Slide 1

Administrivia

- Reminder: Homework 6 design due Tuesday.

Slide 2

GUIs and the Project

- 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.
- 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

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.

Slide 4

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

Sorted Binary Trees (Binary Search Trees)

- Key property — everything in the left subtree is smaller than the root, and everything in the right is bigger.
- Why is this useful? If you want a data structure to hold a collection that will be searched frequently, what are the choices? and how fast is each to search? to modify (insert/remove)? Compare approximate times for arrays (sorted and unsorted), linked lists (sorted and unsorted), sorted binary tree. (Next time.)
- (Sketch some code for `add` and `find`. `remove` is trickier — next time.)

Slide 7

Minute Essay

- Suppose you build a sorted binary tree by adding the following elements. Sketch the resulting tree.

  10, 5, 1, 12, 2

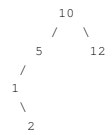
- What would be different if the elements were added in the following order? (Sketch this one too.)

  1, 2, 5, 10, 12

Slide 8

Minute Essay Answer

- Adding elements in the order given produces a tree that looks like this:



- Adding elements in order gives a tree that's basically a list — 1 is the root, and each non-root node has only a right child.