author at: Dept. of Psychology, University of Texas, Austin, TX 78712, or call (512) 471-9228.

George W Holden
Assistant Professor of Psychology
University of Texas
Austin, Texas, USA

ITERATIVE PRISONER'S DILEMMA: A PROGRAM FOR INSTRUCTIONAL AND EXPERIMENTAL USE

Jeffrey Hart and Marc Simon, Indiana University, USA

Abstract: This paper provides an introduction to a program called DILEMMA, written in Turbo Pascal for the IBM-PC, which illustrates the iterative prisoner's dilemma (henceforth, PD) game. The PD is a game theoretic model that has many applications in the social sciences, particularly in our field of international relations. It can be applied to arms races, mobilization races, alliance cohesion, decisions to initiate and escalate wars, or any collective goods problem.

Keywords: Prisoner's Dilemma, international relations, computer

Introduction

The general format of the PD model involves the decision of two actors who each have two choices - cooperation or noncooperation (defection). The outcome or payoff of each actor's decision is dependent on what the other decides. Taking a two-nation arms race scenario, if both nations cogperate and freeze their current armaments level, the payoff in reduced tensions and money saved could be represented as (2,2). If one side builds up heavily while the other side cooperates and freezes its arsenal, the defecting nation gains much political and military clout while the cooperating nation is now the "sucker" who appears weak and vulnerable to attack (10, 10). Finally, if both sides build (because each wants to avoid the sucker payoff), the payoff to both nations represents increased tensions and money spent on defense (-6,-6). The numerical value of the payoffs is meaningless - what is important is the rank order of the payoffs. This ordering creates incentives for an actor to defect, even though both players would be better off if they each cooperate. In an iterated game, the dilemma becomes finding ways to build trust so that mutual cooperation can be achieved.

DILEMMA was developed as an instructional tool to illustrate the strategic interdependence and nonzero-sum incentives present in many decision-making situations in politics. Though the PD seems fairly straightforward when first presented to students, many have difficulty obtaining a deeper understanding of how the game works in practice; these students are therefore skeptical of the relevance of the PD in international relations. This situation presents an ideal opportunity for the use of instructional software - by actually playing the game against different opponents with different decision-making strategies and under various time and payoff constraints, students begin to realize that this model is not as unconnected with real-world situations as they once thought. This paper will discuss the design and use of DILEMMA as well as some general criteria for the use of both instructional and experimental software in the social sciences.

A brief description of the program

The user is asked via a series of menus to record his/her initials so that a record of play can be kept in a separate disk file, to keep or change the original payoff matrix, to decide how many iterations to play (although the program leaves the option of keeping this uncertain), and to choose among five machine-generated players. The program keeps track of all moves, and at the end of each round it displays the choices made by the user and his/her opponent, along with a running tally of the cumulative scores of both players.

The first two screens of DILEMMA describe the story behind the game and give the "mobilization race" application mentioned above. Following this brief introduction, the initial matrix of outcomes is displayed:

		partner/opponent's move	
		C	D
your move	C		-10,0
	D	0,-10	-5,-5

After these on-screen instructions appear, the user is asked if he/she wishes to change the payoffs. As long as the relationship of the payoffs remains in the form of a PD, any change is allowed. The user then selects a partner opponent and, if desired, the number of rounds to play. The five machine-generated opponents have been given the whimsical names of 1) Joseph Stalin, 2) Marvin Miller, 3) Tina Tatter, 4) Larry Lawrence, and 5) Michael Marshmallow. Joseph Stalin always defects, Michael Marshmallow always cooperates, and Larry Lawrence plays randomly. This astute user will quickly learn that the two interesting opponents are Marvin Miller and Tina Tatter.

Tina Tatter uses the `friendly' tit-for-tat strategy recommended by Axelrod (1984). This strategy involves cooperating on the first move and then simply echoing what the other player did on the previous move from then on. Axelrod demonstrates in his book that the friendly tit-for-tat strategy scores highest (on the average) against all other programmed strategies.

Marvin Miller uses a version of an `unfriendly' tit-for-tat strategy: he defects on the first two moves, and then cooperates only when his opponent cooperates on the two previous moves. To the user who never cooperates twice in a row, Marvin plays the same as Joseph Stalin; however, if the user is persistent, the mutual cooperation outcome can be achieved.

The actual play of the game is quite simple: the computer asks the user to type "C" or "D", it gives the opponent's response, the payoff to each player, and the cumulative total points for each player. These scores are also written into a file that can be examined later or handed in to verify completion of an assignment.

The first version of DILEMMA was written on a VAX. It was modified for Turbo Pascal and the IBM-PC in December of 1984, and then revised again in January 1985. The program disk is not copy-protected and both compiled and source code are included for those who might wish to modify the program for special applications. DILEMMA has many strengths and weaknesses; its simplicity ranks high among its strengths. The following sections describe some design considerations, instructional use, and possible improvements for DILEMMA in the context of general criteria for educational software.

Educational use

The program is designed for use by undergraduates at colleges or universities, but it is simple enough to be used by high school students with some mathematical background. DILEMMA has been used as one of a set of required computer simulations in undergraduate courses (Introduction to World Politics, American Foreign Policy). Students are required to write short papers commenting upon and criticizing the computer programs they use. This has proven to be an excellent way of obtaining feedback on the design of the programs, as well as a means of forcing the student to think seriously about how the program applies to topics in the course.

The psychological nature of the prisoner's dilemma model is perhaps the most difficult aspect to get across to students in a lecture format. How should ' they view the other player in the game - as a partner or adversary? This is deliberately left ambiguous in the program to encourage students to think about this issue. Students nearly universally report that they found the program challenging and enlightening, while expressing some puzzlement about how to maximize their scores while still defeating their opponent. The incentives to cooperate or defect in an iterative game depend largely on the player's conception of the opponent. If the user views the opponent as an adversary, he/she will be more concerned with receiving a higher score than their foe, rather than a high overall score. The user will be less likely to trust the adversary, and will thus be less likely to attain a series of mutual cooperation outcomes. If the user can come to view the opponent as a partner or as an adversary who can be dealt with, then the mutual cooperation outcome becomes an achievable goal. Axelrod (1984, p. 176) argues that the tit-for-tat strategy's reciprocity is a means of tacitly communicating with an opponent and encouraging the development of a stable cooperative outcome.

In classes where this DILEMMA program was not used, we have found that many students never grasp the subtleties of the game. After playing the game, however, students find these ideas much easier to handle. In other words, the idea of a mixed-motive iterative game is a hard one to internalize, and an interactive computer program is a very good way of getting it across.

General criteria for instructional programs

For instructional computer programs, the following characteristics are desirable: 1) very readable instructions (preferably on-screen rather than in separate documentation); 2) the program should be highly interactive; 3) liberal use should be made of graphics and humor to keep the user motivated; 4) the content of the program should provide new insights into an important academic subject; 5) the program should allow the user to control the length of play to some extent; 6) the program should not be copy protected; and 7) the program should be user-friendly.

The need for instructions to be clear and simple is obvious. Perhaps what is not so obvious is that instructions for use be incorporated in the program itself, and that separate documents (which can be lost or stolen) should be used mainly to provide supplemental information. The reason for this is to make the program as accessable as possible to the student - they need to interact with the program itself and not spend time referring to a manual before and during the game or simulation. This may seem obvious to the reader, but be assured that it is often ignored in instructional software.

A highly interactive computer environment is important for instructional software so that the user and computer communicate, forcing the user to become more involved with the program and to think about the

subject material. This means that programs should use frequent queries, and that the user should get quick responses to his/her answers. Long delays create boredom, and can mentally distance the user from the program.

Graphics and humor are keys to the success of arcade games, and there is no reason why they cannot be used with equal success in instructional programs. Graphics are important to motivate the user, though they can also be counterproductive. We have found some programs with beautiful graphics that take a long time to paint on the screen; this causes the user to become passive, and the motivating aspect of the graphic is lost.

Again, long delays, especially in the middle of the program, cause a decrease in interactivity that reduces the effectiveness of the instructional program. To avoid this, we recommend using both interactive programming techniques and using computer languages with high speeds of operation. Turbo Pascal is one of these, while Microsoft BASIC is probably not. (New BASIC languages, such as BBC BASIC, are available that run more quickly, however.) DILEMMA uses a design that requires the user to respond frequently to computer queries; it is written in Turbo Pascal, and it tries to add bits of humor where possible. It is very weak on graphics, however, and this may cause some students to find it boring.

The content of instructional software must be sufficiently deep and/or abstract that it takes the student beyond the realm of common knowledge. One way to do this is to simulate some relatively complex phenomenon, such as the navigation of a supertanker in narrow waters, so that users can learn through their own trials and error about the pitfalls of an unfamiliar activity. Another way is to allow the student to interact with a model or simulation in a way that allows them to creatively explore their own ideas and strategies, while the program quickly performs the mathematical drudgery. A third way, used in DILEMMA, is to take a simplification of reality and allow the student to explore the ramifications of that simplification, using the interactive computer environment to speed the process of learning.

The fifth and sixth criteria are especially important for the use of instructional software in the university setting. In practice, these programs are often made available to students in libraries. They can be purchased on disks, but this is sometimes too expensive for students who are already laboring under heavy debts to attend college. In addition, few students can afford to spend more than an hour or an hour and a half using a piece of instructional software. We have found it to be more conducive to learning if users are allowed to interrupt the play and return to it later if the program requires lengthy playing times. In addition, the absence of copy protection appears to be *a sine qua non* for the depositing of disks in libraries. Most libraries will refuse responsibility for reimbursing instructors for copy-protected software that has been lost, stolen, or destroyed in use.

Finally, the user friendliness requirement, though a bit of a cliche, is violated often enough in instructional software to be noted here. By user friendly we mean not only easy to read instructions, abundant graphics, humor, etc. The most important requirement for instructional software is that it does not crash, even in response to `hostile' users. The program should be in a format that allows easy start-up instructions, e.g., "put the disk in the left disk drive and reboot." The input/output routines need to be able to handle all but the most serious user errors. Responses should be allowed either in small or upper-case letters, and should not involve more than one word if possible. Non-computer oriented students may become irate if the program will not work for them; they will either give up or take vengeance on the disk. This is another reason to avoid copy-protection. We had a case where the only copy of a copy-protected disk was accidently destroyed by a student, thus depriving the rest of the class from using the program.

We feel that DILEMMA adequately addresses most of these criteria, though we realize that there are many possible ways of improving the program. Some areas for possible improvement are: 1) allow users to play other games besides prisoner's dilemma, e.g., chicken, 2) put more graphics into the program (e.g., provide a graph of the cumulative scores of the two players after each play), 3) allow users to send "messages" to the computer opponent, 4) replace Joseph Stalin and Michael Marshmallow with more realistic opponents, and 5) allow users to play against other human opponents in a network of PCs. We hope to provide an updated version of DILEMMA that will include these improvements before the end of 1987.

General criteria for experimental software

To design a program for experimental use, the software writer should try to meet the needs of the specific experimenter. This makes a general experimental software package difficult to write. We offer the following suggestions for experimental software writers: 1) use standardized data formats for files that record user interactions, 2) allow the experimental conditions to be altered easily, 3) allow both open-ended and closed-ended responses to computer queries, 4) provide for internal testing of measurement validity and error-checking, and 5) (as always) make the program easy to use.

Because DILEMMA was primarily designed for instructional purposes, it does not score high on these criteria. However, our experience with

DILEMMA suggests that one may be better-off modifying simple programs for experimental use rather than creating very complex software which is designed to adapt to a wide variety of experimental conditions. For example, we just received an inquiry from a social psychologist who is interested in the motives or intentions attributed to machine-generated opponents in iterated PD games. This person would like to video-tape subjects while they are playing the iterative PD on an IBM-PC. Since the code and data structures of DILEMMA are very simple, it can be adapted easily to suit the needs of various experiments: human-human play could be created, the strategies of computer opponents could be altered, and the results which are written onto a separate disk file could be added to or changed completely.

One strategy, then, for designing an experimental program is to include the necessary basic structures along with lots of comments to make the program understandable to anyone who wants to adapt it to their needs. By basic structures we mean two things: 1) that the designer should create standardized input and output from disk files, and 2) that one should divide the "play" part of the program into discrete stages that are independent from each other, so that they may be easily added onto or omitted. These suggestions and those listed above may not apply to every experimental program, but they should be seriously considered in all designs. More and more experimenters will be using PCs and PC networks in their laboratories, and software will have to be written with this in mind.

Summary

This paper has described and analyzed the DILEMMA program, using this discussion to illustrate some general criteria that we feel are important for the success of instructional and educational software. DILEMMA has both instructional and experimental capabilities, though it has been tested so far predominantly in instructional settings. It is designed to teach students about a fairly specific but abstract type of situation, an iterated PD game. It demonstrates some very basic principles about how to do well in playing such games, while also providing experimenters with a way to obtain data upon which they might generalize about human behavior in the PD situation. DILEMMA remains a simple but sound educational program even by today's standards. Future educational software designers are suggested to consider the desireable features listed here, so that their programs will be more functional and adaptable as teaching and experimental tools.

Reference

Axelrod, R *(1984) The Evolution of Cooperation* New York: **Basic Books**

Software availability

DILEMMA costs US$22, and *is* available from: NCSU Software, *Box 8101,* North Carolina State University, Raleigh, NC 77695, USA.

Jeffrey Hart and Marc Simon
Department of Political Sciences
Indiana University
Bloomington
IN 47405, USA

ISAGA 88

The Conference of the International Simulation and Gaming Association will be held between 16-19 August 1988 in Utrecht, The Netherlands.

THEME: THE IMPROVEMENT OF COMPETENCE

in

- intercultural management and exchange
- policy formation and policy making
- delivering quality services in health care, crime prevention, education
- learning
- user oriented game design

For further information write to: ISAGA 88, Faculty of Social Sciences, P.O. Box 80.140, 3508 TC Utrecht, The Netherlands.

CONVERSATIONAL SIMULATION IN COMPUTER-ASSISTED LANGUAGE LEARNING: POTENTIAL AND REALITY

D. *Wells Coleman, University of Lodz, Poland*

Abstact: The potential of conversational simulations for computer-assisted language learning (CALL) is particularly clear. But this potential remains largely untapped. Various factors are responsible. The author identifies three major ones: (1) many CALL authors are waiting for a 'breakthrough' in artificial intelligence (Al) applications; (2) many CALL authors believe conversational simulations require natural language parsers of greater power than those needed for diagnosis of grammatical errors when, in fact, the *reverse* is true; (3) some CALL authors misunderstand the essential nature of conversational CALL as *simulation,* confusing 'realistic' natural language interaction with 'real'. The author discusses a group of programs which have been inaccurately referred to as conversational 'simulations', defines some criteria to clarify what kinds of CALL software should - and should not - be included in this category, and presents a few CALL conversational simulations as examples.

Keywords: Conversational simulation; computer-assisted language learning; parser; artificial intelligence.

Introduction

Many -computer-assisted instruction program authors recognize the potential of software which can engage in simulated student-computer conversation in natural language. The potential of such conversational simulations for computer-assisted language learning (CALL) is particularly clear (see, e.g., the discussions in Wyatt, 1984; Underwood, 1984). But this potential remains largely untapped. It would seem that most CALL authors are content to wait for a 'breakthrough' which will provide easy-to-use, large-scale artificial intelligence (AI) systems to work with. The most realistic prognosis is that they will have a rather long wait. Even the most optimistic predictions would suggest that the kind of large-scale Al capabilities in question are decades away, at least.

In addition, some authors lack the optimism of Wyatt (1984) and Underwood (1984). Compare, for example, Ahmad *et al.'s* (1985:52) rather bleak assessment of Winograd's (1972) SHRDLU with that of Wyatt (1984:99). Ahmad *et al* claim to find the content of SHRDLU's simulated conversations not 'interesting' (1985:52) and thereby seem willing to write off this whole class of endeavor. If the simulated world that SHRDLU can converse about is not interesting, then this is an