**TicTacToe algorithm explanation**

**Sokolov Maxim, Marat Shikhamov**

**Input:** game TicTacToe with cube 4x4x4.

**Output:** create agent which moves as optimally as possible.

**Solution:** For full explanation, we should explain our chair of things.

All complexities are shown in Big O notation with some approximations.

1. First idea was minimax in 3d, but complexity in this case is 64!
2. Alpha beta pruning will down complexity to 64! - 2\*64^32=64!-2\*2^(6\*32)=64!-2^193 = 64! - 10^19 = 10^60 - 10^19 = 10^19(10^41-1)=10^60 - very huge again.
3. After it, we though about decomposition. By this way, a 3d cube can be decompose on several 2d fields. In optimal way, it are 12 fields (4 horizontal + 4 vertical + 4 diagonal). All other field will be combination of previous 12.

Of course, all of it are connected. If a player moves in (x,y,z) than this move will appear on several fields simalteneously.

But several the same (in structure) fields allows to decrease complexity of algorithm. This approach allows to solve this task on 2d field and use these solutions for 3d, but with several changes.

If 2d field solution has utility approach (every step has utility which can be compared for optimal move finding) than in 3d field solution an algorithm should compare list of utilities (12 in this approach). For it, algorithm should has comparison method.

So any algorithm will be described by local utility (utility function in 2d field) and utility comparison method (function for comparison list of utilities) (compare different 3d moves).

1. But after it, we thought: “Maybe we can decompose deeper?”. And answer yes, we can maximally decompose 3d cube on several lines (4x1). We can decompose on 1d points, because we couldn’t detect win/lose in this case. So our solution is following:
   1. We decompose 3d cube on 76 lines
   2. We defined utility function for line as

0 if count(‘X’)>0 and count(‘O’)>0 (in this case nobody win/lose on the line)

count(‘X’)/len(line) if count(‘O’)>0 (probability of win for ‘X’)

count(‘Y’)/len(line) if count(‘X’)>0 (the same)

* 1. In program, firstly, we creates root node of tree with input field (we can start from non empty field)
  2. Secondly, we run BFS which calculates vector of utility for every possible points (at the start 64). But if runtime of calculation more than TIME\_LIMIT constant than the program stop and store MAX\_DEPTH (depth of last node)
  3. In remaining time, the program runs tree traversal for calculating optimal way by precalculated utilities.
  4. For every possible point the program has vector of utilities (utility for every line). And comparison of lines based on comparison of vectors. Firstly, the program calculates hast of utilities = arg(utility\_line). So comparison of utilities is comparison of hashes (more or less depends on who plays).
  5. Finally, we have optimal probability and optimal move.