This code was used to generate the comparison of algorithms reported in the paper:

J. W. Crandall et al., Cooperating with Machines, *Nature Communications*, 2018.

Note that detailed results are included in Supplementary Note 3 of that paper. I'm happy for anyone to use this code if it can be of any use to them. If you do use this code to assist you in any research, I would appreciate it if you would cite the above paper.

Below is a little information to help you get started with the code, which is written in C++ for Mac. I can't vouch for how it will behavior on other systems. I apologize that it isn't perhaps in the most readable state (I have a habit of not commenting in my code, as I don't think of other people perhaps using it).

The basics are as follows:
(1) The main code to pair two algorithms together in repeated games can be compiled using the makefile in the "++" folder.

(2) Once compiled, the code can be run as specified in the comment just before main in main.cpp. For example, to run a 1000-round prisoners dilemma between S++ and MBRL-1 (br1), you would type:

The round-by-round results are printed out in the "log" folder (actions taken by both players followed by the payoffs each received), while a summary of the results is printed out in the "results" folder.

(3) You can create new matrix games using the format of the other matrix games (you should be able to figure out by looking at the other games in the "games" folder. However, you should make sure that payoffs are normalized between 0 and 1, as I believe that I made this assumption when coding up some of the algorithms.

(4) In our paper, we essentially compared 25 different algorithms. The code provided includes more than 25 algorithms. You can infer the algorithms we used in our comparison by looking at the "nameAgents()" function in the file batch.cpp.

(5) You can play around with different parameter settings for the algorithms. The settings I used are reported in Supplementary Note 3 in the above cited paper.

(6) The "games" folder specifies the payoff matrices of the various games. Again, if you decide to test the code with new games, please ensure that the payoffs of the game are normalized between 0 and 1, as the implementation assumes this to be the case for some algorithms. If you try the algorithms out with payoff matrices that do not satisfy this constraint, I can't vouch for the algorithms working correctly.

(7) The "memory_one" and "memory_two" folders contain information to get the algorithms "memory1" (Mem1) and "memory2" (Mem2) to work correctly for each game we used in our comparison of algorithms. Note that these two algorithms require the memory1 and memory2 strategies to be evolved. I've already evolved (and saved) these strategies for many of the games, but if you try to run it on one that hasn't been evolved, the system is set up to compute it automatically (which takes a few minutes) as long as you have compiled the code in the folder

StochasticMemory using the make file first. Note that if you change the payoff matrix in an existing game, without re-evolving the strategy, the new strategies won't work. Thus, if you change the payoff matrix of the game, you' should delete the corresponding file in the Memory1 and Memory2 directories, respectively.

If you really would like to use the code but are having a hard time getting it to run (or something seems strange when you do run it), please contact me at crandall@cs.byu.edu. I'd be happy to help (though I can't promise I'll be able to respond right away).

Jacob Crandall
January 2018