**Tic Tac Toe**

In this project your job is to implement [Tic-Tac-Toe](https://en.wikipedia.org/wiki/Tic-tac-toe) for two players. You also can try writing some AI to play the game. If you find it easy, try to make it unbeatable.

**Tasks**

**Initialize the board**

Implement get\_empty\_board() to return an empty 3-by-3 board, a list of lists filled with dots. The inner lists are rows.

1. A list of lists is returned that represents a list of rows.
2. Every cell of the returned value is .
3. The rows of the returned value are independent, changing one row does not affect the others.
4. Printing the result of the get\_empty\_board() function shows the following in the terminal.

[ [ '.','.','.' ],[ '.','.','.' ],[ '.','.','.' ] ]

**Get players' move**

Implement get\_human\_coordinates that asks for user input and returns the coordinates of a valid move on board.

1. The accepts coordinates as a letter and a number: A2 is first row and second column, C1 is third row and first column, and so on.
2. The function returns a tuple of two integers (row, col).
3. The returned coordinates start from 0.
4. The integers indicate a valid (empty) position on the board.
5. The program keeps asking for coordinates if the coordinates provided are outside of board.
6. The program keeps asking for coordinates if the coordinates provided are taken.
7. The program keeps asking for coordinates if the coordinates provided do not match the format.

**Implement making a move**

When the user has chosen a coordinate, that place is marked on the board.

1. If the cell at row and col is empty (contains a dot .), it is marked with the symbol of the player.
2. Out-of-bounds coordinates are not interpreted as moves.
3. Coordinates of already occupied cells are not interpreted as moves.

**Check for winners**

Implement get\_winning\_player() that returns X or O based on the winning player has three of their marks in a horizontal, vertical, or diagonal row on board.

1. The get\_winning\_player() function returns X if X has a three-in-a-row on board.
2. The get\_winning\_player() function returns None if there is no three-in-a-row on the board

**Check for a full board**

Implement is\_board\_full that returns True if the board is full.

1. The is\_board\_full function returns True if there are no empty cells on the board.
2. The is\_board\_full() returns False if there are empty cells on the board.

**Print board**

Implement display\_board() that prints the board to the screen.

1. Players are indicated with X and 0. Empty fields are indicated with dots (.).
2. Coordinates are displayed around the board.
3. The board is displayed in the following format:

1 2 3

A . | . | .

----+---+---

B . | . | .

----+---+---

C . | . | .

**Print result**

The game shows if X or O or no one has won the game

1. If player X wins, "X has won!" is displayed.
2. If player 0 wins, "0 has won!" is displayed.
3. If nobody wins, "It's a tie!" is displayed.

**Game logic**

Implement all the functions so that the game will run successfully

1. Player X starts the game.
2. Players alternate their moves (X, 0, X, 0...).
3. The board is displayed before each move, and at the end of game.
4. The game ends when someone wins or the board is full.
5. The game handles bad input (wrong coordinates) without crashing.

**OPTIONAL TASK: Quit game**

Allow players to quit the game anytime by typing quit in any form of capitalization (quit and QuIt both work).

1. Typing quit instead of coordinates results in the program exiting.

**AI can play**

Implement player-against-AI mode. The AI can drive one of the players, and the game is fully playable against the computer.

1. When the user selects the game mode against AI the function, get\_random\_ai\_coordinates or get\_unbeatable\_ai\_coordinates is called instead of get\_human\_coordinates() when it's Player 0 turn.
2. When the user selects the game mode again AI 'AI-HUMAN', get\_random\_ai\_coordinates or get\_unbeatable\_ai\_coordinates when it's Player 0 turn.
3. Function get\_unbeatable\_ai\_coordinates() and get\_random\_ai\_coordinates returns a valid move (if possible) without asking for any input.
4. Function get\_random\_ai\_coordinates() and get\_unbeatable\_ai\_coordinates returns None if the board is full.
5. Function get\_menu\_option() is implemented as a menu for between choosing 2-player mode and against-AI mode by pressing 1 or 2, respectively.

**OPTIONAL TASK: AI goes for easy win**

AI is capable of recognizing the opportunity to win the game with one move.

1. Function get\_ubeatable\_ai\_coordinates picks the winning move if there is one on the board.

**OPTIONAL TASK: AI prevents easy lose**

AI is capable of recognizing if its enemy could win the game with the next move, and (supposing there is no direct winning move) moves against it.

1. Function get\_unbeatable\_ai\_coordinates (when there is no winning move in one step) picks a move which prevents a certain winning move for its enemy.
2. When there is a direct winning move, function get\_unbeatable\_ai\_coordinates() still picks that.
3. When there are multiple one-step options for the enemy, get\_unbeatable\_ai\_coordinates() tries to prevent one of them.

**OPTIONAL TASK: Unbeatable AI**

AI is unbeatable in all cases.

1. There is no strategy or combination of steps that can win the game against the AI.

**OPTIONAL TASK: AI vs AI**

AI can play against itself

1. When the menu option is called with the argument 'AI-AI', it calls get\_unbeatable\_ai\_coordinates or get\_random\_ai\_coordinates for both players.
2. The game ends without any user input.
3. There is a one second delay between moves to make gameplay easier to follow.

**OPTIONAL TASK: Leaderboard stores in file / database**

Implement a functionality to store a leaderboard of the game in a file or into a database

1. Ask for the players’ name in the beginning of the game.
2. At the end store the results of the game (who won, who lost was it a tie) into a database along with the exact date and time of the game
3. Upon asking with a menu option: show a leaderboard of the players: who got the most points, based on this rule  
   win : lose = 2 : 0, tie = 1:1

**Hints**

* You don't have to come up with an AI strategy. You can search the internet for strategy descriptions. Do not use external code; implement written instructions instead.
* You don't have to implement a general playing strategy. Tic-Tac-Toe has a rather easy unbeatable strategy that can be expressed as a sequence of conditionals.