Design Rational

The goal of the game system class is to manage the high-level game logic, it has methods like start new game, get a random tile from tile stack and place the tile. After each move, game system will call the game board to get any complete features. For each complete features, game system class will directly call the feature class to calculate its score and add score for winner. I separate the get complete features and calculate scores to two different class (game board and feature) because each class should have clear responsibility (calculate score for feature should be in feature class and add score for winner should be in game system class).

The goal of game board class is to simulate the placement of all the tiles in the game board. When adding one tile in game board, it has method for checking whether this position is valid (should have at least one neighbor tiles) and checking whether this placement match the edge feature of all its neighbor tiles. This class is also responsible for adding this new tile to its adjacent matched features because only this class have information of all the placed tiles. When checking for completed features after each move, road features and city features share the same implementation because they only need to consider four abut tiles. However, since monastery feature has to consider its eight abut tiles and monastery can only occupy one tile, we have to consider processing the monastery feature separately.

For each tile in Tile class, it has a Tile Type which indicates the type and an id which distinguishes this one from other tiles in the same type. The tile is initialized by four edge features and its default orientation is UP. Because in some cases, some of these four edge features could be interconnected. So I made a 2d array for indicating four edge features in tile class, the first dimension indicates different features, the second type indicates which orientation is contained in this feature. For field feature, I set it to null because field feature does not contribute to any scoring. Also noted that the tile can be rotated before placing, so I add a setOrient method to change the orientation from default to other orientations, and when other class wants to get edge feature by specified orientation after orientation, I added a getRotateFeatureByOrient to make it consistent. Finally, there are two special marks in tile class, the monastery and coat of arm. Because they are special in any sense, I made them into two boolean values to indicate whether the tile contains these special marks or not.

Feature class is an abstract class and have three subclasses: city feature, road feature and monastery feature. This class should be able to combine with another feature to get one bigger feature. For city and road, the method for checking complete is to check whether all the edge features in this combined feature can be grouped into pairs. By grouping into pairs, the two edge feature should be in two abut tiles and their orientations should match accordingly. For monastery feature, the method for checking complete is to count all its abut tiles, if this number goes up to 9 (including itself), then the monastery is complete. Since I also include the list of meeples in Feature class, when the feature is complete and calculating its score, the feature should determine the winners (one or maybe more) and return to game system.

Finally, player class contains the score, name and list of his meeples, the reason I put the list of meeples here is that it is more convenient to get how many meeples are left for this player. If player has used up all his meeples, then he cannot specify any meeple position in game system.