Schach Use Case für ChatGPT:
Schach Spiel in ChatGPT selber programmieren

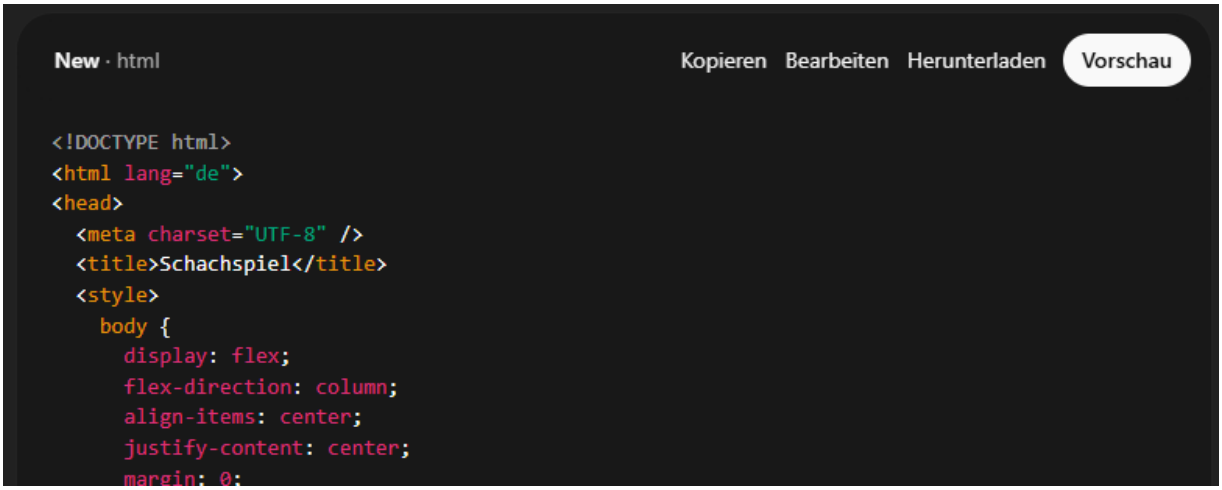
Einleitung:

Anleitung

- Öffne ChatGPT und sag: „Canvas: Erstelle eine neue HTML-Datei.“
- Füge den kompletten Code in die Datei ein.
- Speichern – die Canvas zeigt dir sofort die Vorschau.
- Klicken, um zu spielen: Figur wählen → Zielfeld klicken.

Code kopieren:

Den Code findest du hier:



```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8" />
  <title>Schachspiel</title>
  <style>
    body {
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      margin: 0;
      padding: 0;
      font-family: sans-serif;
    }

    h1 {
      margin-top: 1rem;
      margin-bottom: 0.5rem;
    }

    #status {
      font-size: 1.2rem;
      margin-bottom: 1rem;
      padding: 0.5rem 1rem;
      border-radius: 8px;
      background-color: #f2f2f2;
    }

    #board {
      display: grid;
      grid-template-columns: repeat(8, 60px);
      grid-template-rows: repeat(8, 60px);
      gap: 0;
      border: 2px solid #333;
    }

    .cell {
      width: 60px;
      height: 60px;
      display: flex;
      align-items: center;
      justify-content: center;
      font-size: 3rem;
      cursor: pointer;
      user-select: none;
    }

    /* Hintergrundfarben, aber keine Font-Farbe: */
    .black-cell {
      background-color: #b58863;
    }

    .white-cell {
```

```
      background-color: #f2d9b1;
    }

    .highlight {
      outline: 3px solid rgba(255, 255, 0, 0.7);
      outline-offset: -3px;
    }

    .turn-indicator {
      font-weight: bold;
      color: #e60000;
    }

    #container {
      margin: 1rem;
      text-align: center;
    }

  </style>
</head>
<body>
  <h1>Schachspiel für 2 Spieler</h1>
  <div id="status"></div>
  <div id="board"></div>

  <script>
    // Unicode-Bauern (Weiß ♙, Schwarz ♟), Türme (Weiß ♖, Schwarz ♜),
    // Springer (Weiß ♘, Schwarz ♞), Läufer (Weiß ♗, Schwarz ♝),
    // Dame (Weiß ♕, Schwarz ♛), König (Weiß ♔, Schwarz ♚)

    let boardState = [];
    let whiteToMove = true; // true = Weiß am Zug, false = Schwarz am Zug
    let selectedCell = null; // zuletzt angeklickte Zelle
    let possibleMoves = [];  // Liste von Zellen (row,col), auf die gezogen
werden kann
    let gameOver = false;

    // Unicode-Mapping für die Figuren
    const pieceToUnicode = {
      'Pw': '♙', // Weißer Bauer
      'Pb': '♟', // Schwarzer Bauer
      'Rw': '♖', // Weißer Turm
      'Rb': '♜', // Schwarzer Turm
      'Nw': '♘', // Weißer Springer
      'Nb': '♞', // Schwarzer Springer
      'Bw': '♗', // Weißer Läufer
      'Bb': '♝', // Schwarzer Läufer
      'Qw': '♕', // Weiße Dame
      'Qb': '♛', // Schwarze Dame
      'Kw': '♔', // Weißer König
      'Kb': '♚'  // Schwarzer König
    };

    // Standard-Setup
```

```
    function initBoard() {
      const initial = [
        [{type:'R', color:'b'},{type:'N', color:'b'},{type:'B',
color:'b'},{type:'Q', color:'b'},{type:'K', color:'b'},{type:'B',
color:'b'},{type:'N', color:'b'},{type:'R', color:'b'}],
        [{type:'P', color:'b'},{type:'P', color:'b'},{type:'P',
color:'b'},{type:'P', color:'b'},{type:'P', color:'b'},{type:'P',
color:'b'},{type:'P', color:'b'},{type:'P', color:'b'}],
        [null,null,null,null,null,null,null,null],
        [null,null,null,null,null,null,null,null],
        [null,null,null,null,null,null,null,null],
        [null,null,null,null,null,null,null,null],
        [{type:'P', color:'w'},{type:'P', color:'w'},{type:'P',
color:'w'},{type:'P', color:'w'},{type:'P', color:'w'},{type:'P',
color:'w'},{type:'P', color:'w'},{type:'P', color:'w'}],
        [{type:'R', color:'w'},{type:'N', color:'w'},{type:'B',
color:'w'},{type:'Q', color:'w'},{type:'K', color:'w'},{type:'B',
color:'w'},{type:'N', color:'w'},{type:'R', color:'w'}]
      ];
      boardState = initial;
    }

    function onBoard(r, c){
      return (r >= 0 && r < 8 && c >= 0 && c < 8);
    }

    function generateMoves(row, col, board = boardState) {
      const piece = board[row][col];
      if(!piece) return [];
      const {type, color} = piece;
      const directions = [];
      const moves = [];

      function sameColor(r, c){
        return board[r][c] && board[r][c].color === color;
      }

      switch(type){
        case 'P': // Bauer
          if(color==='w') {
            if(onBoard(row-1, col) && !board[row-1][col]) {
              moves.push({r: row-1, c: col});
              if(row===6 && !board[row-2][col]) {
                moves.push({r: row-2, c: col});
              }
            }
            if(onBoard(row-1, col-1) && board[row-1][col-1] &&
board[row-1][col-1].color==='b') {
              moves.push({r: row-1, c: col-1});
            }
            if(onBoard(row-1, col+1) && board[row-1][col+1] &&
board[row-1][col+1].color==='b') {
              moves.push({r: row-1, c: col+1});
            }
```

```
          } else {
            if(onBoard(row+1, col) && !board[row+1][col]) {
              moves.push({r: row+1, c: col});
              if(row===1 && !board[row+2][col]) {
                moves.push({r: row+2, c: col});
              }
            }
            if(onBoard(row+1, col-1) && board[row+1][col-1] &&
board[row+1][col-1].color==='w') {
              moves.push({r: row+1, c: col-1});
            }
            if(onBoard(row+1, col+1) && board[row+1][col+1] &&
board[row+1][col+1].color==='w') {
              moves.push({r: row+1, c: col+1});
            }
          }
          break;
        case 'R': // Turm
          directions.push({dr:1, dc:0}, {dr:-1, dc:0}, {dr:0, dc:1}, {dr:0,
dc:-1});
          directions.forEach(dir => {
            let r = row + dir.dr;
            let c = col + dir.dc;
            while(onBoard(r, c)) {
              if(!board[r][c]) {
                moves.push({r, c});
              } else {
                if(!sameColor(r, c)) {
                  moves.push({r, c});
                }
                break;
              }
              r += dir.dr;
              c += dir.dc;
            }
          });
          break;
        case 'N': // Springer
          const knightMoves = [
            {r: row+2, c: col+1},
            {r: row+2, c: col-1},
            {r: row-2, c: col+1},
            {r: row-2, c: col-1},
            {r: row+1, c: col+2},
            {r: row+1, c: col-2},
            {r: row-1, c: col+2},
            {r: row-1, c: col-2}
          ];
          knightMoves.forEach(m => {
            if(onBoard(m.r, m.c) && (!board[m.r][m.c] || !sameColor(m.r, m.c)))
{
              moves.push({r: m.r, c: m.c});
            }
          });
```

```
          break;
        case 'B': // Läufer
          directions.push({dr:1, dc:1}, {dr:1, dc:-1}, {dr:-1, dc:1}, {dr:-1,
dc:-1});
          directions.forEach(dir => {
            let r = row + dir.dr;
            let c = col + dir.dc;
            while(onBoard(r, c)) {
              if(!board[r][c]) {
                moves.push({r, c});
              } else {
                if(!sameColor(r, c)) {
                  moves.push({r, c});
                }
                break;
              }
              r += dir.dr;
              c += dir.dc;
            }
          });
          break;
        case 'Q': // Dame
          directions.push(
            {dr:1, dc:0}, {dr:-1, dc:0}, {dr:0, dc:1}, {dr:0, dc:-1},
            {dr:1, dc:1}, {dr:1, dc:-1}, {dr:-1, dc:1}, {dr:-1, dc:-1}
          );
          directions.forEach(dir => {
            let r = row + dir.dr;
            let c = col + dir.dc;
            while(onBoard(r, c)) {
              if(!board[r][c]) {
                moves.push({r, c});
              } else {
                if(!sameColor(r, c)) {
                  moves.push({r, c});
                }
                break;
              }
              r += dir.dr;
              c += dir.dc;
            }
          });
          break;
        case 'K': // König
          for(let dr=-1; dr<=1; dr++) {
            for(let dc=-1; dc<=1; dc++) {
              if(dr===0 && dc===0) continue;
              let r = row+dr;
              let c = col+dc;
              if(onBoard(r,c) && (!board[r][c] || !sameColor(r,c))) {
                moves.push({r, c});
              }
            }
          }
```

```
          break;
      }

      return moves;
    }

    function onCellClick(e) {
      if(gameOver) return;

      const cell = e.target;
      const row = parseInt(cell.getAttribute('data-row'));
      const col = parseInt(cell.getAttribute('data-col'));
      const piece = boardState[row][col];

      if(selectedCell) {
        const {row: selRow, col: selCol} = selectedCell;
        const moveFound = possibleMoves.find(m => m.r===row && m.c===col);
        if(moveFound) {
          boardState[row][col] = boardState[selRow][selCol];
          boardState[selRow][selCol] = null;

          if(boardState[row][col].type==='P') {
            if((boardState[row][col].color==='w' && row===0) ||
               (boardState[row][col].color==='b' && row===7)) {
              boardState[row][col].type = 'Q';
            }
          }

          whiteToMove = !whiteToMove;
        }
        selectedCell = null;
        possibleMoves = [];
        renderBoard();
        checkGameEnd();
        return;
      }

      if(piece && ((whiteToMove && piece.color==='w') || (!whiteToMove &&
piece.color==='b'))) {
        selectedCell = {row, col};
        possibleMoves = generateMoves(row, col);
        renderBoard();
        highlightCells(possibleMoves);
      } else {
        selectedCell = null;
        possibleMoves = [];
        renderBoard();
      }
    }

    function highlightCells(moves) {
      moves.forEach(m => {
        const cellDiv =
document.querySelector(`[data-row='${m.r}'][data-col='${m.c}']`);
```

```
        if(cellDiv) {
          cellDiv.classList.add('highlight');
        }
      });
    }

    function checkGameEnd() {
      let whiteKingAlive = false;
      let blackKingAlive = false;

      for(let r=0; r<8; r++) {
        for(let c=0; c<8; c++) {
          const p = boardState[r][c];
          if(p && p.type==='K') {
            if(p.color==='w') whiteKingAlive = true;
            else blackKingAlive = true;
          }
        }
      }

      if(!whiteKingAlive) {
        document.getElementById('status').textContent = 'Schwarz gewinnt! Weißer
König ist geschlagen!';
        gameOver = true;
      } else if(!blackKingAlive) {
        document.getElementById('status').textContent = 'Weiß gewinnt! Schwarzer
König ist geschlagen!';
        gameOver = true;
      }
    }

    function renderBoard() {
      const boardDiv = document.getElementById('board');
      boardDiv.innerHTML = '';

      if(!gameOver) {
        document.getElementById('status').innerHTML =
          'Am Zug: ' + (whiteToMove
            ? '<span class="turn-indicator">Weiß</span>'
            : '<span class="turn-indicator">Schwarz</span>');
      }

      for(let r=0; r<8; r++) {
        for(let c=0; c<8; c++) {
          const cellDiv = document.createElement('div');
          cellDiv.classList.add('cell');

          if((r + c) % 2 === 0) {
            cellDiv.classList.add('white-cell');
          } else {
            cellDiv.classList.add('black-cell');
          }

          cellDiv.setAttribute('data-row', r);
```

```
            cellDiv.setAttribute('data-col', c);
            cellDiv.addEventListener('click', onCellClick);

            const piece = boardState[r][c];
            if(piece) {
              const pieceKey = piece.type + piece.color;
              cellDiv.textContent = pieceToUnicode[pieceKey] || '?';
              if(piece.color === 'w') {
                cellDiv.style.color = '#FFFFFF';
              } else {
                cellDiv.style.color = '#000000';
              }
            } else {
              cellDiv.textContent = '';
            }

            boardDiv.appendChild(cellDiv);
          }
        }
      }

      initBoard();
      renderBoard();
    </script>
  </body>
  </html>
```