MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.090—Building Programming Experience
IAP 2007

Problem Set 7
Due Tuesday January 23, 1pm

This project is aimed at giving you experience modifying code that you did not write. The objective is for you to understand just enough of how the code is working to make the modification. The top of this document describes the system from a high level.

Munchkin MIT

Munchkin MIT is a game that reduces the MIT experience into a "card game" which pits two players against each other to acquire enough clue to escape. The process of acquiring clue requires doing battle with problem sets of varying difficulty. Success results in more clue, failure in a loss of sanity. The game ends when a player reaches 30 clue or runs out of sanity. Weapon cards can be played to make problem sets easier to complete. Download the file `advgame.scm` from the website, and evaluate the whole file. Then play the game:

```
(play-game null cards (initial-student "Ben") (initial-student "Jacob"))
```

Students

A student is a player of the game. A student is composed of a name and a list of properties. These properties include their sanity, their clue, their hand of cards, etc. Create a student with `(initial-student "your name here")` and test out the selectors `student-name` and `student-value`:

```
(define s (initial-student "Ben"))

(student-name s)
;Value: "Ben"

(student-value s 'sanity)
;Value: 10

(student-value s 'clue)
;Value: 0

(student-value s 'hand)
;Value: ()
```

One property of a student is a hand. A hand is a list of cards that they have not yet played. Another property is weapons, which is the list of weapons that the student has in play. Playing a weapon card puts a weapon in play for that student.

Cards

Cards come in a couple of varieties. pset cards are problem sets that students must defeat. weapon cards make it easier for students to defeat problem sets. There exist abstractions for each of these types of cards. The code that implements the effect of the card is in `pset-card-proc` and `weapon-proc`.

Problems

For each of these problems, turn in just the changes you made, and a log of test cases. Do not simply turn a copy of your `advgame.scm` file after doing to exercises.

Problem 1

Play the game. Attempt to win. Attempt to lose. What does sleeping through class do?

Problem 2

Change the game so that the game ends when one player reaches 20 clue instead of 30. Turn in the changes you made.

Problem 3

Change the game so that each player only has 3 cards in their hand not 5. Test your modification to make sure it works and turn in the change you made.

Problem 4

Change the game so that you only get the effect of the STRONGEST weapon you have, not the sum of all their effectiveness. Test your modification to make sure it works and turn in the change you made.

Problem 5

Implement chance cards. A chance card modifies the student's sanity rating (either up or down). Some of the implementation is already completed. Once you've filled in the code for the abstraction, uncomment the lines in the `(define cards ...)` code at the end of the file and re-evaluate the cards variable. Once you've written the chance-proc you will need to re-evaluate the card-procs definition to make your changes visible to the system. Then play again with the chance cards working.