Slide 1

Administrivia

- Reminder: Homework 3 code due today. Homework 4 to be on Web soon, due dates next week.
- If I haven't said this before: Please do *not* reboot the machines in HAS 340; they are sometimes used by students and faculty for long-running background work, and rebooting disrupts that. If you think a reboot is needed, find a faculty member and ask him/her to make the decision.

Slide 2

Homework 3 Hints — General

- Remember that most game framework interfaces and classes are generic, so to use them you should supply two "type parameters" (your block and entity interfaces). Why is it written that way? should start to become clearer with this assignment.
- Two groups of methods to define:
  - Methods of two framework interfaces, `Player` and `GameEntity`. Called once per game tick (normally — but you can have it called less often).
  - Methods of appropriate listener interface(s). Called when human player provides input.
- Think about what variables you need — in general, if there's something that's part of the object's "state" and needs to be used by more than one method, it should be an instance variable.

Slide 3

Homework 3 Hints — Drawing Things

- Individual blocks and entities: What's drawn is controlled by `getImage`. Blocks all scaled to same size. Entities scaled based on `partialSizeX/Y`. Positions of entities based on locations.
- "Partial"? The framework allows you to define, for the purposes of moving and scaling, a "partials in whole" number (to allow moving in fractions-of-a-block units).

Slide 4

Homework 3 Hints — Drawing Things, Continued

- Laying out screens: You can do this in code (probably in your screen class) or using the "screen editor" (brief description and links to more info in writeups for Homeworks 2 and 3).

  Potential "gotcha": If you set "partials in whole" to a non-default value (in your game setup class), and you want to use the screen editor, you also need to set "partials in whole" in your screen class.
- Notice / recall that not everything has to be part of the playing field: Your game can also include "panels" on any or all four sides. (We won't add those until Homework 6; for now you could consider just printing information to the console.)

Slide 5

Homework 3 Hints — Responding to Input, Moving Around

- Game ticks and keyboard/mouse events aren't particularly in synch.
- So listener methods should probably just record information, to be processed by `update` method.
- Look at documentation of (Java library) listener interfaces to know what methods to write. Follow links to find out about other useful classes (e.g., `KeyEvent`). You may only need to put code in `keyPressed`.
- "Move" by changing location. Useful methods in (framework) `Location` class.

Slide 6

Homework 3 Hints — Interacting With Blocks

- At least some code for interacting with blocks goes in player classes. Similarly for other entities. However, good use of block hierarchy can help.
- Example — how do you not go through walls?

  (Contrast the "old way" using `instanceof` versus the "new way" using polymorphism and your interfaces.)
- Interacting with other entities starts in Homework 4, and is done in a similar (but not exactly the same) way.

Slide 7

Lists

- List ADT:
  - "Values" are lists of elements.
  - Many operations possible — add element, remove element, search for element, etc., etc. Also "walk through elements" with "iterator".
- Implementation:
  - Using an array.
  - Using a "linked list".

  How do these compare with regard to efficiency of various operations? efficiency of memory use?

Slide 8

Linked Lists

- Think about implementing some basic list operations (add, remove, find) using a linked list. First, draw pictures . . .
- Then think about what you need to turn the pictures into code. Probably you'll need:
  - Variables (e.g., something to point to the first "node" (little box).
  - Classes-within-the-class (for nodes / little boxes, iterators).
  - Methods for interface.
- (Code as time permits.)

Slide 9

Iterators

- Something we often want to do with lists and other "container classes" is do something to all elements — i.e., we want a way to visit all elements, in some (or any) order.
- An object-oriented way to address this is to have "iterator" objects with methods to support "visit every element, one at a time". In Java — `java.util.Iterator` interface. (Look at its methods.)

Slide 10

Iterators, Continued

- To see how this plays out in code, we could define a simple interface for lists, including an iterator, and implement it using arrays.
- Let's do that . . .

  (No, it's not a very sensible implementation of the list ADT, but it's one I'm willing to put on the Web as sample code. You'll write a linked-list class as part of Homework 4.)

Slide 11

Minute Essay

- None — sign in.