Threes!

Threes! is a recent abstract board game available on the internet and for download (and possibly elsewhere?). The rules of *Threes!* **that we shall use in this project** are as follows, although it's simpler than it sounds here! Example moves are shown in the next section.

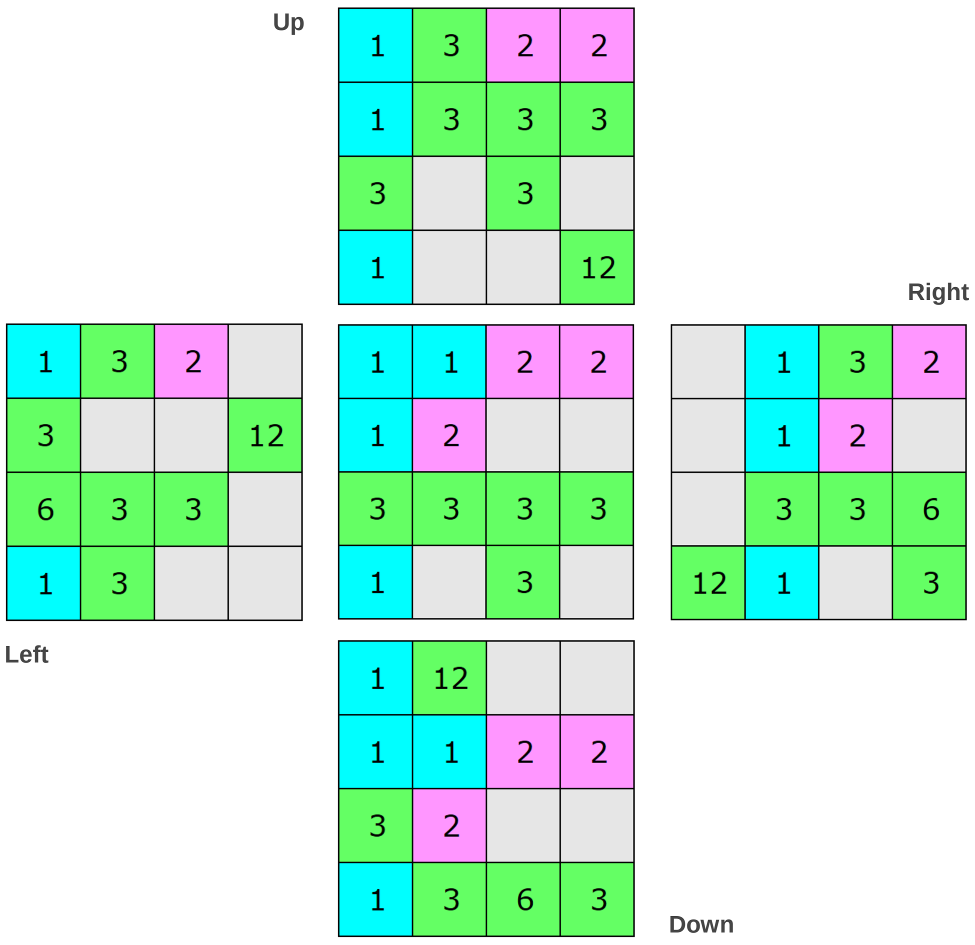
1. *Threes!* is played on a 4x4 board. At any given moment, each square on the board is either empty, or it holds a tile bearing an integer in {1,2} U {3 * 2^k | k in Z}, i.e. {1, 2, 3, 6, 12, 24, 48, ...}.
2. The starting position is a non-empty board plus a sequence of tiles. Most tiles both on the starting board and in the sequence will be low numbers. The player makes a sequence of moves, each of which consumes one tile from the sequence.
3. Each move is one of LRUD, i.e. Left, Right, Up, or Down. The description below is for Left moves - hopefully you can translate for the other three possibilities.
    - A Left move does the following.
        - all rows on the board attempt to shift to the left;
        - if all rows fail to shift, the game terminates immediately;
        - if one or more rows successfully shift, the next tile in the sequence is placed on the rightmost square of one of them.
    - A row's attempt to shift to the left will be successful if any of the following slides is available:
        - a tile can slide left into an empty square;
        - two adjacent tiles hold 1 & 2 in either order, when the rightmost of them can slide left (into the other one's square) and become a 3;
        - two adjacent tiles hold the same number x > 2, when the rightmost of them can slide left (into the other one's square) and become a 2x.
    - If any slides are available in a row:
        - that row's shift is deemed successful;
        - the leftmost possible slide on that row is performed;
        - all tiles to the right of the sliding tile are slid to their left.
    - Note that a row which has x tiles at its left (0 <= x <= 4) without any combinations, and 4-x empty squares at its right, cannot be shifted. This includes (as special cases) empty rows (x = 0), and full rows (x = 4) with no combinations.
    - After a successful move, the row which receives the next tile from the sequence is chosen as follows. (Examples and more details are given below.)
        - the rows which successfully shift to the left are given a score which is their list of tiles after the shift, read from right-to-left;
        - the row with the lowest score lexicographically receives the next piece from the sequence at its rightmost square;
        - if two or more rows have the same (lowest) score, the lowest of them on the board receives the new tile (in general, the "most clockwise" row/column).
4. The game is over if the player attempts a move which does not successfully shift any row/column, or if the sequence of tiles or moves is exhausted. The score for the game is the sum of the scores for the individual tiles on the final board, using the formula:

    score(empty) = 0
    score(1) = score(2) = 1
    score(x) = 3 ^ (log2(x / 3) + 1), x > 2
    Bigger scores are better.

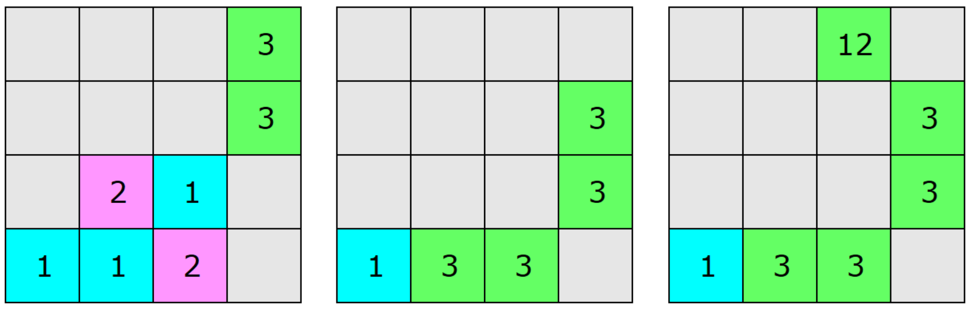
Your task in this project is to write a program that can play *Threes!*.

Example moves

Each of the moves in the first example starts from the board in the centre: the new tile introduced in each case is the 12.



The second example illustrates the rule that determines which row/column receives the next tile from the sequence. Again the new tile is the 12. From the board on the left, the player does a Down move, giving the board in the middle. The first column is not shifted, so we discount that possibility. For each of the others, we list the column downwards, i.e. the two middle columns are each [0,0,0,3], and the rightmost column is [0,3,3,0]. The first two values tie as the smallest lexicographically: to break this tie, we choose the "most clockwise" of the columns. For Down, this is the rightmost of the two, and the 12 appears there. (For Up, the most clockwise is the leftmost column; for Left, it is the lowest row; for Right, it is the highest row.)



Understanding where new tiles appear is probably the most complicated bit of the game mechanics in *Threes!*. Make sure you understand the five placements above: also in the second example, starting from the board on the left, after an Up move, the new tile would appear in the leftmost column; and after a Left or a Right move, the new tile would appear in the third row down.

Representations and files

An input file will have eight or more lines.

- Lines 1-2 will contain any comments about the file: they should be ignored by your program.
- Lines 3-6 will each have four tiles (or empties), separated by spaces: they represent the initial state of the board.
- Line 7 will be blank: it should be ignored by your program.
- Lines 8 and after will contain the tiles in the sequence that your program must process, separated by spaces.

An example input file is available. Note that a realistic input file will have hundreds (or thousands?) of tiles: the intention is that there will be more tiles than the player can process successfully.

An output file should have three or more lines.

- Lines 1-2 can contain anything: they are intended for any comments you want to attach, and will be ignored.
- Lines 3 and after should contain a sequence of characters in {L,R,U,D}. All characters not in this set will be ignored.

An example output file is available.

Submission

You are required to construct a player for the version of *Threes!* described here. Your program should read an input file specifying the initial board and the sequence of tiles, and it should output a file specifying its moves. It should choose a sequence of moves that generates the highest score possible.

- Aim for a system that produces maybe 5-10 moves per second.
- More example data will be available from here in due course.

You must submit two deliverables via cssubmit:

- the code you generate in the project, as a zip archive; and
- a paper describing and analysing your work, as a PDF file.

The paper should be about 5-10 pages long. It should describe

- the structure and design of your algorithm, and
- any interesting implementation details or strategies, and
- the experimental and theoretical analysis that you have performed.

If you use or refer to other people's work, make sure that you give all due credit.

Analysis and assistance

I may offer assistance of various kinds at some stages of the project. Such assistance will be announced via *help3001*, or on this page. On the following list, more-recent additions are nearer the top.

1. Consider carefully what type of algorithm to use, before you start any coding. Is an exhaustive search feasible? Is there an opponent to interfere with our planning? Answering these and similar questions will allow you to make intelligent design choices.
2. Here is a Haskell program that takes an input file and an output file, and generates a (usually large!) set of svg files that illustrate the transition of the board through the game.
3. Here is a Python program that you can use to create input files for various problem instances.
4. Here is an archive containing an input file, an output file, and the SVG files which display the game played.

Assessment

The project is designed to be done in pairs. No larger groups will be allowed, but you may do a solo project if you prefer. You may decide how to distribute the various project tasks (for example programming, testing, collecting empirical data, producing plots, writing, etc) between the group members, but the marks will be divided evenly, so the workload should be roughly equal too.

The project counts for 30% of your mark for CITS3001. Assessment will be based on

- how well you have used the algorithms that we have studied,
- how well you have analysed the problem,
- how well you have analysed your algorithm and program,
- how well your program plays *Threes!*,
- how well your paper is written,
- and anything else that occurs to me at the time.

You should consult the School's guidelines on plagiarism, available from the unit outline.

Deadline

The project is due by **4pm on Friday 30 May 2014**.
Standard penalties will apply for late submission, as specified in the unit outline.

Prize competition

There will be a prize for the best-performing *Threes!* program submitted as part of the main project. Details will be announced in due course. **Lyndon's decisions are final.**

Please don't neglect your exams or other assessment to put time into this competition.

Taken From: http://undergraduate.csse.uwa.edu.au/units/CITS3001/project/