Project Introduction:

For this project you will write a class called LudoGame that allows two to four people to play a simplified version of the game (https://en.wikipedia.org/wiki/Ludo_(board_game)).

For this board game, two, three, or four players can play. Each player has 2 tokens. At the beginning of the game, each player's two tokens are in the player's home yard. Each player takes a turn by rolling the dice. On a turn, a player can move a token that is on the board clockwise the number of steps indicated by the die along the track. Moving a token out of the home yard onto the board' "ready to go position" can only be done by rolling a **6**. Rolling a 6 earns the player an additional roll in that turn. If the bonus roll results in a 6 again, there is no additional roll again.

After a certain number of steps, the token will enter that player's home squares which no opponent may enter. Then the token will try to enter the finishing square (end). The token must reach the finishing square on an **exact roll**. If the roll number is larger than the steps needed to get to the finishing square, the token will bounce back the remaining number of steps.

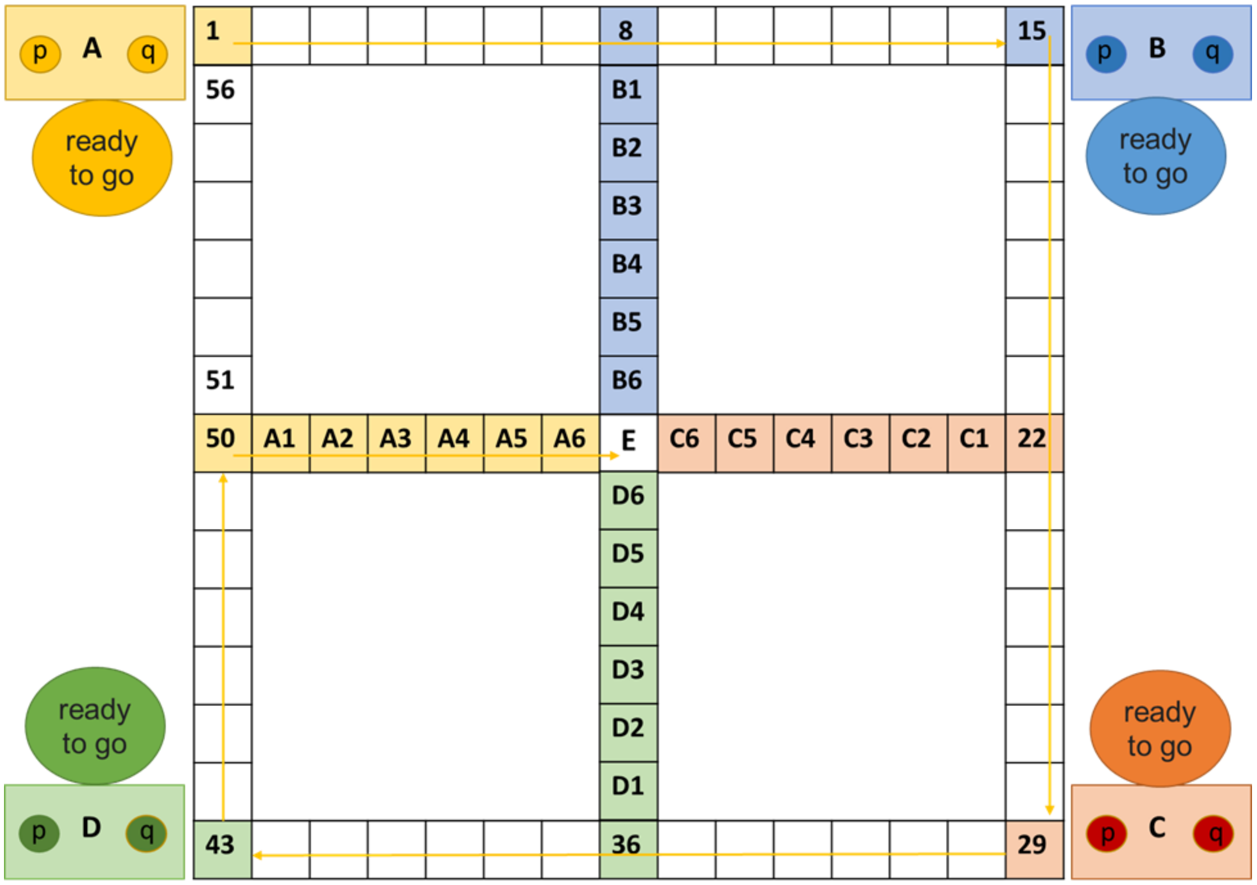
Winning the game: the first player whose 2 tokens have entered the finishing square will win the game. The rest will continue playing until there is only one player left.

Additional Playing Rules:

When a token finishes one move, if it lands on a space occupied by an opponent's (other player's) token, the opponent token will be **returned** (kicked back) to its home yard. The returned token can re-enter into play when the owner rolls a 6.

If the player's two tokens land on the same space on the board, the player will **stack** the two tokens and move them as one piece until they reach the finishing square. When stacked pieces are sent back to their home yard by an opponent landing on them, they are no longer stacked. Note that if two tokens are both at the "ready to go" position, they are not stacked.

An example game board can be seen in the following picture. We have four positions 'A' 'B 'C' and 'D' for players to choose. All tokens will move **clockwise 50 steps** on the board and then enter their corresponding home squares, but different positions have different **start** and **end** space. At the game beginning, if the player at position A rolls the number 6, token "Ap" in player A's home yard will move to the "ready to go" position. Then player A gets another roll and rolls, for example, the number 4, and the token will enter the board from space "1" and then move 4 steps to space "4". Then the token will move clockwise up to space "50" and then enter position A's **"home squares"** A1, A2, … A6. When the token reaches square "E" on an exact move, the token has finished. If the token lands on space "A5", and then moves 4 steps, it will bounce back to A5 again. If the token moves 5 steps instead, it will land on A4. Here, space “1” is position A’s start space number, and space “50” is position A’s end space number. For player at position B, the start and end number are 15 and 8. For C they are 29 and 22. For D they are 43 and 36.



Decision-making Algorithm:

In this project, you will also implement a **decision-making** algorithm for a player to choose a certain token to move. With a given roll, if the player can't move any token, or can only move one token (or if the two tokens are stacked), the player has no other choice. If the player has two tokens on the board that can be moved, then player will use the following priority rules to decide which token to move:

1. If the die roll is 6, try to let the token that still in the home yard get out of the home yard (if both tokens are in the home yard, choose the first one 'p')
2. If one token is already in the home square and the step number is exactly what is needed to reach the end square, let that token move and finish
3. If one token can move and kick out an opponent token, then move that token
4. Move the token that is further away from the finishing square

For testing purposes, we will not use the random function to generate the dice number for each roll. Instead, we will pass in the roll numbers manually when we call the method. We also don't require a GUI for this project and all the output will be in the text format. You can improve your code later on after you finish the required part to make it your own portfolio project.

Your code for the game must define the class and methods described below, but you are encouraged to define other methods or classes that may be useful for the game. All data members must be **private**.

Player:

- The Player object represents the player who plays the game at a certain position. The class must contain the following information (but may have more):
  - the position the player choose (A, B, C or D)
  - start and end space for this player
  - current position of the player's two tokens: in the home yard, ready to go, somewhere on the board, or has finished.
  - the current state of the player: whether the player has won and finished the game, or is still playing
- The Player class should have those methods (but may have more):
  - get_completed: takes no parameters and returns True or False if the player has finished or not finished the game
  - get_token_p_step_count: takes no parameters and returns the total steps the token p has taken on the board (use steps = -1 for home yard position and steps = 0 for ready to go position) The total step should not be larger than 57. Please note that if the token is bounced back in the home squares, this bounced part should be subtracted from the step count. For example, when token p is at space A5, the total step is 55 now. If it moves 5 steps and bounces back to A4, the total step should be 54, not 60.

    - get_token_q_step_count: takes no parameters and returns the total steps the token q has taken on the board (use steps = -1 for home yard position and steps = 0 for ready to go position)
    - get_space_name: takes as a parameter the total steps of the token and returns the name of the space the token has landed on on the board as a string. It should be able to return the home yard position (‘H’) and the ready to go position (‘R’) as well.

<u>LudoGame:</u>

- The LudoGame object represents the game as played. The class should contain information about the players and information about the board, it must contain the following methods (but may have more):
    - get_player_by_position takes a parameter representing the player’s position as a string and returns the player object. For an invalid string parameter, it will return "Player not found!"
    - move_token method takes three parameters, the player object, the token name (‘p’ or ‘q’) and the steps the token will move on the board (int). This method will take care of one token moving on the board. It will also update the token’s total steps, and it will take care of kicking out other opponent tokens as needed. The play_game method will use this method.
    - play_game method takes two parameters, the players list, and the turns list. The players list is the list of positions players choose, like [‘A’, ‘C’] means two players will play the game at position A and C. Turns list is a list of tuples with each tuple a roll for one player. For example, [('A', 6), ('A', 4), ('C', 5)] means player A rolls 6, then rolls 4, and player C rolls 5. This method will create the player list first using the players list pass in, and then move the tokens according to the turns list following the priority rule and update the tokens position and the player’s game state (whether finished the game or not). After all the moving is done in the turns list, the method will return a list of strings representing the current spaces of all of the tokens for each player in the list after moving the tokens following the rules described above. **‘H’ for home yard, ‘R’ for ready to go position, ‘E’ for finished position, and other letters/numbers for the space the token has landed on**.

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

```
players = ['A', 'B']
turns = [('A', 6), ('A', 4), ('A', 5), ('A', 4), ('B', 6), ('B', 4), ('B', 1), ('B', 2), ('A', 6), ('A', 4), ('A', 6), ('A', 
3), ('A', 5), ('A', 1), ('A', 5), ('A', 4)]
game = LudoGame()
current_tokens_space = game.play_game(players, turns)
```

```
player_A = game.get_player_by_position('A')
print(player_A.get_completed())
print(player_A.get_token_p_step_count())
print(current_tokens_space)
player_B = game.get_player_by_position('B')
print(player_B.get_space_name(55))
```

And the output will be:

```

False
28
[‘28’, ‘28’, ‘21’, ‘H’]
B5
```