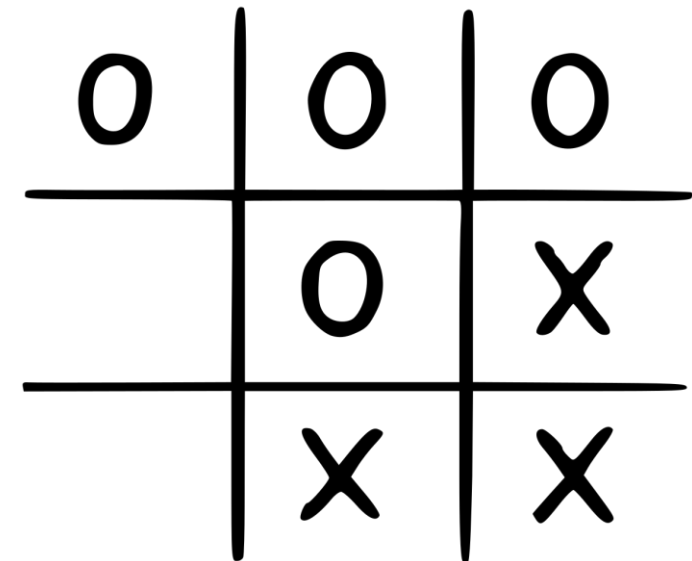


Abb. 1: TicTacToe, Quelle: <https://de.wikipedia.org/wiki/Tic-Tac-Toe>

Inspiration

- <https://minimax-visualizer.herokuapp.com/>

Sai Sunku

[GitHub](#) | [LinkedIn](#) | [Blog](#)

Tic Tac Toe Minimax Visualizer

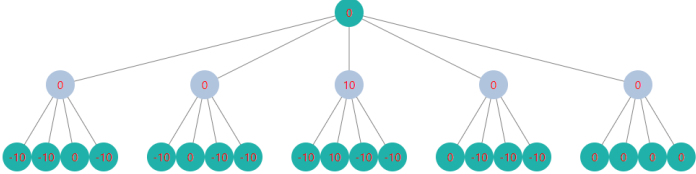
Backend: Python, Django, WebSockets, pytest
Frontend: HTML/CSS, JavaScript, d3.js

Player X
 Player O

O	X	
	X	
	O	

Current player X

Minimax tree (scroll to zoom, hover over node to see board position)



Tree Depth

The Minimax algorithm calculates all possible board positions into the future and picks the optimal move. This site provides a visualization of the algorithm.
 The root of the tree is the current position and each node is a possible future board position. Each node is assigned a score assuming optimal play from both players.
 When playing against the Minimax AI, a score of 0 means that the game is bound to end in a draw. A score of +10 is a guaranteed win for the AI and a score of -10 is a guaranteed loss.

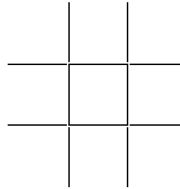


Abb. 2: Minimax Visualizer,
 Quelle: <https://minimax-visualizer.herokuapp.com/>

Verwendete Technologien

- Basierend auf Vue.js
- Out-of-the-Box Support für Browser
- Performance fokussiert



QUASAR

BEYOND THE FRAMEWORK

Abb. 3: Quasar Framework, Quelle:
<https://cdn.quasar.dev/logo-v2/svg/logo-vertical.svg>

Verwendete Technologien - Vorteile

Reaktivität

Viele nützliche
Funktionalitäten

Eigene
Komponenten

Verwendete Technologien – Tree-Element

- ✓ Darstellung von Entscheidungsbaum
- ✓ Unkonventioneller Weg
- ✓ Klassisches Konzept verlassen
- ✓ Akkordeon

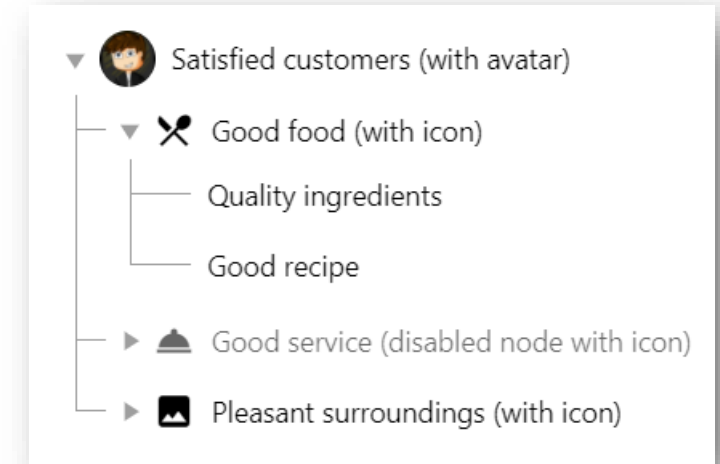


Abb. 4: QTree, Quelle: <https://quasar.dev/vue-components/tree>

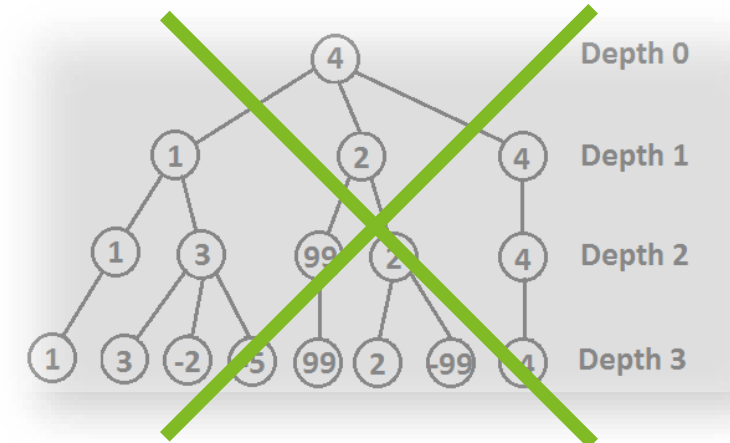
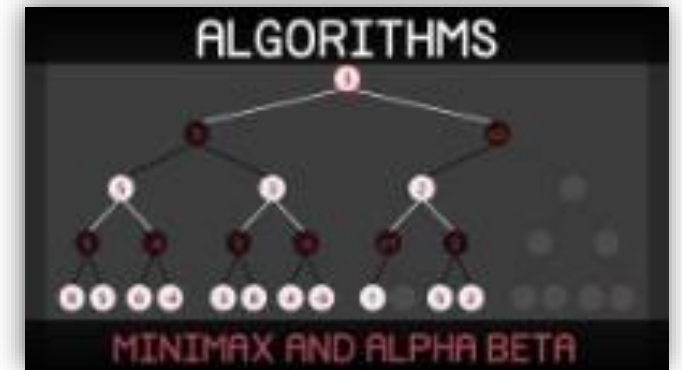


Abb. 5: MinMax-Tree, Quelle: <https://ds055uzetaobb.cloudfront.net/brioche/uploads/LbUstMINM1-minimaxtree.png?width=1200>

Algorithmus - Recherche

Wenig Vorerfahrung, daher...

- Wie arbeitet der Algorithmus?
- Wie lässt sich TicTacToe damit verbinden?
- Gibt es andere Versionen des Spiels?



Quelle: <https://youtu.be/l-hh51ncgDI>



Quelle: <https://youtu.be/trKjYdBASyQ>



Quelle: <https://youtu.be/ft3YWCKvuQE>

Algorithmus – Code-Basis

- Alpha/Beta
- Pruning
- Einstellbare Suchtiefe

```
if (isMaximizing) {  
    let bestScore = -Infinity;  
    for (let i = 0; i < 3; i++) {  
        for (let j = 0; j < 3; j++) {  
            // Is the spot available?  
            if (board[i][j] == '' && typeof depth === 'number') {  
                board[i][j] = ai;  
                // eslint-disable-next-line @typescript-eslint/no-unsafe-assignment  
                let score = minimax(board, depth - 1, false, alpha, beta);  
                board[i][j] = '';  
                bestScore = Math.max(score, bestScore);  
                // check for alpha  
                alpha = Math.max(alpha, bestScore);  
                // Check for alpha beta pruning  
                if (beta <= alpha) {  
                    break;  
                }  
            }  
        }  
    }  
    return bestScore;  
}
```

Abb. 6: Code-Ausschnitt Alpha/Beta, Quelle: eigene Darstellung

Algorithmus - Recycling

- Identisch bei Special-TTT
- Lediglich Schleife angepasst



universell einsetzbar

```
} else {  
  let bestScore = 1000;  
  for (let i = 0; i < 10; i++) {  
    for (let j = 0; j < 5; j++) {  
      // Is the spot available?  
      if (board[i][j] == '' && typeof depth === 'number') {  
        board[i][j] = human;  
        // eslint-disable-next-line @typescript-eslint/no-unsafe-assignment  
        let score = minimax(board, depth - 1, true, alpha, beta);  
        board[i][j] = '';  
        bestScore = Math.min(score, bestScore);  
        // check for beta  
        beta = Math.min(beta, bestScore);  
        // Check for alpha beta pruning  
        if (beta <= alpha) {  
          break;  
        }  
      }  
    }  
  }  
  return bestScore;  
}
```

Abb. 7: Code-Ausschnitt Alpha/Beta, Quelle: eigene Darstellung

Visualisierung

■ Hervorhebung des gewählten Boards

```
<div
  :class="'board' + index"
  v-for="(entry, index) in boardStatesComp.value"
  :key="index"
>
  <view-board
    :current-board="entry.state"
    :id="index"
    :score="entry.score"
    :class="checkChosenMove(entry.state) ? 'trueStyle' : ''"
  />
</div>
```

Abb. 8: Code-Ausschnitt Template, Quelle: eigene Darstellung

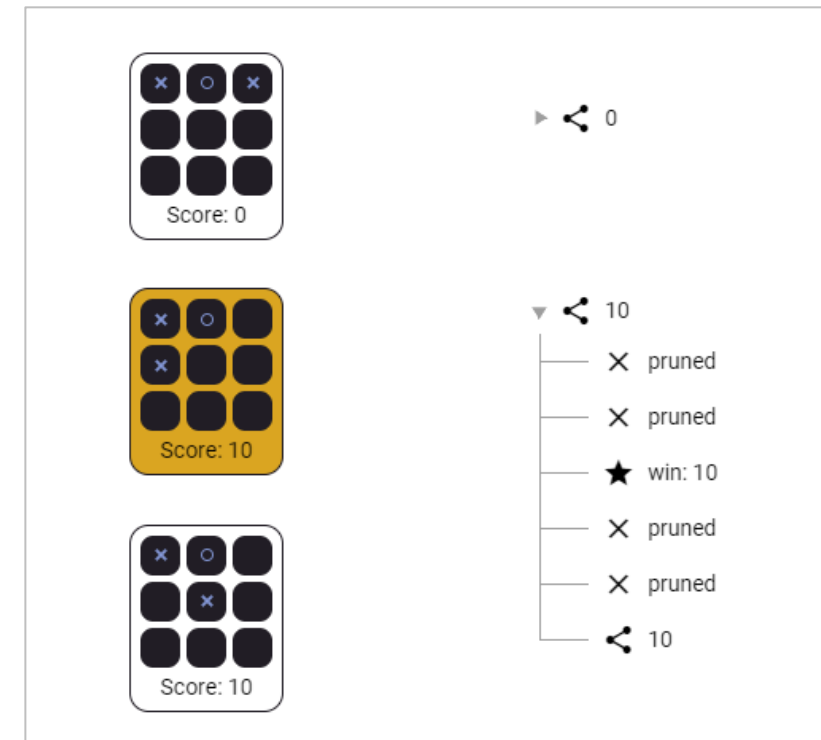


Abb. 9: Browser Ansicht, Quelle: eigene Darstellung

Visualisierung

- Tree Element
 - Komplizierter Aufbau für unsere Zwecke
 - Icons
 - Disabled Modus

```
interface tree {
  label: string;
  key: string;
  tree: [
    {
      label: string;
      key: number;
      icon: string;
      disabled: boolean;
      children?: [
        {
          label: string;
          key: number;
          icon: string;
          disabled?: boolean;
          children?: treeChild[];
        }
      ];
    }
  ];
}
```

```
interface treeChild {
  label: string;
  key: number;
  children?: treeChild[];
}
```

Abb. 10: Code-Ausschnitt Interfaces, Quelle: eigene Darstellung

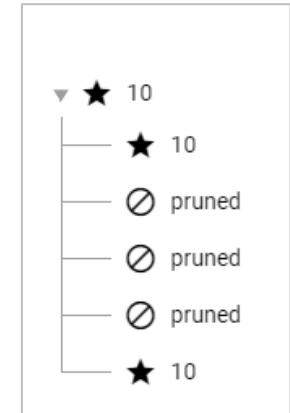


Abb. 11: Browser Ansicht,
Quelle: eigene Darstellung

Visualisierung

- Animationen
 - Anfangs geplant
 - Nur bedingt umgesetzt

```
@keyframes myAnim {  
  0% {  
    transform: scale(1);  
    opacity: 0;  
    visibility: visible;  
  }  
  
  50% {  
    transform: scale(1.1);  
    opacity: 0.5;  
    visibility: visible;  
  }  
  
  100% {  
    transform: scale(1);  
    opacity: 1;  
    display: block;  
    visibility: visible;  
  }  
}
```

Abb. 12: Code-Ausschnitt Animation, Quelle: eigene Darstellung

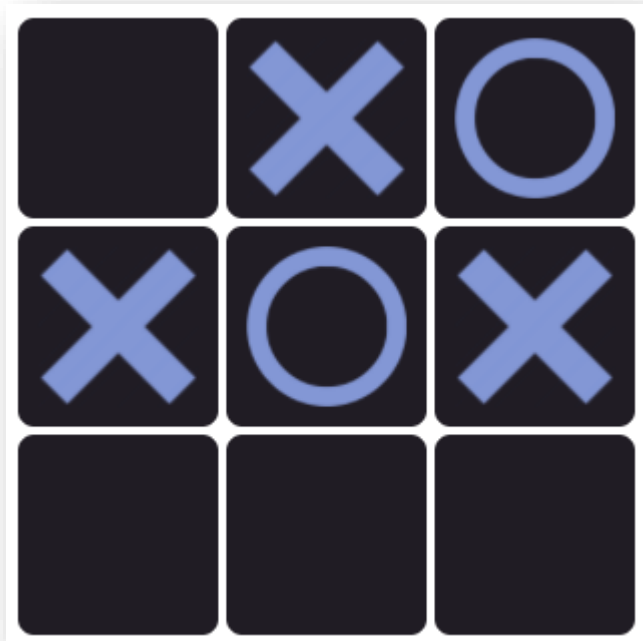


Abb. 13: TicTacToe, Quelle: eigene Darstellung

Live Demo