How would you solve this problem using recursion?

Key insight: if my opponent has 0 winning moves after I make a move, then that move guarantees I win.

Basic ideas in backtracking search

- **Enumerate all possible choices/moves**
  - We try these choices in order, committing to a choice
  - If the choice doesn’t pan out we must undo the choice
    - Backtracking step, choices must be undoable

- **Inherently recursive, when to stop searching?**
  - When all columns tried in N queens
  - When a player has won or lost the gridgame
  - When the sum has been found, or when we know the sum cannot be found because we’ve gone over
  - In general: when we know if our choices have led to success or failure

- **Summary:** enumerate choices, try a choice, undo a choice, this is brute force search: try everything
Where we are thus far

- You have seen the general idea of backtracking:
  - Step 1: Make a choice
  - Step 2: Check to see if the choice leads to success
  - Step 3: If not, undo the choice and make the next one

- Coming Up: Subclasses: like interfaces but with code

- A few more hints on Boggle

Subclasses: When a Class “extends” Rather than “implements”

Inheritance concepts

- Parent/super class can be extended by subclassing
  - Possible to use methods from parent class, subs have them!
  - Possible to override parent methods, change behavior
  - Possible to do both! Call super.doThis();

- Often you don't have access to parent .java file
  - Still can subclass, use methods, extend/override them
  - Do NOT have access to private data fields
  - DO have access to protected data fields

- Hard to do OO design, leave for later courses
  - But get an idea now as to power and utility
Where we are thus far

- You have seen the general idea of backtracking:
  - Step 1: Make a choice
  - Step 2: Check to see if the choice leads to success
  - Step 3: If not, undo the choice and make the next one

- You’ve seen how subclasses work:
  ```javascript
  var superclass = new Subclass();
  var.printStuff(); //calls a subclass method
  var.printfoo(); //calls a superclass method
  ```

- Coming Up: A few more hints on Boggle

BinarySearchLexicon

- You do not have to implement binary search (it's in Collections.binarySearch) BUT you do have to sort the list before you use binary search (the provided code does this sort for you...just be aware in general)

- Three results from Lexicon (see SimpleLexicon for an example of how to return these)
  - NOT A WORD (e.g. “zz”)
  - A WORD (e.g. “do”)
  - Not a word, but the PREFIX of a word (e.g. “dy”)

Boggle Search for Word

- Starting at board location (row,col): find a string S
  - We want to keep track of where we are in the string
  - Also track what board locations used for S search

- How do we know when we're done?
  - Base case of recursive, backtracking call
  - Where we are in the string?

- How do we keep track of used locations?
  - Store in array list: tentatively use current one, recurse
  - If we don’t succeed, take off the last one stored!

Base Cases for GoodWordOnBoardFinder

- In to find a particular word on the board, I can think of 4 different kinds of base cases.
  1. Word is not at this square
  2. You’ve already used this cell
  3. You’ve found ever letter in the word
  4. You are off the board
- Does the order we check these base cases matter?
Using Howto Hints

- **Helper method in GoodWordOnBoardFinder**
  - Needed: board, row, col, word, word-index, list
  - Search for "skunk" at (0,0) at (0,1) at ...
  - Call `help(board, row, col, "skunk", 0, list)`
  - If we find the 's', where do we look next and for what?
  - If we don't find the 's' what does that mean?
  - If (row,col) isn't legal position then ...
  - If we've found the last 'k' what does that mean?

Where we’ve been

- **You have seen the general idea of backtracking:**
  - Step 1: Make a choice
  - Step 2: Check to see if the choice leads to success
  - Step 3: If not, undo the choice and make the next one

- **You’ve seen how subclasses work:**
  - `SuperClass var = new Subclass();`
  - `var.printStuff();` //calls a subclass method
  - `var.printFoo();` //calls a superclass method

- **A few more hints on Boggle**
  - You don’t have to write your own search for `BinarySearchLexicon`
  - Be careful with your base cases in `GoodWordOnBoardFinder`