**portfolio-project**

Write a class named QuoridorGame for playing a board game called Quoridor.

Details about the game, validation rules, implementation, game play and extra credit are below.

**Game**

You can see the rules of the game in this [video](https://www.youtube.com/watch?v=6ISruhN0Hc0) and [Page 9 of this educative-sheet\_quoridor-english.pdf](https://en.gigamic.com/files/media/fiche_pedagogique/educative-sheet_quoridor-english.pdf) You can implement the fair play rule for extra credit but you are not required to.

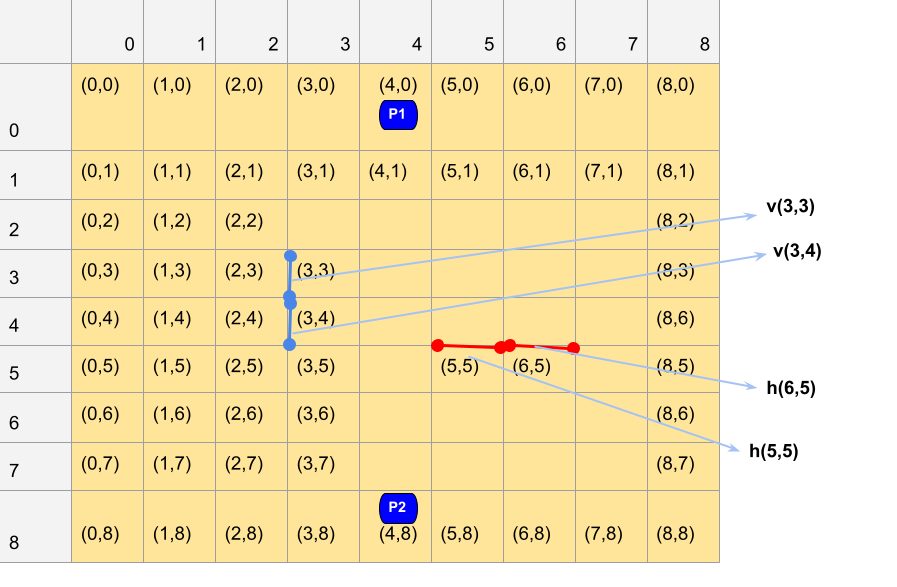
Address of the video: <https://www.youtube.com/watch?v=6ISruhN0Hc0>

Address of the PDF document: <https://en.gigamic.com/files/media/fiche_pedagogique/educative-sheet_quoridor-english.pdf>

You will be writing a program for a two-player version of the game. Each player will have 10 fences.

The board is formed by 9x9 cells, and the pawn will move on the cells. The fence will be placed on the edges of the cells. The four sides of the board are treated as fences and no more fence should be placed on top of it.

The board should be treated as the following picture shows:



The cell coordinates are expressed in (x,y) where x is the column number and y is the row numberThe board positions start with (0,0) and end at (8,8). At the beginning of the game, player 1 places pawn 1 (P1) on the top center of the board and player 2 places pawn 2 (P2) on the bottom center of the board. The position of P1 and P2 is (4,0) and (4,8) when the game begins.

The four edges are labeled as fences. The row of the cells where the pawns are positioned at the start of the game are called base lines. A fence is 1 cell long in contrast to what you find the video and PDF saying.

When each player tries to place a fence on the board, the position of the fence is defined by a letter and coordinates. For vertical fences, we use v and for horizontal fences, we use h. As an example, for the blue fence (vertical) in the picture, we use the coordinate of the top corner to define it and for the red fence (horizontal), we use coordinate of the left corner to define it.

**Validation rules**

Consult the video and the PDF file's page 9, linked at the top of this README, to understand the game play rules and make sure that your program implements them.

For example, jumping over the pawn is allowed only when the two pawns face each other. Diagonal movement is allowed when blocked by pawn + fence. A fence only blocks 1 square. Fences cannot be moved once placed thus they cannot be reused. All the rules from the video and the PDF apply unless the README or an Instructor explicitly says otherwise.

**Playing the game**

Player 1 will start the game. Each player takes turn playing. On a player’s turn they will make one move. They can either move the pawn (move\_pawn) or place a fence (place\_fence). Your program should be able to determine whether the movement is valid. A turn lasts until the player has made a valid move.

The first player whose pawn reaches any of the cells of the opposite player's base line wins the game. No turn can be played after a player has won.

**Implementation**

Your QuoridorGame class must include the following methods:

* init method that initializes the board with the fences (four edges) and pawns (P1 and P2) placed in the correct positions.
* move\_pawn method takes following two parameters in order: an integer that represents which player (1 or 2) is making the move and a tuple with the coordinates of where the pawn is going to be moved to.
  + if the move is forbidden by the rule or blocked by the fence, return False
  + if the move was successful or if the move makes the player win, return True
  + if the game has been already won, return False
* place\_fence method takes following parameters in order: an integer that represents which player (1 or 2) is making the move, a letter indicating whether it is vertical (v) or horizontal (h) fence, a tuple of integers that represents the position on which the fence is to be placed.
  + if player has no fence left, or if the fence is out of the boundaries of the board, or if there is already a fence there and the new fence will overlap or intersect with the existing fence, return False.
  + If the fence can be placed, return True.
  + If it breaks the fair-play rule (and if you are doing the extra credit part), return exactly the string breaks the fair play rule.
  + If the game has been already won, return False
* is\_winner method that takes a single integer representing the player number as a parameter and returns True if that player has won and False if that player has not won.
* You might also find implementing a print\_board method useful to print the board to the screen. It's not required that you implement this method.

Feel free to add whatever other classes, methods, or data members you want. All data members must be private. All methods must have no more than 20-25 lines of code - don't try to get around this by making really long or complicated lines of code. (The rule in real life is just to not stuff too much into a single function, but that's probably too nebulous a rule for you at this point, and if your function is over 25 lines, you probably are trying to stuff too much into it.)

**How your game will be played?**

Here's a very simple example of how your QuoridorGame class will be used and is expected to behave, by the autograder or a TA:

q = QuoridorGame()

q.move\_pawn(2, (4,7)) #moves the Player2 pawn -- invalid move because only Player1 can start, returns False

q.move\_pawn(1, (4,1)) #moves the Player1 pawn -- valid move, returns True

q.place\_fence(1, 'h',(6,5)) #places Player1's fence -- out of turn move, returns False

q.move\_pawn(2, (4,7)) #moves the Player2 pawn -- valid move, returns True

q.place\_fence(1, 'h',(6,5)) #places Player1's fence -- returns True

q.place\_fence(2, 'v',(3,3)) #places Player2's fence -- returns True

q.is\_winner(1) #returns False because Player 1 has not won

q.is\_winner(2) #returns False because Player 2 has not won

**Notes**

The program file must be named **Quoridor.py**.

**You cannot use any library, unless it's approved by an Instructor**. To get approval, make a post on Ed stating the name of the library and the reason or scenario you want to use it for. Once approved, anyone in the class can use that library. **Currently the following library or methods are approved: math, pyplot, pprint, copy**