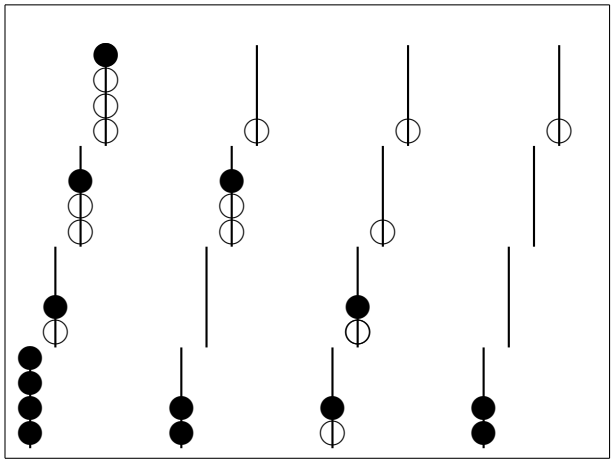

Figure 1: Lines in a Score Four game.

The board shown to the left has four black lines and two white ones. The position shown could never arise in a real game.

Problem statement

The goal of this project is to write programs to play a 3-dimensional tic-tac-toe game that is sold commercially as "Score Four".[1]

The Game Board Score Four is played on a board which consists of a 4×4 grid of pegs (thin metal spikes—see, for instance, https://flickriver.com/photos/acesfinds/6266498896/). On each spike you can slide as many as four coloured beads. Beads cannot hang in the air; they slide down to the bottom of the peg. This makes Score Four different from $4 \times 4 \times 4$ tic-tac-toe. There are two colours: White and Black. and 32 beads of each colour.

Lines The object of Score Four is to get four beads of one colour in a straight line. A straight line can be horizontal, vertical, or diagonal. Diagonals can be singly or doubly "skewed" as shown in Figure 1. If you count carefully, you should find that there are 76 lines on a completely-filled board.[2]

Playing a Game Score Four is a game played between two players. Players choose by some random method which player gets to play White, and which player gets to play Black. White plays first. White has an advantage, so when playing mulitple games in a row, the previous loser usually gets to play white.

The players then alternate taking turns placing beads. On each player's turn, the player places a bead of their colour on one of the 16 pegs that is not yet full. A player *must* play a bead on their turn.

The first player to get four beads in a straight line wins.[3] Should the entire board be filled before either player completes a line, the game is a draw.

[1] see `http://www.boardgamegeek.com/game/3656`. It appears that this game was first produced in the 1970's, so patents have likely expired.

[2] See the Appendix for more discussion of the mathematics of lines.

[3] Played between two humans, there are some times additional complicated rules about a player needing to realize that they have won.

Program specifications

Your team needs to create

- for distribution to other students, an executable `.jar`-file (see below),
- for submission to the instructor, a `.tar` (or `.jar`) file containing
  - your source code (`.java` files),
  - `javadoc` documentation for your program(s),
  - an executable `.jar`-file (see below),
- A brief guide that explains how to run your program as a game.

The executable `jar` file

One submitted version of the project must be an executable `.jar`-file. It need not contain `.java` files, but should be self-contained and easy to run.

Coding Requirements

Your team's program(s) must be written in JAVA, and must run on a standard JAVA installation. If you intend to use downloaded additional graphics libraries, or advanced JAVA features, consult with the instructor first.

Code must generally conform to the coding style specifications in Appendix A of the textbook *Big Java*: *Early Objects*. Additional coding style requirements will be specified in a future document.

Code must contain `javadoc`-style comments for packages, classes, public-, protected-, and package-level methods. All member variables must be private.

All code lines must be at most 80 characters long, and MUST NOT contain tabs.

Unlike previous versions of this project, Your team needs to write *one* program that runs in multiple modes.

The Computer AI The computer AI is a subsystem of your program that can propose a move in a given board situation.

In interactive game-playing mode, the computer AI can be harnessed to provide a computer opponent that a human user can play against. However, the computer AI must not make assumptions about about the board situations it is given.

Testing mode Testing mode serves multiple purposes:

- It allows for program development and testing;
- It allows other student reviewers to test functionality in a controlled manner;
- It gives the instructor an easy path to creating a tournament where your program plays against other student programs.

Testing mode Commands Each command ends with a period (.). The commands are:

1. "**clear.**". Empties the current board.
2. "**quit.**". The program should gracefully quit.

```
A1: BBBB
A2: BB
A3: WB
A4: BB
B1: WB
B2:
B3: WB
B4: BB
C1: WWB
C2: WWB
C3: W
C4:
D1: WWWB
D2: W
D3: W
D4: W
```

Figure 2: Output from "show board." commend

3. Commands that match the pattern "**add *colour* bead to *B3*.**" Here "***colour***" can be either "black" or "white", and "***B3***" can be any of the 16 possible locations. A location consists of a letter followed by a number. for instance, "**B1**". The letter is '**A**', '**B**', '**C**' or '**D**'; and the number is '**1**', '**2**', '**3**', or '**4**'. Like other commands, the last character is ".".

   If it is possible to add the specified colour bead to the current board position at the specified location, the program should do so, and respond "**Done.**" Otherwise, it should respnod "**Impossible.**" and leave the board position unchanged.

4. Commands that match the pattern "**remove bead from *B3*.**" Here "**B3**" can be any of the 16 possible locations.

   If it is possible to remove a bead to the current board position from the specified location, the program should do so, and respond "**Done.**" Otherwise, it should respnod "**Impossible.**" and leave the board position unchanged.

5. "**get black move.**". "**get white move.**". The computer AI should propose a move for either black or white as requested. This command should *not* cause the board to be updated. The response should normally be a location consisting of a letter followed by a number followed by a period, for instance, "**B1.**". The letter is '**A**', '**B**', '**C**' or '**D**'; and the number is '**1**', '**2**', '**3**', or '**4**'. Alternatively, the computer AI can respond "**Impossible.**", if, for instance, the board is full.

6. "**show board.**"

   This command produces output suitable for input into another computer program for analysis. For the (impossible) board situation shown in Figure 1, the output should be as shown in Figure 2.

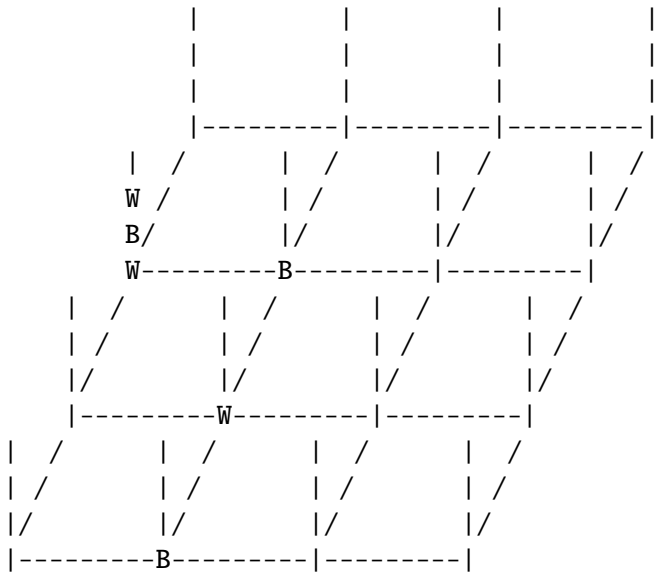

Figure 3: Possible output from "draw board." commend

7. "**draw board.**"

   This command produces output suitable for human understanding. There is no precise specification, but output should look something like Figure 3.

8. "**go interactive.**". "**go gui.**". These commands should cause the program to switch to a mode more suitable enjoyable game playing.

9. Other commands. Feel free to enrich this command set. However, the commands above must be implemented, and where responses are indicated, your program responses must exactly mactch the specified format.

Interactive mode In interactive mode (either ASCII or GUI based) it should be possible for a human player and a computer opponent to play an interactive game of Score Four. The human player should be able to choose which colour she wants, and the human player should be able to quit at any time.

General comments

For the computer opponent program, correctness is far more important than cleverness. The computer AI must work correctly, and the program produce correct respnoses to testing-mode commands, even when used by a referee program other than your own. However, intelligent play by the computer opponent is not necessary, and should not be a priority when completing the project.

David Casperson
2024-01-21

A The Mathematics of straight lines

We can generalize from a $4 \times 4 \times 4$ board to an $n \times n \times n$ board, and we can generalize from 3 dimensions to k dimensions, and imagine a $n \times \cdots \times n$ or n^k board.

An n^k-board has $\left((n+2)^k - n^k\right)/2$ lines. In particular, 1-dimensional boards have exactly one line, making them boring.

Two dimensional boards have straight and diagonal lines; three dimensional boards have straight, diagonal, and skew-diagonal lines. In general, k-dimensional boards have k different kinds of lines. We'll say that a line in a k-dimensional board is *d-skew* if $k - d$ co-ordinates of the points in the line don't change. For instance, in a 3×3 board, the line $[(1,2),(2,2),(3,2)]$ is 1-skew;, and the line $[(1,3),(2,2),(3,1)]$ is 2-skew.

On an n^k board d-skew lines run in $2^{d-1}\binom{k}{d}$ directions.

For instance, on a 4^3-board, there are three directions for 1-skew (*straight*) lines: N-S, E-W, and ↑-↓ ($3 = 2^{1-1}\binom{3}{1}$). There are six directions for 2-skew (*diagonal*) lines: NE-SW, SE-NW, ↑E-↓W, ↑W-↓E, ↑N-↓N, and ↑S-↓N (note that $6 = 2^{2-1}\binom{3}{2}$). Finally, there are four ($4 = 2^{3-1}\binom{3}{3}$) directions for 3-skew (*skew diagonal*) lines: NE↓-SW↑, NW↓-SE↑, NE↑-SW↓, and NW↑-SE↓.

In each of $2^{d-1}\binom{k}{d}$ directions that d-skew line can run on an n^k board, there are n^{k-d} lines. For instance, on a 4^3 board we have

Skewness		Directions	Lines per direction	Total
Straight	1	3	16	48
Diagonal	2	6	4	24
Skew-diagonal	3	4	1	4
				76