ARTIFICIAL INTELLIGENCE IN CS1*

NIFTY COURSE ASSIGNMENTS

Brian C. Ladd
Assistant Professor of Computer Science
St. Lawrence University
Canton, NY 13617
315-229-5293
blad@it.stlawu.edu

ABSTRACT

The GuessingGame program from introductory artificial intelligence serves as a rewarding challenge for students in a Java-based CS1 course. The consistent use of manipulatives eases students into such complex topics as recursive data structures, Visitors, and Factories. The game aspect of this project connects the idea of separating data from code to their own experience with user community "mods" of commercial computer games.

ASSIGNMENT

Near the end of their first semester learning to program in Java many students need a project that illuminates the relationship between their programming the computer and their using the computer; animated robots and "Hello, World!" are not obviously related to word processing, communicating, or playing games. Students also benefit from a program that applies everything they have learned and gently introduces some design patterns. I have found that the standard "twenty questions" game fits this bill perfectly.

The GuessingGame, a staple in any introductory artificial intelligence text, asks the user to think of a member of some set of objects, typically animals or movie stars. Through a series of yes/no questions the program zeros in on the selected item until it hazards a guess. If the guess is wrong, the program asks the user what the right answer was and how to differentiate the right and wrong guesses; whenever it loses, the player improves the program's future performance. Programming this project in CS1 proceeds

---

in several phases over two weeks; some phases are done quickly in class, others sketched out and assigned as homework:

- *Introduce GuessingGame* (Instructor): First we play one or two on-line versions of the game [1]. Seeing the similarity of the interface when guessing in two different domains permits the introduction of the idea of a "GuessingGame engine" and a separate, replaceable, database.

  Punched index cards connected with string introduce the tree data structure and how the game is played by a student who can only "see" a single card. This manipulative is reused to introduce each new tree concept.
- *Design class hierarchy* (Group): **Question** and **Guess** classes, modeled on the index card manipulative, are designed. Commonalities are factored into a **DecisionTree** class and ad hoc groups implement the design in lab.
- *Implement GuessingGame* (Individual): The instructor guides the class through the design of a non-learning version of the game; students implement the design as homework. A hard-coded database (hand-built in the game's constructor) and a standard game loop are hallmarks of this phase.
- *Extend decision tree* (Group): Tying and untying strings between index cards develops intuition for extending the tree. Groups make and teach their games. Hard-coded trees are extended through play; losing all new information at restart motivates saving/loading trees.
- *Save decision tree* (Group): Instructor presents a pre-order **DecisionTree** pretty-printer and the class discusses how to reconstruct tree from the indented output. Type identifier tags are introduced and student groups modify their programs to save the entire database whenever it is extended.
- *Load decision tree* (Individual): The instructor guides the class through the design of a **DecisionTree** node factory (it switches on the type identifier tags and constructs the appropriate node). Students implement the load function as homework; they also remove the hard-coded database in favor of loading from a file.

When they return with a working program, a new, non-animal database is provided in their standard format. The very same program they wrote to guess the animal can now guess which movie star the user is thinking of; this emphasizes the power of separating code from data and provides another opportunity to compare this game to larger commercial game products.

This assignment is a fun way to recap object-oriented design, inheritance, file processing, and command-driven programming. A standard "game loop" (display-read-update) is presented; students are surprised to find out it is ast the heart of most interactive programs. Their simple "game engine" can work with different databases, permitting "mods" to be written for their game. The assignment uses the Visitor and Factory patterns and introduces CS1 students to recursion. It challenges the students who are going on the CS2 and serves as a final reward for those who choose not to.

REFERENCES

[1] gamemaster.com Homepage, http://www.gamemaster.com, retrieved January 20, 2005.