Computer Writing and Research Lab

White Paper Series: #040505-4

MOO Bots

Olin Bjork
objork@mail.utexas.edu
University of Texas at Austin

5 May 2004

Keywords: bots, MOO, object-oriented programming, regular expressions

Abstract: This paper explores educational ways to program and use bots in a MOO environment.

How MOOs differ from IM or IRC

Many CWRL instructors use our MOO (Multi-user domain Object Oriented) exclusively as a means of synchronous communication. When used in this way, the MOO is little more than a bulky alternative to Instant Messaging (IM) or Internet Relay Chat (IRC) applications. The new generation of IM software, such as Apple's iChat AV or Cerulean Studios' Trillian, may well be superior options for an instructor whose main purpose is to hold networked group discussions in real time and keep transcripts of those discussions. Interchange, a Daedelus Integrated Writing Environment (DIWE) program that the lab unfortunately can no longer support, was almost certainly superior. Interchange allowed an instructor to move between different group discussions and quickly get up to speed by skimming through the back histories (i.e., running logs) of those discussions, whereas in the virtual reality (VR) of a MOO, an instructor manipulates a "player" and can only read a record of those comments and actions he/she/it has been present to "hear" or "see"—the instructor will have to "play the tape" to find out the rest. Unlike IM or IRC, the primary function of a MOO is to model a world or worlds.

As text and/or image based virtual worlds, MOOs allow users to take on personalities and identities different from their real life (RL) personas. While many other chat environments enable this practice technically and may even encourage it socially, MOOs are role-playing environments by definition. MOO spaces are designed for exploration and discovery, not merely as venues for group conversation.

What makes MOOs truly unique, however, is that they combine

many of the features of an IM or IRC application (and often of a Web site) with an object-oriented programming (OOP) environment. Users can edit the players they own, and users whose players are builders can create new objects from existing object types, called classes. Users whose players are programmers, meanwhile, can create new classes and edit old ones.

What are Bots?

One of the most intriguing MOO classes is the bot class. A bot, in info-tech parlance, is a non-mechanical, intelligent user agent or software robot. The most common form of bot is the Web robot, or spider, a bot that indexes information from Web pages for search engines or spammers. Next most common is the chatterbot, or natural language processing program, "a bot capable of carrying on a conversation with a human."[1] A MOO bot is a species of chatterbot described by Cynthia Haynes and Jan Rune Holmevik as

> An object designed to interact in real time with MOO users in its vicinity. The bot (short for robot) consists of a series of programs that interpret and act on conversations and sentence patterns through random responses or question responses, all of which are user programmable. The most common bot in enCore based MOOs was designed by Ken Schweller of CollegeTown MOO, but a number of other types of bots can also be encountered in MOOs, including bots that can "walk" from location to location, and bots that can "learn" and "remember" from their interactions with users. Bots require more advanced building and programming skills, but can be used creatively for tutorials, dramas, or presentation of research, among other things.[2]

[1] Leonard, Andrew. *Bots: The Origin of New Species*. San Francisco: HardWired, 1997. 190-95.

[2] Holmevik, Jan Rune and Cynthia Haynes. *MOOniversity: A Student's Guide to Online Learning Environments*. Boston: Allyn and Bacon, 2000. 124.

MOO bots, like players, live in the MOO. But when a user is not logged in, his or her player is asleep. A bot is never asleep or awake, it is either active or inactive. And an active bot, unlike a player owned by a user who has forgotten to disconnect from the MOO, is never idle. In fact, an active bot can be a quite an annoyance, because bots respond to nearly everything they hear. Creating a bot that is more interesting than irritating is quite a challenge, but the effort alone can be a rewarding experience for students and teachers.

Bots, players, and the Turing test

In Allan Turing's famous 1950 article, "Computing Machinery and Intelligence," he describes an experiment called the "imitation game" in which an interrogator (C) tries to determine the sex of two unseen

persons, a man (A) and a woman (B), based on printed messages. A's object is to deceive C by passing himself or herself off as a person of the opposite sex. B wants C to guess correctly. Turing then proposes a version of this experiment in which position A is taken over by a machine. Turing believes that a machine can be programmed to perform just as well as a human in that the interrogator will be no more successful in distinguishing between person and machine than he or she was in distinguishing between the sexes. If a machine can "fool" the interrogator, argues Turing, it is intelligent.[3] This spin on the imitation game has come to be called the "Turing Test," and has been a focus of many discussions in Artificial Intelligence (AI), philosophy, and cognitive science for over 50 years. To this day, no chatterbot has ever passed a rigorous Turing Test.

[3] Turing, Allan. "Computing Machinery and Intelligence." *Mind* 59 (1950). <http://www.abelard.org/turpap/turpap.html>

In general, MOOs are not the best environments for such games or tests, as user's real names are not protected. For example, the CWRL implements Xpress, enCore's Web interface, for our ("Silver Sea") MOO. When users click on a player's icon, not only will they see a description and/or image of that player, but also the real name and e-mail address of the user who owns that player. Assuming that this user submitted an authentic name (a recommended practice), he or she has just been fingered! MOOs based on enCore also allow the command @whois "player name," so even if a player is communicating from a locked room, users can still identify the user behind the player (and quite possibly the user's gender as well).

How then can users distinguish between a player and a well programmed bot? There are myriad ways: if they click on the bot's icon, they will see no user info, only a description. If they use an @whois command, the system will inform them that the bot's name cannot be found in the player database. If they examine the bot, they will see a list of bot-specific commands. Thus, no bot can hope to fool a determined user. And even if the instructor outlaws any means of distinction other than direct communication, a user might still be able to distinguish between a bot and a character imitating a bot because bots tend to respond faster than most users can type.

The point is, enCore MOOs are about education, not deception. Students need not be in disguise; they need only be in character. MOOs are perfect for role-playing and devil's advocate exercises because in many cases students will not be able to assume that opinions expressed in VR on a RL issue are the same as their classmates' RL opinions on that issue. Bots add yet another layer between user and object; whose views are they programmed to express? It is often preferable, therefore, for students to adopt characters that are not mere simulacra of themselves in RL, but instead represent subject positions different from the ones they commonly find themselves in.

Programming Bots

A bot programming assignment will not only teach students about AI and OOP, it will help them think about language and rhetoric in a new way. But bot programming takes a fair amount of time to learn, so some instructors may be better served to create bots themselves and/or make bot programming an option for students who are particularly intrigued by the MOO or bots.

A generic bot can be created through the Xpress Object Editor, where it is listed under educational objects, or by typing the following (with a different bot name): '@create $bot named Tony Blair'. The default description of a bot is the following:

> You see a Turing Robot designed to interact in vigorous 'Eliza-like' conversation with other folks in its vicinity. It has key words, sentence patterns, random responses, and question responses - all user programmable. It has some ability to recognize where it is and to whom it is talking. Type @exam botname to see all the available commands. For detailed assistance in programming this bot type 'help botname'. For a discussion of the issues involved in testing machine intelligence see Turing's paper 'Computing Machinery and Intelligence'. To start up the bot, drop it, activate it, and say 'hi'. Please report bugs to ken/cdr@CollegeTown.

'Eliza' is a program developed in 1965-66 by Joseph Weizenbaum of MIT and named after Eliza Doolittle from Shaw's *Pygmalion*. Eliza responds to statements on any topic by asking questions in the manner of a Rogerian psychotherapist.[4] The faux dialogue that results can be fairly convincing (see http://www-ai.ijs.si/eliza/eliza.html).

When we enter 'help botname,' we see that there are three main ways to program a bot:

> 1. ADDING A WORD FOR YOUR BOT TO RESPOND TO: To see what words your bot already responds to, type 'seewords botname'. To teach your bot to respond to 'donut' with either 'I like donuts too.' or 'Donuts are very tasty!' just type 'addword botname' and enter the keyword 'donut'. Then enter the appropriate responses a line at a time. End with a single period on a line by itself.

Programming bots with keywords is an easy and effective way to make them remark on specific issues of interest. However, this method is insufficient to simulate an actual dialogue because it will soon become clear to the interlocutor that the bot is simply responding to keywords and is not actually processing any other words.

[4] Weizenbaum, Joseph. "ELIZA—A Computer Program For the Study of Natural Language Communication Between Man and Machine." *Communications of the ACM* 9.1 (1966): 36-45. <http://i5.nyu.edu/~mm64/x52.9265/january1966.html>

2. ADDING A PATTERN FOR YOUR BOT TO RESPOND TO:
Suppose you wished your bot to hear something like
MY DONUT ISN'T VERY TASTY
and respond with
WHAT'S SO GREAT ABOUT A TASTY DONUT?
To do this you must teach your bot to respond to the pattern
MY a ISN'T VERY b.
To understand what patterns look like, type 'seepat botname' and study the examples. For additional assistance on understanding the syntax of patterns type 'help regular'. When you think you are ready to add a pattern type '@addpat botname' and enter the following line when asked to do so:
my %(%w*%) isn't very %(.*%)
Then type in the response form:
What's so great about a %2 %1?
Add as many response forms as you wish on separate lines. End with a period on a single line.

Programming bots to find particular syntactical patterns and respond in appropriate patterns requires learning a few regular expressions: "a regular expression, or regex for short, is a special text string for describing a search pattern. You can think of regular expressions as wildcards on steroids. You are probably familiar with wildcard notations such as *.txt to find all text files in a file manager. The regex equivalent is .*\.txt."[5] To find out how to use the regexs implemented by the MOO programming language, type 'help regular-expressions'. Regexs are also useful in other contexts, such as searching electronic text or code.

[5] Goyvaerts, Jan. *Regular-Expressions.info* <http://www.regular-expressions.info/>.

3. ADDING RANDOM RESPONSES:
These responses are triggered whenever your bot can't find a keyword, a pattern, or a question. To see the responses already programmed type: 'seeresponses botname'. To add a new random response type: 'addrandom botname' and enter a new response.

4. ADDING A RANDOM RESPONSE TO A QUESTION
When your bot senses a question is being asked it responds with a random 'answer'. To see the random question responses already programmed into your bot type 'seequestionresponses botname'. To add a new response type 'addquestionresponse botname'.

Random responses will prevent your bot from going mute, but they can also drive interlocutors to distraction. So it is generally advisable to program random responses to questions only.

In addition to the four built-in methods discussed above, there are many other ways to program a bot. Bots can be programmed to travel around the MOO, open other objects, never repeat themselves, etc. For more information, see the MOO programmer's manual.[6]

[6] <ftp://ftp.lambda.moo.mud.org/pub/MOO/ProgrammersManual.html>

Pedagogical Applications

AI professionals are still struggling to design believable user agents that behave in effective and entertaining ways, so students and teachers who venture into bot programming should not expect to achieve such goals with the limited resources and time available to them. However, they should discover, as AI philosophers already have, that bot programming leads to interesting discussions about the nature of cognition and discourse.

One excellent pedagogical context for bots is the simulation assignment. Bots can enliven virtual spaces and rhetorical situations. For example, students assigned to do a MOO project on the Spanish Inquisition might choose to create a virtual Seville circa 1492. After building the cathedral and tribunal chamber, they might then decide to create a Tomás de Torquemada bot to preside there. Their challenge would be to program this grand inquisitor bot to ask players realistic questions, respond to their answers convincingly, and make fitting accusations. Players could then take turns being interrogated by this bot, and other players could serve as judges. Students could even contrive alternative historical events, like Christopher Columbus being interrogated instead of celebrated after confirming that the world is round.

Another context for bots is the group debate or discussion. Instructors should place bots in rooms where these meetings are to be held. After the players in each group have all arrived, they will activate the bots to serve as moderators. The bots will then provide a prompt to begin debate or discussion, and when certain keywords are mentioned by the participants, the bots will respond with new prompts. Since an instructor cannot moderate every group at once, using bots as proxies can be an excellent way to keep groups on track or veer them into new territory.

Bots are perhaps best suited to be guides in virtual museums or archives. A guide bot can be programmed to be a loquacious expert on a particular room or exhibit within a MOO. With more advanced programming, a guide bot can become a traveling companion for players, but this option often requires a type of annotated environment more suited to a Multi User Domain (MUD) than a MOO. MUDs tend to have relatively few themes and programmers, therefore rooms can more readily be designed to send appropriate information to the bots who enter them.

Bibliography

Goyvaerts, Jan. *Regular Expressions.info*. <http://www.regular-expressions.info/>.

Holmevik, Jan Rune and Cynthia Haynes. *MOOniversity: A Student's Guide to Online Learning Environments*. Boston: Allyn and Bacon, 2000.

Leonard, Andrew. *Bots: The Origin of New Species*. San Francisco: Hard-Wired, 1997.

Turing, Allan. "Computing Machinery and Intelligence." *Mind* 59 (1950).

Weizenbaum, Joseph. "ELIZA—A Computer Program For the Study of Natural Language Communication Between Man and Machine." *Communications of the ACM* 9.1 (1966): 36-45.