Rationale

Based on design, I choose to use strategy pattern to design the feature part of the game, since roads, cities, monasteries and fields are all kinds of feature. I additionally added the center block and center connected for solving the issue that might potentially happened for the center of tile type W, X, and O, P. All six features will share same functionality for checking to get the enum feature type. The design makes this part low representational gap and robustness.

Furthermore, a tile class will contain five segments that is not in the previous design before for representing different segments on a tile: center, left, right, up, and down. Each of the five segments will contain information include if it has a meeple on it, the feature of the segment, its position on the tile, and its neighbor segments. At the same time, the tile class will have a coordinate object for representing the position in the board.

At the part of generating all types and numbers of tiles into a stack, I have first created a JSON for tile type collection that has 24 different types of tiles with distinct features on 4 sides and the center, as well as the type number from 0 to 23. Then the tile generator will read the JSON file and construct a list of all 72 tile types and then shuffle then into a Deque data structure for representing a tile stack in order to feed player a tile. A tile stack object will contain such a tile generator. When calling to get a tile, the tile stack will first pop out a tile type from the tile generator and then use the method the tile generator has to form an actual tile object from the tile type and return it to whatever object call the get a tile function. In this case, it is easy for the application to initialize since there is no need to generate all kinds of tiles at the initialization stage, but get a tile each time when player get a tile type from the tile stack. The design will meet the robustness, scalability, effectiveness, low coupling and high cohesion at once for design goals and design principles.

In the player part, a player class contains a player ID in order to distinguish between different players, the number of current meeples the player has, and the player's current score.

Meanwhile, the scoring system object is contained in the overall object and has two methods: score completion and score the whole board. The score completion is for the time after a player finished placing a meeple. It will check from the tile just placed and search for three different features: road, city and monastery. It will sequentially check the feature completion, if so, then search all the meeples in the completed feature and score for the players. For the other method, the scoringWholeBoard, it happens when there is no tile on the tile stack and after the last tile is being scored, it will be called for final scoring.

In the Carcassonne class that contains the current whole game setting, it has the tiles stack, the player list, the board representation, and the scoring system that score the points to user after each meeple is placed on a tile.

Additionally, there will be a gaming interface that allow user to enter the number from 2 to 5 of players and pass it to the game for player list initialization. This design is of high cohesion and low coupling, with robustness, and efficiency.