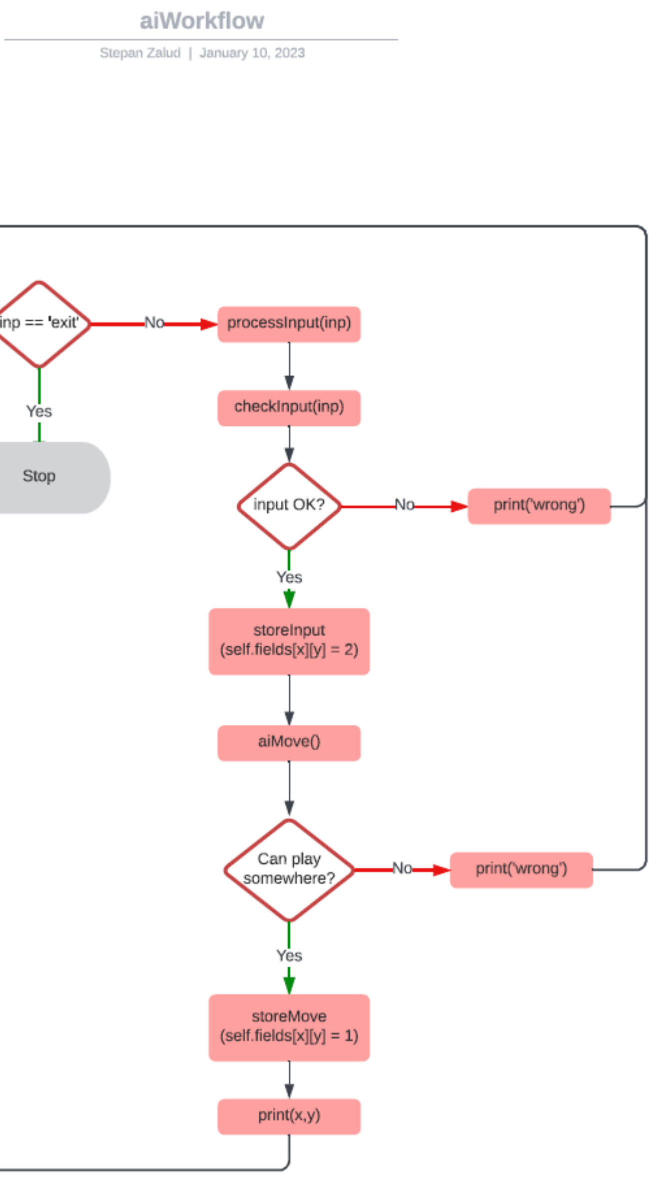
AI documentation

This is a documentation of solution for simple algorithm that serves a purpose of opponent for TIC-TAC-TOE game.

Main workflow of the program can be observed in following flowchart.

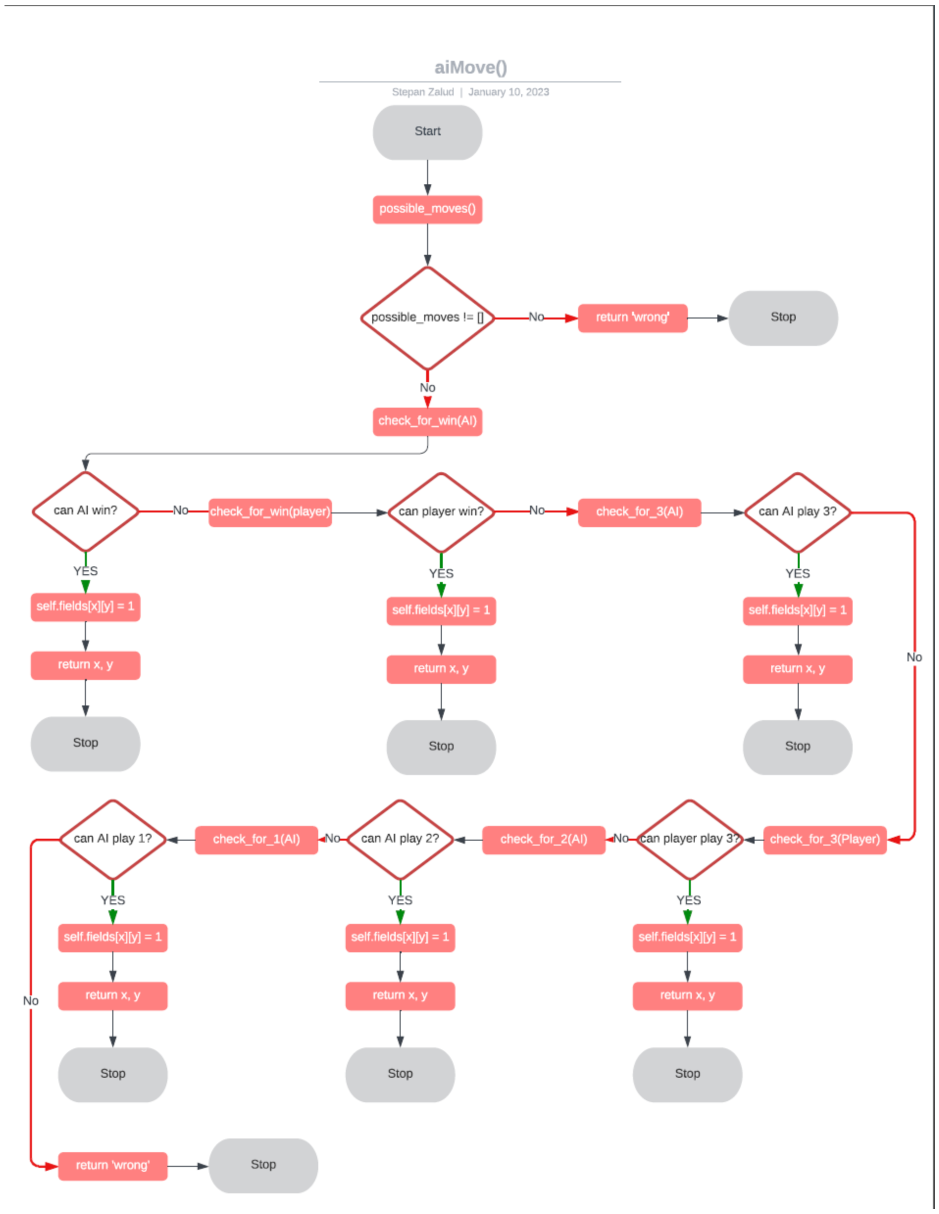


Description:
Program waits for an input in a while loop. Once input is accepted there are three cases possible.
1. Inp == “start”
- AI has the first move. Function self.firstMove() is triggered and AI place it’s first field. In this implementation first move is rather simple. AI applies floor division by 2 for both number of rows

and number of columns and stores the values to x and y variables. This position is saved to self.field variable which stores information whether each field is free or taken, and if taken by whom.
X and Y are then printed as stdout and the loop starts all over.

2. inp == ”exit”
   - if inp takes value “exit” program is terminated

3. else
   If inp is not equal to “start” nor “exit”. Function process input is called with inp as parameter. First program tries to extract 2 numbers for x and y from the input and convert them to integers. If succeeded, function check_inputs() is called to evaluate whether these values are valid. This will happen only if field with these x and y is not yet taken, and x and y are in range of rows and columns of the field. If everything succeeds, this field is saved to self.field variable as opponents field – with value 2 (2 is used for opponent 1 is used for AI) and function aiMove() is called. If anything fails program returns value “wrong” and waits for another input.



aiMove()

This function is responsible for determining where AI's move will be played. This is done by using simple algorithm where AI looks for all the possible moves with different priorities. In this implementation AI expects that whoever manages to take 4 consecutive fields vertically, horizontally or diagonally first, wins. So AI first looks for all the empty fields (function possibleMoves) and than tries to play in each one of them and evaluates the final outcome with different priorities. If ai finds a field that complies with current priority conditions, move is played in this field. If not the whole process is repeated for next priority. If no possible field is find the program returns value "wrong":

1. Priority 1:
   Winning. Can ai play a field which leads to 4 consecutive fields and win the game?
2. Priority 2:
   Prevent opponent from winning. Can ai play a field which prevents opponent from having 4 consecutive fields and win the game next round?
3. Priority 3:
   Make 3 consecutive. Can ai play a field which leads to consecutive fields with possibility to play 4th in next round (means that at least 1 neighboring field is empty)?
4. Priority 4:
   Prevent opponent from making 3 consecutive. Can ai play a field which prevents opponent from having consecutive fields with possibility to pal 4th?
5. Priority 5:
   Make 2 consecutive. Can ai play a field which leads to consecutive fields with possibility to play 3rd and 4th in next rounds (means that at least 2 neighboring field are empty)?
6. Priority 6:
   Play with possibility to win. Can ai play a field which leads to possibility to play 2nd, 3rd and 4th in next rounds (means that at least 3 neighboring fields are empty)

NOTE: for Priority 1 and 2, whenever there is a possible solution found this field is played immediately. For priority 3 – 6 all the possible fields are stored in list and 1 value is randomly picked.

Advantages and disadvantages:
+ easy logic behind the algorithm, playes moves that make sense
+ this logic can be scalable for different win conditions (5 in a row) but grows exponencionally

- not really unbeatable opponent

- hardcoded for 1 winning condition – not flexible
- priorities might be more sophisticated to make ai "more clever"

1. Priorita:
   můžu vyhrát? - > podívá se jestli může někde dát dohromady 4 znaky (vodorovně, svisle, diagonálně
2. Může zabránit protihráči vyhrát? -> podívá se jestli někde může v příštím tahu dát dohromady protihráč 4 znaky (vodorovně, svisle, diagonálně)
3. Může někde poskládat 3 znaky tak aby měl místo na 4? (vodorovně svisle diagonálně)
4. Může zabránit protihráči aby příštím krokem získal 3 znaky s možností v dalším kole zahrát I 4?
5. Může někde napojit aby měl 2 s možností 3, 4 a tahu?
6. Pokud nic z předešlého tak vybere random volné políčko.