GamesCrafters

Odd or Even?

Pai-Hsien Peter Yu (Developer)

Documentation for Developers

Fall 2007: Moonway Lin (Documentation)

Date	Version	Notes
2007.12.05	1.0.1	First revision
2007.11.27	1.0.0	First version created

Table of Contents

Gamescrafters Documentation: Odd or Even

0. Game Overview

Odd or Even is a game in which players alternate removing 1, 2, or 3 matches from an initial pile of 15 until there are no more matches left. The player with an even number of matches wins.

Odd or Even is a **non-loopy**, **non-partizan game**, and **ties are not possible**.

1. Design Overview

The script constructs a game tree to find the optimal sequence of moves for the player to win with an even number of matches.

The initialization function `initializeGame()` prepares the game for execution by initializing the necessary starting variables, such as the initial number of matches, the number of matches that can be removed per move, et cetera.

`GenerateMoves()` creates a linked list of every move that can be reached from a given position, then returns a pointer to the head of the linked list.

`DoMove()` applies the move by changing the position string so that it is consistent with the new board after the move.

2. Data Structures

There are two data structures, both integers:

The board's current **`position` integer** has the following format:

1234567, where

- 1 = the current player's turn (P1 or P2); in this case, it's P1
- 23 = the number of matches left on the game board
- 45 = the number of matches P1 has
- 67 = the number of matches P2 has

An example of a legitimate position string is

2120300, which corresponds to a game board in which P1 has just removed 3 matches from the initial 15 matches and P2 is set to go next.

The board's current **`move` integer** has the following format:

103, where

- 1 = the player number (P1 or P2)
- 03 = the number of matches removed

To obtain the appropriate values from either data structure, we take the position or move integers and modulo 100 or 10000, depending on the information we wish to extract.

3. C Text Game Checklist

There are no help strings that change with variant. One of the help strings, `kHelpGraphicInterface`, has not been written yet. Debug is NOT turned off.

There are no memory leaks.

`PrintPosition()` works fine, printing out the number of matches left and those owned by P1 and P2.

The code is clean and well-commented. Helper functions include descriptions above their implementation.

`MoveToString()` has been implemented.

There are calls to common functions in `GetAndPrintPlayersMove()`, which calls `HandleDefaultTextInput()`, and `PrintPosition`, which calls `Unhash()` and `GetPrediction()`.

There are no symmetries and no GPS.

4. Complexity Analysis

The exact number of games possible can be computed via a game tree search. It is known that at least at the top levels, the game tree grows exponentially of order n^3, while the depth is approximately equal to the logarithm base 3 of 15.

In Gamesman, the number of positions is estimated to be 3000000, with no exact count.

5. Optimal Strategy

The optimal Odd or Even strategy is uncovered via game tree search, and allows Player 1 to force a win via an optimal sequence of moves that ensures an even

number of matches at the end of the game.