Assignment 1

### Implemented by Kabakov Mikhail

This document describes the AI made for the TIC-TAC-TOE game.

TIC-TAC-TOE is a fully observable, deterministic, static, but multi-agent environment. I also assume that it is episodic – because, although our previous decisions do affect future states – the game does not afford much planning to be done. And, more importantly – it is not required to play it optimally (i.e. never lose the game).

I have added the rule that <X> always start first (like the white in chess), but the game is built that the AI plays as <X> or <O> at random.

### The algorithm

On the highest level – the algorithm consists of 3 steps:

1. It determines if it’s possible to win the game immediately – i.e. whether there exists a line of 2 marks that just needs one more mark to form a line of 3. If one is found – it goes for it and the game is won.
2. If step 1 does not find such a mark – the algorithms checks if the opponent has such an opportunity to win in the next step – and blocks it, if one is found.  
   *It should be noted that no more than one such opportunity may arise in case of optimal play from the beginning of the game, so the algorithm stops search and blocks the first one as soon as it finds one. If, theoretically, more than one opportunities would appear – it would mean that the game is lost, but even so closing the first one would be as good (or, rather, as useless) as closing any other.*
3. If no immediate win opportunity, nor immediate threat was found – the AI runs the “makeBestMove” method that does the following:
   1. If the central square is free – put a mark there! This is because the central square can be used in the most number of ways to form a 3-line (4 possible lines include it).
   2. If the central square is already occupied – it evaluates each of the three horizontal rows, vertical columns, and both diagonals – to determine the possibility to form a 3-line on each of them. If a “friendly” mark is found – the weight of that line is increased by one. If an opponent’s mark is found – decreased by 10.
   3. Each of the squares is evaluated based on the sum weight of the lines that go through it. If one of the squares gets a score is -18 or less – it means there is a “double-threat” at that square (i.e. if an opponent puts a mark there – he will get two simultaneous 2-lines – and can consequentially win). The AI blocks such squares immediately.
   4. If none of the previous conditions have triggered – a mark is added to one of the squares that adds to one of the lines that has best chances of becoming a 3-line (if the opponent will make a mistake, of course). Basically, this means that the algorithm avoids putting marks that would help forming lines that are already blocked. I.e. in the situation below

| | (the AI plays as <X> and it is X’s turn)

---+---+---

O | X | 🡨 the medium-right square is a bad place to place an <X>.

---+---+---

| |

| | O

---+---+---

O | X |

---+---+---

X | | 🡨 putting an <X> here will create 2 immediate possibilities of making a 3-line (this became possible because putting an <O> on the left was a mistake the player made).

So, the AI should be classified as either a simple reflex agent (with the reflex function including computing and testing certain conditions, and a bit of searching for a maximum if they are not met), or a model-based reflex agent (with a very simple model of the world that is rebuilt from scratch during every move, and is actually larger than the digital representation of the persistent game world). However, it plays optimally: it never loses, and wins, as far as I’ve concluded – whenever it is possible.

The used algorithm is specific for the tic-tac-toe game, because given the simplicity of the game I decided it would be an overkill to implement a general algorithm that is used in more advanced games. But tic-tac-toe is rather simple, and it’s impossible to win the game, unless the opponent makes a mistake. In fact, only the first 3 moves (<X>, <O> and <X> again) have some interesting options – after this the game is about the <O> having to block the <X> from forming either a 3-line, or two simultaneous 2-lines. This leads to either a tie, if <O> plays correctly, or to the win of <X> - if <O> makes a mistake.

Such approach is suitable for environments where there is a very limited number of possible reasonable choices, especially if a significant amount of the moves are constrained (i.e. there is only one thing to do). It works well in environments that just require detecting one of a very limited number of patterns – and responding to it in the pretty much single possible way.