Administrivia

- Reminder: Homework 5 code due today (okay to turn in tomorrow if coming to office hours).

- Reminder: Homework 6 design due Tuesday.

Slide 1

Homework 6 — GUI Features

- Overall layout of game is `BorderLayout`, with screen in middle and "game status panels" on four sides — returned by `getGameStatusPanel` (in player), usually a `JPanel`.

- Menu bar is in `GameSetup`, can be modified.

Slide 2

- Screen editor program has support for "editing properties" (of screens, blocks, entities) — `getEditPropertiesPanel`. Could use this to give slightly different properties to different instances (e.g., walls of different colors, enemies with different speeds).

- Homework 6 asks you to use these features to (1) display something, and (2) get input from the user (either in the game or in the screen editor).

Slide 3

Recursion — Recap and More Examples

- Recap — identify base case, figure out how to split any other case into smaller instance(s) of problem plus othe processing.

- Example — list-like data structures.

Slide 4

Trees — Mathematical Definition

- One definition —
    - Set of nodes, one called root.
    - Set of edges (directed connections between nodes).
    - Root has no incoming edges; all other nodes have exactly one (from parent).
    - Each node can have 0 or more outgoing edges (to children — if none, leaf node).

- Another, recursive definition — tree is one node connected by edges to 0 or more subtrees.

- This is a general tree — e.g., to represent hierarchy such as filesystem.

Implementing Trees

- Define `Node` data structure, analogous to linked list, with reference to data and references to children (array or linked list or . . .).

- Easier if number of children is limited to two, and this turns out to be sufficiently useful in practice — "binary tree". Then `Node` consists of pointers to data and left and right subtrees.

Slide 5

Tree Traversals

- For linked lists we defined a way to visit all elements — "iterator". Is there something analogous for trees?

- Well — three orders that are easy to define and implement:

  - Preorder — root first.

  - Postorder — root last.

  - Inorder — leftmost subtree first, then root, then remaining subtrees. (Admittedly a little weird for non-binary trees.)

- (Sketch some code for at least one of these.)

Slide 6

Slide 7

Minute Essay

- None — quiz.