Consider any two player deterministic board game (no dice) (ex. tic-tac-toe). We can use the following algorithm to figure out next move

(move, score) best_move (board, Player)

1. for each possible next move μ of player
   1a) if playing μ wins immediately
   1b) return (μ, ∞)

2| for each possible move μ of player

2a| let $(\mu', k) \leftarrow$ best_move(board + μ, other player)

2b) Return μ s.t. k is minimized above.

Explanation The best_move(board, Player) returns the best move of Player on board and the minimum score the player can achieve using this

Move. We assume ∞ is the score for a win.

Board is some type that represents a board
A 3x3 char array for ttt.

Move represents an action of the player
A (row, col) pair for ttt.

Board + μ is the board after playing μ
from board.

For ttt, scores are +1, 0, -1
win, draw, loss

As soon as we see -1 in (2a) we can return as the other player will definitely lose if we play this move.

if there are no -1 in (2a) we return some move with score 0. if it exists.

o/w all moves lose, so we return an arbitrary move.

Why does best_move work?

best_move returns the move with the highest possible worst-case score over all moves.

You can prove this by induction on maximum number of moves left in the game if the current player plays an optimal move.

In the code:

```
for (int row = 0; row < 3; ++row) {
  for (int col = 0; col < 3; ++col) {
    if (board[row][col] == '.') {
      board[row][col] = player;
      if (has_won(board, player)) {
        board[row][col] = '.';
        return (move_t) {
          row,
          col,
          1
        };
      }
      board[row][col] = '.';
    }
  }
}

for (int row = 0; row < 3; ++row) {
  for (int col = 0; col < 3; ++col) {
    if (board[row][col] == '.') {
      board[row][col] = player;
      response = best_move(board, other_player(player));
      board[row][col] = '.';
      if (response.score == -1) {
        return (move_t) {
          .row = row,
          .col = col,
          .score = 1
        };
      } else if (response.score == 0) {
        candidate = (move_t) {
          .row = row,
          .col = col,
          .score = 0
        };
        no_candidate = 0;
      } else { /* response.score == +1 */
        if (no_candidate) {
          candidate = (move_t) {
            .row = row,
            .col = col,
            .score = -1
          };
          no_candidate = 0;
        }
      }
    }
  }
}
return candidate;
```

step 1

1a, 1b

This is only done if playing (row, col) wins immediately.

step 2

2a

If we play (row, col) other player will lose so play this.

A drawing strategy.

we haven't found a winning strategy yet. So store this as a candidate

2b

If we haven't found any winning or drawing strategy. store a losing one so that we can return something.