**project-10**

**Remember that this project cannot be submitted late.**

Write a class called BuildersGame that represents the board for a two-player game that is played on a 5x5 grid. During the game, each players' builders will move around the board and add levels to towers. The winner is the first one to move a builder on top of a 3-story tower.

First, x places her two builders on the board, then o places her two builders on the board. Throughout the game, no two builders can ever occupy the same square. After the initial placements are complete, x must move either one of her builders to an adjacent square (one square orthogonally or diagonally). Builders always move to the top of the tower on their destination square. Builders can move any number of levels down, but can move at most 1 level up (they can also stay at the same level). You can visualize it as the builders hopping from the top of one tower to the top of another, but they can never move to a tower that is more than one level higher than the tower they're on. After a builder moves, it **must** add a level to an adjacent square (it must be adjacent to the builder that moved). A level cannot be added to a square that is occupied by any builder. Once a tower has a 4th level, no further levels can be added. After x has moved and built, the players alternate moving and building in this way until the game ends. If a player moves one of her builders on top of a 3-story tower **or** if her opponent will not have a legal move available, then she has won.

The class should have the following **private** data members - a representation of the board; the current state, which holds one of the three following values: "X\_WON", "O\_WON", or "UNFINISHED"; and something to keep track of whose turn it is.

It should have an init method that initializes the board to being empty, initializes the current\_state to "UNFINISHED", and appropriately initializes any other data members.

Tip: Probably the easiest way of representing the board is to use a list of lists. The init method could then initialize the board to a list of 5 lists, each of which contains 5 empty strings (or whatever character you want to use to represent an empty space).

It should have a get method named get\_current\_state, which returns the current state.

It should have a method named initial\_placement that takes five parameters: the row and column of each of the player's two builders, and either 'x' or 'o' to indicate the player who is placing builders. Rows and columns will be integers in the range 0-4. For example, initial\_placement(0,1,4,2,'o') would place o's builders at row 0, column 1 and row 4, column 2. If one of the chosen squares is already occupied, initial\_placement should return False. Also, if the player placing builders doesn't match the player whose turn it is, or if this method is called for a player that has already made a valid initial placement, then it should return False. Otherwise, it should update the board, update whose turn it is, and return True.

It should have a method named make\_move that takes six parameters: the row and column of the piece to move, the row and column of the square it's moving to, and the row and column of where to build. For example, make\_move(2,0,3,1,3,0) would move the builder at row 2, column 0 to row 3, column 1 and then build a level at row 3, column 0. If the game has already been won or drawn, or if the move is invalid, make\_move should return False. Also, if the builder being moved doesn't belong to the player whose turn it is, or if this method is called before both players have made their initial placements, then it should return False. Otherwise, it should record the move, update the current state, update whose turn it is, and return True. To update the current state, you need to detect if this move put a builder on top of a 3-story tower, or if the opponent will not have a legal move available.

It's not required, but you'll probably find it useful for testing and debugging to have a method that prints out the board.

Whether you think of the list indices as being [row][column] or [column][row] doesn't matter as long as you're consistent.

As a very simple example, your class could be used as follows:

game = BuildersGame()

game.initial\_placement(2,2,1,2,'x')

game.initial\_placement(0,1,4,2,'o')

game.make\_move(2,2,1,1,1,0)

game.make\_move(0,1,1,0,2,0)

game.get\_current\_state()

Your file must be named: BuildersGame.py