Game Programming Project
Based on introduction by Prof. Betty O'Neil.
Prof. Ethan Bolker wrote the Game Package
CS310 Spring 2011: pa4 (intro) and pa5

1. Introduction.

The object of the Game project (intro in pa4, then all of pa5) is to explore various algorithms for choosing the best move in a two person game. The game for which you are choosing moves is encapsulated in a Game object whose implementation is unknown to your algorithm: you write pure application code, playing the game by calling Game API primitives without even knowing what game is being played.

The Game API captures the essence of games like tic-tac-toe, checkers or nim which seem on the surface to have little in common. Thus in this project we see a wider range of implementations that satisfy the publicly advertised interface than in our previous collection class examples - we tend to think that maps are more alike than are games. Here the power of thinking in terms of APIs does more than hide the implementation of some useful programming concept. It shows us that we can write one game-playing algorithm to play any of these games precisely because we have identified and captured the critical properties they share in the design of the Game API.

The games that fit the Game model described by the Game API share the following properties. Each is played by two players, henceforth called 1 and 2. At any time during play it is possible to specify the current position, to see whether the game is over and who has won, and, if not, determine whose turn it is and which of the possible moves is legal for that player at that time. We assume that Players usually take turns moving but in some games the rules sometimes allow a player to move twice or more.

No player has information unavailable to other players (no cards held in your hand). No randomness is allowed during play (no dice), although the initial position may be set randomly and told to both players.

We have provided implementations for some simple games: tic tac toe, easy, 15, game0, putnam, nim, c-m-n, and sticks. We also provide a driver that allows a user to play any of the games interactively against the computer by entering moves at the terminal. For tic-tac-toe only, we have a GUI representation of the game playing, once the game is started.

2. The Game Tree

The game starts in a certain position. Then the first player makes a move, one of the moves that are available with that game position, or “game state.” Each such move brings to game to another state. Then further moves make the game states branch out further. The whole configuration of game states caused by moves from the original state is called the game tree. It is like a recursive function’s call tree in that it is a figment of computation or “what-if” thinking, not a data structure in memory. The idea of the game tree is basic to computer game-playing. The computer can analyze a huge game tree and see what could happen whatever the other player does. By analyzing many what-if scenarios, it can choose the best move.
In pa4, we are not yet trying to find the best move. We are just exploring the game tree. In pa5, we will tackle the backtracking best-move search.

3. The PlayOneGame Class

The heart of the interactive driver is the loop over moves by the computer and the human:

```
while the game is not over
  call findbest for current player, where
    human-findbest: get a move from the terminal
    computer-findbest: search to find the best move
  make the move
```

Thus the human's opponent is the function findbest (for the computer-as-player) called by PlayOneGame when it's the computer's turn to move. The computer's findbest is the function you will write in pa5. We've given you a stub file BackTrack.java, which now has code that calls on the human to choose the computer's move. So when you run PlayGame with this supplied faked-up findbest you will be playing against yourself.
To run the Easy game, see $cs310/games/readme.txt. Once you have the basic setup in place and have run Easy successfully, you can try other games:

```
java cs310.PlayGame <game> <player1> <player2>
```

where <game> is Easy or Tictactoe or Game0 or any of the games in the package. A <player> can be "human" or "cs310.Backtrack" or some other algorithm. Until you have written some implementations of findbest algorithms the only algorithm you can use is cs310.Backtrack. So the important forms at the start are:

```
java cs310.PlayGame <game> human cs310.Backtrack
java cs310.PlayGame <game> cs310.Backtrack human
```

For convenience, the first command above can be shortened to "java cs310.PlayGame <game>".

Tictactoe GUI: in the case of <game> being "Tictactoe", you can run a GUI version with the command.

```
java cs310.PlayGame –cg Tictactoe
```

Don't use this if you're using a ssh login to run Java! But if you are running on a local PC or local UNIX/Linux/MacOX system, it should work fine.

4. The Game API
The PlayGame driver can play games whose rules it does not know because all games implement the Game API. You will learn that API quite well in the next week or two as you construct various implementations of the function findbest() that the computer uses to choose its move.
See readme.txt in the games top-level directory for a guide to source files to look at, from the most important on down.