chooseMove(side)
Pseudocode

- See if the board is full (a leaf), and if so determine value: 0 for win by human, 1 for draw, 3 for computer win, and return it.
- If side == COMPUTER, set opponent opp = HUMAN and initialize value to 3 (highest) for running min. Else if side == HUMAN set opp = COMPUTER and init. value to 0 (lowest) for running max
- Find the blank spots in the board, and for each:
  - Fill it in for this side (the trial move)
  - Call chooseMove(opp) to find best countermove by opponent, and update running max or min with it (looking for weakest best-countermove that the opponent could come up with)
  - Undo the fill-in to return board to old state to try next spot.
- Return the best move-from here for this side to the caller

The minimax search, part 1

```java
public Best chooseMove(int side) {
  int opp; // The other side
  Best reply; // Opponent's best reply (a move)
  int simpleEval; int bestRow = 0; int bestColumn = 0;
  int value;
  if( (simpleEval = positionValue() ) != UNCLEAR )
    return new Best( simpleEval ); // leaf
  // set up opp, the opponent, for non-leaf case--
  if (side == COMPUTER)
    { opp = HUMAN; value = HUMAN_WIN; }
  else
    { opp = COMPUTER; value = COMPUTER_WIN; }
  return new Best( value,
```
Dynamic Programming

- For dynamic programming, we save results as we go along, and reuse them.
- Here, each board position has a value, so we need to save that value for each board, up to 19,683 of them.
- Idea: we need a Map of board → value

Position class

- Code: at Weiss's website, for Chap. 10, in TicTacToe.java
- The Position constructor copies in the board array contents.
- Equals checks for type (not perfectly*), then compares this.board and rhs.board element-by-element
- hashCode: Computes hashVal = 4*hashVal + board[i][j] across array elements, so 2 bits/value for array elements that can be 0, 1, or 2, i.e., binary 00, 01, 10. BTW, a perfect hash.
- So Position qualifies as a good HashMap element class.
  *It's OK as long as no code creates a subclass of Position, then tries to .equals its object using this code. We're supposed to use getClass() for the type check.

chooseMove(side)

- Choose if the board is full (a leaf), and if so determine value: 0 for win by human, 1 for draw, 3 for computer win, and return it.
- DP: Check if this board has a saved value and if so return it.
- If side == COMPUTER, set opponent opp = HUMAN and initial value to 3 (highest) for running min. Else if side == HUMAN set opp = COMPUTER and init. value to 0 (lowest) for running max.
- Find the blank spots in the board, and for each:
  - Fill it in for this side (the trial move)
  - Call chooseMove(opp) to find best countermove by opponent, and update running max or min with it (looking for weakest best-countermove that the opponent could come up with)
  - Undo the fill-in to return board to old state to try next spot.
- DP: Save the newly computed board value
- Return the best move-from here for this side to the caller

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.

Dynamic Programming

- Idea: we need a Map of board → value
- HashMap sounds great for this.
- The board is held in an array, int[3][3]
- Arrays in Java are objects, but they don't have hashCode or equals based on their content, just the Object methods, not good enough
- So we need to wrap the array in an object that we can provide with these crucial methods
- That's what Weiss's Position class does.
Dynamic Programming (DP)
in general

- What is it about chooseMove that makes DP work so well for it?
- Once we have computed the value of a subtree, it is useful for many bigger calculations.
- Example of a problem that can’t use DP: find the median value in a tree of nodes with values.
  - The median of a subtree is pretty useless.
  - Need to sort the values, take the middle one.

DP for Change Making

- Change making can use solutions of smaller problems: for change for N, consider using one penny along with change for N-1, one nickel along with change for N-5, etc., then choose best
- \#coins = change(N) = min_k (change(N-k) + 1)
  - Where k = coin size (1, 5, 10, ...)
- We saw that a greedy algorithm worked with US coinage, but not with other possible coin sizes
  - So DP can help with those harder cases.

DP Slides from Nurit Haspel

- Prof. Nurit Haspel has been teaching cs310 from 2009-2019 at umb.
- She uses DP in her own research, try Googling "Nurit Haspel DP" and see a paper.
- She is an enthusiast about it, so let’s look at her slides on the subject.
- The slides cover three topics...