Homework 17: C++ Programming

Due:

Read Stroustrup's *The C++ Programming Language* for Background. There is a reference manual for C++ on reserve at the library for this course.

1 The Prisoner's Dilemma

Object-oriented programming techniques are touted as being good for simulation. In this problem you will investigate an iterated prisoner's dilemma, using simulation techniques and C++.

The prisoner's dilemma is a model of certain kinds of social interactions. The idea is that people are often faced with situations where they may either "cooperate" or "defect". As the standard example, consider two criminals, Bonnie and Clyde, who have been arrested on a minor charge but are suspected of committing a major crime. The police make the following offer to each: "if you confess and implicate your partner, then you will go free and your partner will get 5 years in jail, but if you do not confess, you will have to serve one year in jail on the minor charge. Oh by the way, if you both confess, you will each serve 3 years in jail." In this case cooperation (among the prisoners) means keeping silent, defection means implicating the partner. The police offer can be represented by the following payoff matrix:

Bonnie\Clyde	cooperate	defect
cooperate	(-1,-1)	(-5,0)
defect	(0,-5)	(-3,-3)

where the entry (0,-5) means that if Bonnie defects and Clyde cooperates, then Bonnie gets a jail term of 0 years, while Clyde gets 5 years in jail (-5). Each prisoner wants to maximize his or her payoff (i.e., each wants to get the fewest years in jail).

The iterated prisoner's dilemma is like the prisoner's dilemma, only repeated over many encounters. Imagine Bonnie and Clyde both cooperate, and after a year in jail, they get together again, rob more banks, and then wind up in the same situation. More to the point, imagine a world of agents who are continually interacting. For example, consider graduate students and their use of disk space. One may cooperate by deleting unused files, but this takes some time and effort. Hence there is a cost to cooperation. If others use disk space without deleting unneeded files (the defectors), you are just subsidizing their lethargy with your time! However, if everyone cooperates, then there will usually be plenty of disk space for all. Similar examples can be found in many social interactions, from driving a car to trade relations among nations.

1. Files containing C++ source code for the iterated prisoners dilemma are in the directory `~leavens/src/c++/prisoners/`. (There is also a noweb document there which you can look at if you wish more information about the program. You do not have to edit the noweb version, but can simply edit the C++ files.)

   For this part you are to extend the simulation of the iterated prisoner's dilemma, by adding and testing at least three (3) different, history-sensitive strategies for the behavior of prisoners to see which is best. A strategy is an algorithm for deciding whether to cooperate or defect, based on internal state such as a history of how the other agent has cooperated or defected in previous rounds, and some random possibilities, (using a random number generator). A strategy is *history-sensitive* if it does different things depending on what its partner did (i.e., it is not completely random or thick-headed).

   Include comments in your program to explain your strategies, and also show the output of test cases. Submit only the additional files you add to the program. You should not edit any of

the program itself, with the exception of the files `WorldMaker.C` and `PrisonerTypes.h`. You should submit an English explanation of your program and the strategy you found to be the best, along with your experimental results.

2. Let us call a run of the program in the preceding problem a *generation*. In this problem you will make another subtype of `World`, call it `EvolvingWorld` that "evolves" between generations. The idea is to compute a "fitness index" for each strategy, based on its winnings in the previous generation, and to populate the next generation so that the more fit a strategy is, the more descendents it has in the next generation. Your task is to make the program work with the `EvolvingWorld`. Run the simulation for, say 5 or 10, generations, and then print the results. (It may be helpful to define another class `TraceEvolvingWorld` that prints more debugging information about what happened at each generation.) You will have to come up with a suitable formula for determining the "fitness index" and for assigning strategies to the next generation. (Hint: figure out what strategy did the worst, and watch out for when they all do equally well.)

   What to edit and hand in is the same as for the previous problem. You won't need to use the files `StdWorld.h` and `StdWorld.C` for this problem.

2 Extra Credit

3. Compare the template mechanism in C++ to generics in Ada. What are the advantages of each? How could the C++ mechanism be improved?

4. Compare the list class in LEDA to the `CookList` files discussed in class. What are the differences in how they are used? Which is more convenient and why?

5. In the simulation of the iterated prisoner's dilemma, perfect communication is assumed. Suppose that sometimes, with a fairly small probability, one prisoner doesn't get correct information about what the other decided. Can you simulate that? What happens?

6. Design a library of classes for C++ and formally specify it in Larch/C++ (see Gary for details).

7. C++ is a big language. How could it be simplified?