Eric Roberts
CS 106A

Handout #58
February 26, 2010

Adventure Contest

Due date: Friday, March 12, 5:00P.M.

Assignment #6 requires you to build a minimal Adventure program. In terms of features, it has about as little as you can get away with and still have a chance to construct data files that allow it to play even mildly interesting games. For the Adventure contest, your goal is to extend your Assignment #6 submission to be the most interesting Adventure game you can create. To create your new game, you will have to change both the program and the data files. You are free to incorporate parts of the Crowther cave, but you will be judged only on the material you add. The most exciting and playable submission wins.

For the purposes of the contest, it is acceptable to abandon the principle that programs of this sort should be entirely data-driven. In Will Crother's original Adventure game, the FORTRAN program included highly specific code to implement most of the operations. In particular, each of the action verbs was associated with a section of code that explicitly took care of each possible situation. If the player typed in **WAVE**, for example, the program would explicitly check to see what object was being waved and whether the player was standing in a particular room. If the right conditions applied, the program would take whatever actions were necessary to update the state. For your contest entry, you should feel free to do the same thing.

Possible extensions

The following extensions would make the Adventure program much more powerful and would allow the construction of more interesting games:

- *Active objects*. The biggest weakness in the current game is that the objects are entirely passive. All you can do with an object is to pick it up or drop it. Moreover, the only way in which the objects enter into the play of the game is in the specification of locked passages in the room data file: if you're carrying an object, some passage is open that would otherwise be closed. It would be wonderful if it were possible to type **WAVE WAND** or **UNLOCK GRATE** and have the appropriate thing happen. Moreover, being able to **READ** or **EXAMINE** an object adds a lot of interest to the game.
- *Permanent objects*. Many objects such as the metal grate are in fixed locations. In the Assignment #6 version of the game, these features are part of the room description and not part of the object or vocabulary files. This situation leads to silly exchanges like

```
You are in a 25-foot depression floored with bare dirt.
Set into the dirt is a strong steel grate mounted in
concrete.  A dry streambed leads into the depression from
the north.
>TAKE GRATE
I don't know the word 'GRATE'.
>
```

  It would make far more sense if the grate were an object that could not be taken.
- *Object state*. In the original version of Adventure, objects can have different states. For example, the grate at the entrance to the cave can be either locked or unlocked; similarly, the snake in the Hall of the Mountain King can be blocking your path or driven away. You might add some way to allow the program to keep track of the state of each object and then make it possible for the motion rules to indicate that a

particular passage can only be taken if an object is in a certain state: you can go through the grate only if it is unlocked.

- *Containment*. In Don Woods's extension to Adventure, some objects can contain other objects. Putting this concept into the game adds dimensionality to puzzle construction, but also requires implementing prepositional phrases in the parser so that the program can parse such constructions as

```
>PUT NUGGET IN CHEST
```

- *Filler words*. The current parser limits the player to using commands that consist of one or two words. Saying

```
>TAKE THE KEYS
```

causes an error because the program doesn't know the word **`THE`**; if the parser ignored articles and other filler words, the program would seem more conversational.

- *Adjectives*. A similar extension to the parser is the introduction of adjectives that allow the player to issue commands like

```
>TAKE BLACK ROD
```

In the classic Adventure game, adjectives are associated uniquely with the noun to which they refer. In Zork, on the other hand, adjectives were used to differentiate many different objects of the same time, so that there could be both a black rod and a green rod in the same game.

- *Convenient shorthands for "all" and "it"*. When you're in a room with many objects, it is extremely useful to be able to type

```
>TAKE ALL
```

to take all the objects at the location. Similarly, the conversation flows more smoothly if you can refer to the last mentioned object as **`IT`**.

- *Random passages*. There are several rooms in the original adventure game at which the motion through a passage is probabilistic. You could implement this sort of feature by specifying a percentage chance on a locked passage rather than an object. Thus, if the data for a room specified the connection entries

```
SOUTH       17/30
SOUTH       18
```

moving south would go to room 17 thirty percent of the time and to room 18 the rest. (The program can differentiate this specification syntax from traditional locked passages because the percentage chance begins with a digit.)

- *Lazy evaluation of the rooms database*. When the data file for the rooms is large—as it is with the **`Crowther`** files, for example—the adventure program can take a long time to start up. That startup delay is not really necessary, because the rooms module doesn't actually need to read the data all at once. It would work equally well to open the data file and then to initialize all the rooms to a **`null`** object. Whenever the code called a function that required the actual data for a room, the **`AdvRooms`** class could process the data file (presumably initializing other rooms in the structure along the way) until it got to the desired room. In programming, this strategy is called **lazy evaluation.**

- *Room names*. When writing the data files, it is hard to keep track of the room numbers under the current design. It would be extremely useful if the data files instead used names for the rooms, as in the following excerpt from a rooms data file:

```
OutsideBuilding
Outside building
You are standing at the end of a road before a small brick
building.  A small stream flows out of the building and
down a gully to the south.  A road runs up a small hill
to the west.
-----
WEST      EndOfRoad
UP        EndOfRoad
NORTH     InsideBuilding
IN        InsideBuilding

EndOfRoad
End of road
You are at the end of a road at the top of a small hill.
You can see a small building in the valley to the east.
-----
EAST      OutsideBuilding
DOWN      OutsideBuilding
```

- *Graphics and sound*. Given the success of games like Doom and Myst, the feature that most people would like to add to this game is graphics of some kind. You are free to add graphics and sound to the game, although doing so places you in a different judging category, as outlined later in this handout.

- *Multiplayer interactivity across the network*. Some of the most popular games available today are the massively-multiplayer online role-playing (abbreviated as MMORG or MMORPG, depending on whom you ask) games like World of Warcraft or EverQuest). While it might initially seem completely infeasible to build a multiplayer networked game in CS106A, the fact that we already have FacePamphlet changes the equation. It should be possible for friends on FacePamphlet to share information through the repository.

The real test of an adventure game, however, is not how many features the driver program contains but rather the quality of the puzzles you have designed. As you write your contest entry, you should focus on implementing those features that allow you to design a better and more playable game.

Submitting entries

To enter the contest, you must submit the following items to the Adventure Contest assignment on the web page:

- A version of the `Adventure.java` file that always reads in the particular data files relevant to your contest entry instead of asking for the name of an entry. The comments at the top of `Adventure.jar` should summarize any extensions you have made.
- Any relevant data files.

Judging categories

Past experience has shown that it is impossible to compare entries that use graphics and sound against purely text-based adventure games. Each style has its own characteristics and requires very different kinds of creativity. The programming necessary to add graphics to the game is not really that difficult, but it takes a lot of time to collect all the images that are appropriate to the game. A text-based adventure often requires more programming—often in the form of adding sophisticated features such as containment or object state to the game—to achieve the same levels of playability. Thus, there are two separate categories of prizes for this game:

1. Purely text-based adventures that make no use of graphics or sound.
2. Graphical adventures that make use of these features in any way.

The entries will be judged separately, and we will award a grand prize in each category.

Prizes

The first prize in the Adventure contest is that you (or both members of your team if you are working in pairs) will be able to replace the lowest individual score—assignment, midterm, or final—with a 100%. Unfortunately, the winners will not be announced until the final review session on Sunday, March 15, which means that you won't know about your victory until the very last minute.

Instead of awarding runner-up prizes, each extension you make to the game will be treated as an extension to Assignment #6 and bring you closer to a + or ++ score.

Contest rules

1. Only students registered in CS 106A are eligible to submit entries.
2. If you are working as a two-person team on Assignment #6, you must submit your entry as that team. Both members of a team will receive the prize.
3. Your entry is due by 5:00P.M. on Friday, March 12 and must be submitted electronically to the **`AdventureContest`** assignment on the CS106A web site. Thus, only assignments that are submitted on time are eligible for the contest.
4. Contest entries should be sensitive to Stanford's individual and cultural diversity. Programs or narratives which have the effect of perpetuating negative stereotypes will not be accepted.
5. The contest entries will be judged by Eric Roberts and Chris Piech, who will award the final prize to the two entries—one in each category—that implement the most interesting adventure game. Decisions of the judges are final. Winners will be announced at the review session on Sunday, March 14.