YEAH! Hours - Fun With Collections

Ricardo Iglesias (That's me!)

Part 1 - Crystals!

An idea developed by Stanislaw Ulam (really interesting guy!)

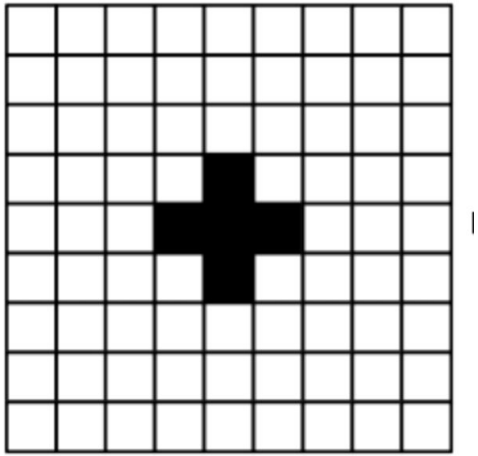
Have a two-dimensional grid and place a crystal.

Next, the crystal grows in each cardinal (N/S/E/W) direction.

Each of *those* crystals grows in a cardinal direction

When things get crowded, only cells with *exactly one adjacent neighbor* get to have a crystal.

Part 1- In Action!



Part 1 - Implementation Details

- Pretty simple... Just use a Grid<Bool> right?
  - ...Not exactly. If that were the case, we'd need to know how big our crystal's going to be
  - Solution: *Set<Point>*! This allows us to have as many points as needed, but how to keep track of what to do...
  - It turns out that a *Queue<Point>* works beautifully, since Queue's are a great way of modeling *to-do lists*.

Part 1: Implementation Details:

- Two functions:
  - ***`void crystallizeAt(Crystal& crystal, int x, int y);`***
    - Responsible for simply adding a crystal at <x,y>,
    - Adds things to the Queue
  - ***`void step(Crystal& crystal);`***
    - Moves forward one generation.
    - This means processing everything in the Queue *while also* getting the Queue set up for the *next* generation.

Tips:

- Recursion isn’t necessary!
- It can very difficult to *eyeball* if your solution is right. Write tests!
- You shouldn’t need to write a lot of code for this.
- Go to LaIR :)

Part 2: EVIL HANGMAN

- It’s hangman! With one twist:
- The computer cheats! How so?
  - You have a dictionary of words. When the user chooses a letter, you see all the possibilities where that letter could be in. You then choose the group with the *most* amount of words in it.
  - If that group doesn’t contained the guessed letter, the user didn’t “guess correctly” and so you mark them as incorrect.

Part 2: Implementation Details:

1) *Set up the Game*
    a) Prompt the user for word length. Reprompt if there are not words of that length.
    b) Prompt user for the number of guesses.
    c) Prompt user if they want a running total of the words remaining in the word list.
2) *Play the Game*
    a) Print out how many guesses the user has remaining
    b) Propmt for a single-letter guess
    c) Partition list of words into groups/families, and choose the largest one.
    d) Repeat
3) *Report Result*, and *Ask to play again.*

Tips:

- Choose Appropriate Collections
- Decompose!
- Think about how you should pass arguments to functions around.
- Letter position matters as much as frequency (BEER is not in the same family as HERE)