Slide 1

Administrivia

- Midterm scheduled for next time. Reschedule? (If so, also reschedule Quiz 3.)
- Reminder: Homework 2 code due today (midnight). Homework 3 design due Tuesday, code Thursday.

Slide 2

A Little About Homework 3

- In this homework you start writing code for your player, to replace the stick figure in the starter game.
- Key parts of this assignment are making the player
  - interact with different kinds of blocks.
  - move in response to keyboard or mouse input from human player.

  (If these don't apply to your game, talk to me about whether there are reasonable substitutes.)

  For design phase, you just need to describe this interaction.

Slide 3

Homework 3, Continued

- `Player` defines some constants you should use.
- You will implement `KeyListener` or one/both of the mouse-listener interfaces. When you do this, the framework will deliver key and/or mouse "events" to you.
- Most logic will go in `update`, `getUpdateTime`, and the listener methods.
- A general comment: If you find yourself looking up something like the ASCII value of a character, or the value of one of the game framework's constants — *stop*. There is probably an easier and more Javaesque way to do what you want.

Slide 4

Sorting and Searching, Continued

- Recall the problems — "sorting" to put an array (or list) in order (based on some ordering), "searching" to search for an element in an array (or list).
- Sorting algorithms include simple-but-slow (bubble sort, selection sort, insertion sort), faster-but-more-complex (to be discussed later).
- Searching algorithms include sequential search, binary search (faster but required sorted array/list).
- What do "slower" and "faster" mean here? Defined in terms of "order of magnitude" of algorithm.

Slide 5

Order of Magnitude of Algorithms

- Idea is to estimate how work (execution time) for algorithm varies as a function of "problem size" (e.g., for sorting, size of array). (Similar idea can be applied to how much memory is required.)
- Usually do this by counting something that represents most of the "work" in the algorithm and varies with problem size (e.g., for sorting, how many comparisons).

Slide 6

Order of Magnitude of Algorithms, Continued

- Informally, $O(N)$ means work/time is proportional to N (problem size). $O(N^2)$ means . . . ?
  (Compare aN and bN^2 as N increases, for different values of a and b. bN^2 larger for larger enough N.)
- Formal definition (from CSCI 1323): $g(n)$ is $O(f(n))$ if there are positive constants n_0 and c such that for $n \geq n_0$,

$$g(n) \leq cf(n)$$

Order of Magnitude of Sorts and Searches

- Usually we count comparison (and sometimes also swaps).
- How many comparisons for simple-but-slow sorts?
- How many for sequential and binary search?

Slide 7

Order of Magnitude of Sorts and Searches, Continued

- Bubble sort: For N elements, first pass through the array makes $N-1$ comparisons, next pass makes $N-2$, etc. Total is $(N-1)(N-2)/2$ — which in order-of-magnitude terms is $O(N^2)$.
- Selection sort and insertion sort are also $O(N^2)$.
- Quicksort and mergesort are $O(N \log N)$. (More about this later.)
- Sequential search is $O(N)$, binary search $O(\log N)$.

Slide 8

Slide 9

Polymorphic Sorting and Searching

- Sort/search algorithms are (mostly) independent of the kind of data being sorted — all of the comparison-based sorts just require that a "total ordering" relation on the data (for any two distinct elements a and b, $a < b$ or $b < a$). ("Comparison-based"? yes, as opposed to, e.g., radix sort or counting sort described last time.)
- So we'd like to be able to turn the algorithm into code just once, and let it operate on different kinds of data — "polymorphic sort". C's `qsort` is polymorphic, though the mechanics are a bit ugly. Java provides nicer mechanisms — for objects anyway.

Slide 10

Polymorphic Sorting and Searching in Java

- Java library interface `Comparable` is helpful in writing comparison-based sorts. (Look at its API. Example code as time permits.)
- But what if you sometimes want to sort data one way and sometimes another? With C's `qsort` you can pass in a function pointer. In Java? You can't do that. What you can do (very typical) is create an object whose purpose is to contain the desired code. Here, we want something to hold our `compareTo` method. Simplest to illustrate using a library class (next slide).

Sorting and Searching Arrays in Java

- Writing your own sorting routines is pedagogically useful, but in practice you would probably use something from Java library.
- `Arrays` class has some useful methods. The ones for objects require either a class that provides a `compareTo` method, or a `Comparator` object that defines the ordering you want.

Minute Essay

- For some well-known problems, the best known algorithms are $O(N!)$ (N factorial). Why is this a problem (or is it?).

Minute Essay Answer

- Because $N!$ increases so fast that it severely limits the size (N) of the problem that can be solved in a reasonable amount of time.