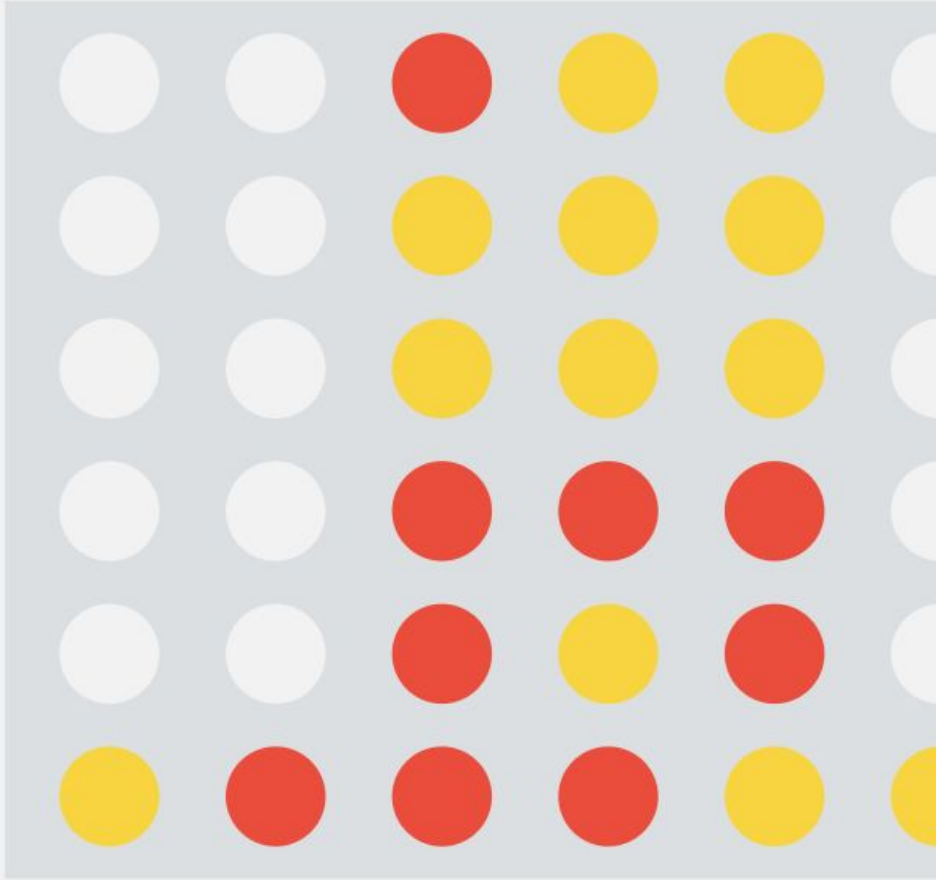
Connect 4

Leon Shams-Schaal

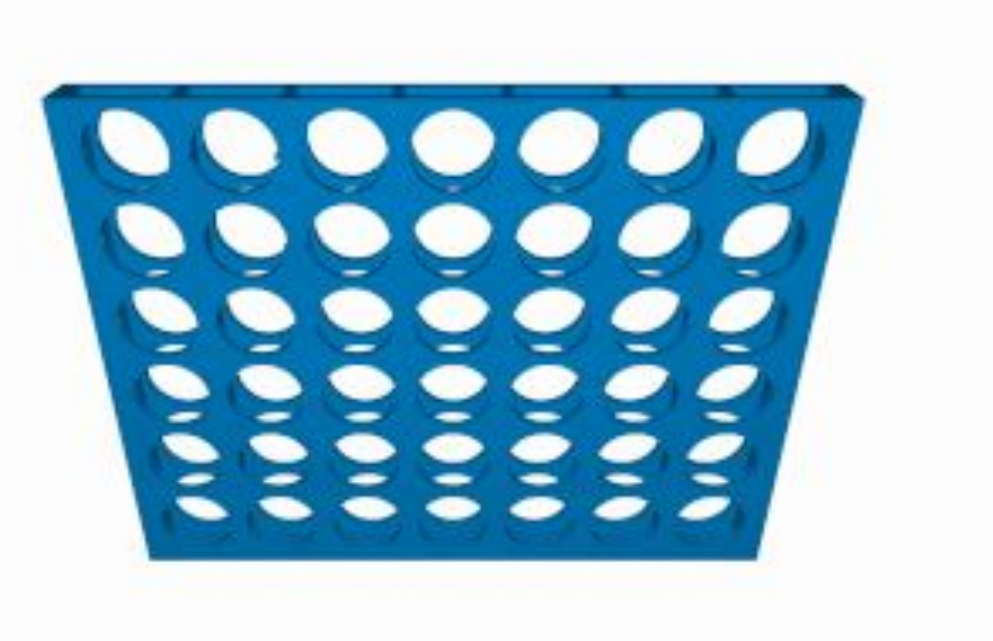


What is Connect 4

Connect 4 is a board game similar to tic-tac-toe where two players compete to get 4 pieces of their color in a row.

Pieces are placed at the top and dropped to the bottom of each column.

However, most importantly the game consists of a 7 discrete actions that result in deterministic outcomes, with a maximum of 42 moves possible in one game.

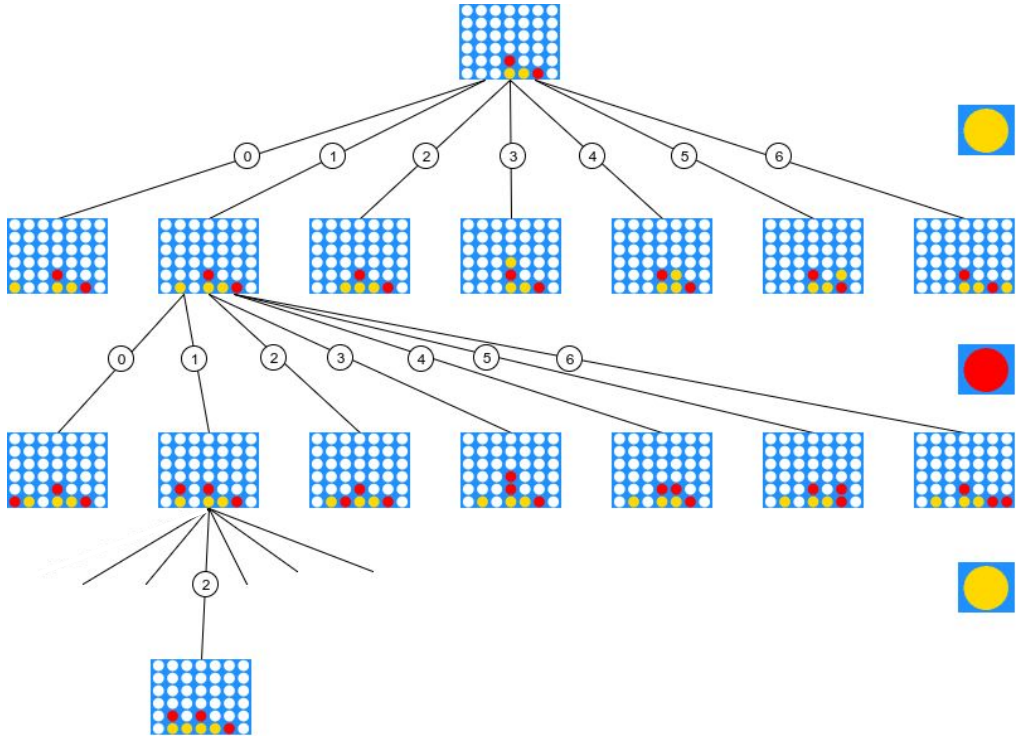


Our goal

I’m not going to explain building the game as it’s a pretty self explanatory process instead our goal will be to build an AI that plays the game at a human level.

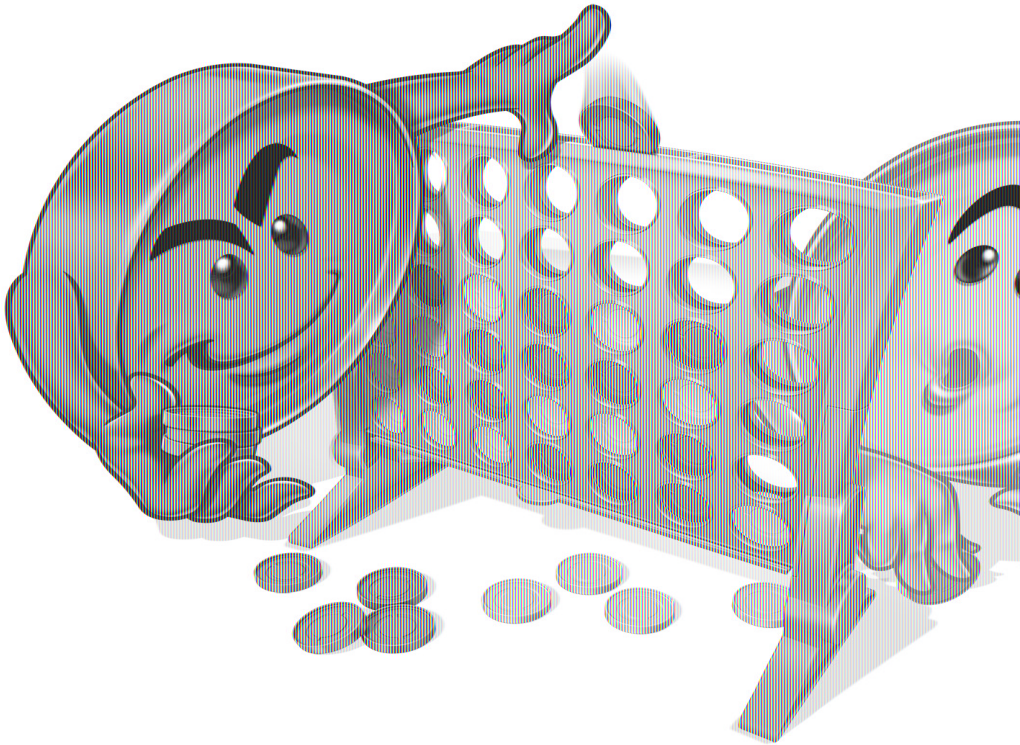
Our Approach

We will use a technique called minimax where we project future board positions until we reach a terminal state. The terminal state can then be assigned a reward (based on how many moves ahead it is) which will be propagated up the tree. Once all rewards have been propagated up we take the action with the highest associated reward.



Flaws

Connect 4 has about 4.5 trillion unique board positions making it extremely difficult to look into the future all 42 moves, so we going to take the easy way out and limit its depth to 12 moves ahead. Then as the game is played out column will fill up bringing the moves the explore down from seven allowing us to progressively search deeper. Furthermore, we can store all searched moves in a big dictionary which we can reference if we see a familiar state. Since there are repeating states in Connect 4 this reference dictionary will lead to a significant performance improvement allowing us to search the full 12 moves ahead at the beginning of the game.



Some code

Memoization function

```
var memoize = function (func) {
  const cache = {};
  return (...args) => {
    const key = JSON.stringify(args);
    return key in cache ? cache[key] : (cache[key] =
func(...args));
  };
};

minimax = memoize(minimax);
```

Action selection from minimax reward output

```
function bestMove(depth) {
  let bestAction = 0;
  let lowestScore = boardDims[0] * boardDims[1];

  let action, nextBoard, top, winner, score;

  if (Game.gameover(game.board)) {
    return 0;
  }

  for (let i = 0; i < boardDims[0]; i++) {
    action = columnOrder[i];

    nextBoard = copyArray(game.board);
    nextColumnFills = game.column_fills.slice();
    top = nextColumnFills[action];
    if (top < 0) continue;
    nextBoard[action][top] = game.turn;
    nextColumnFills[action]--;

    winner = Game.gameover(nextBoard);
    if (Game.gameover(nextBoard)) {
      score = -(boardDims[0] * boardDims[1] -
game.num_turns) / 2;
    } else {
      score = minimax(nextBoard, nextColumnFills, {
        num_turns: game.num_turns + 1,
        turn: -game.turn,
        depth: depth
      });
    }

    if (score < lowestScore) {
      lowestScore = score;
      bestAction = action;
    }
  }

  return bestAction;
}
```

Final Product

https://codepen.io/schaall/full/dyJbgZx