Yale University
CPSC 474/574 - Computational Intelligence for Games
Fall 2020

Assignment 2: Dynamic Programming

Objectives

- to solve a probabilistic game using backwards induction (dynamic programming)

Introduction

Shut the Box is a dice game for two or more players. The game is played with tiles numbered 1 through n that are initially all in an upright, "open" position. Each player takes a turn trying to close the tiles by rolling the dice and then closing an open set of tiles whose sum equals the sum on the dice. For example, if the open tiles are $\{1, 2, 3, 4, 7\}$ and the roll total is 7, then the player has three choices: close the 7; close the 4 and 3; and close the 4, 2, and 1. Each player keeps rolling until they close all the tiles or are unable to make a sum equalling the roll total using the open tiles (for example, if the open tiles are $\{1, 2, 4, 9\}$ and the roll total is 8). If a player's turn ends with all tiles closed ("shutting the box"), then that player wins immediately – the other players do not get to take a turn. If a player's turn ends with some tiles open, then that player's score is the sum of the open tiles and the next player starts their turn with all tiles open. If no player shuts the box after all players have taken their turns, then the player with the lowest score wins (for the last player, the game ends as soon as they have beaten the other players' scores).

There are many variations of Shut the Box with different n, and different rules for determining how many dice a player rolls in each position. We will consider only the two-player game with $n = 9$, and with players rolling two fair six-sided dice when the sum of the open tiles is strictly greater than 6 and one die otherwise.

Assignment

Write a program called `ShutTheBox` that calculates the expected number of wins for a given player, assuming optimal play by each player starting from a given position and counting a tie as half a win for each player, and also determines the optimal move (which numbers to close) given a position and roll.

The parameters for a given run will be given as command-line arguments as follows.

- The first command-line argument will be either `--one` or `--two` to indicate whether to solve for player one's expected number of wins or optimal move or for player two's.
- The second command-line argument will be either `--expect` or `--move` to indicate whether to calculate the expected wins for the player specified by the first argument or that player's optimal move.
- The position will be given as the third argument and will be a string of unique, increasing digits in the range 1 through 9 indicating which numbers are still open.
- For `--two`, the fourth argument will be a decimal integer specifying player 1's score.
- For `--move` there will be an additional argument (fourth or fifth for `--one` or `--two` respectively) following the position giving the sum of the roll to determine the move for.

All input will be as specified, and the given position will be nonterminal, there will be valid moves for the roll for `--move`, and player one's score will be possible (for example, it is impossible for player one's turn to end with 45 points because they would have had a valid move for any roll).

Your program should be able to perform any computation in no more than a few seconds.

Output

Expected wins should be printed to standard output with 6 digits after the decimal point, with the last digit rounded to the nearest value (so you can use the `%.6f` format specifier for functions that take `printf`-style formats). Optimal moves should be printed to standard output as a comma-separated, strictly increasing list of tiles to shut, delimited by square brackets. If there is more than one optimal move then you may choose which one to output arbitrarily, and the test cases will accept any optimal move.

Testing

Although not required by the specification, you may find it easier to test if you generalize your code to allow for versions of the game with different ranges of numbers on the boxes and dice with different numbers of sides, since you can then test on small games that require fewer calculations to solve by hand.

Examples

```
$ ./ShutTheBox --one --expect 123456789
0.502810
$ ./ShutTheBox --one --expect 146789
0.256254
$ ./ShutTheBox --one --move 146789 9
[9]
$ ./ShutTheBox --two --expect 123456789 8
0.381212
$ ./ShutTheBox --two --expect 12345689 41
1.000000
$ ./ShutTheBox --two --expect 13456789 43
0.986111
$ ./ShutTheBox --two --move 13456789 17 12
[3, 9]
```

Submissions

Submit any necessary code, your time log, and a makefile with default target `ShutTheBox`.