Discussion

From a, For the segment part in my design, there is no such concept since I was using feature to represent each segment on the tile. I was imaging that each player will have its own scoring system to help them add scores and return meeples. And for the tiles generator part, I have thought of using the Enum for representing different tile types and added all of the 72 tile types into a list and shuffled them to a deque for representing the tile stack. Then when a player needs a tile, the tile stack will pop out a tile type and use a method to make it a tile object in order to use. Moreover, there is no center block and center connected feature for solving some design dilemma for tile type J and K.

In b part, after getting some advice from the TAs, I found that adding a segment class might be better solving the representing gab and some conceptual conflict when I was designing the algorithm. And I really found it useful after I added the segment concept to the algorithms. This is one of the major design change in my Carcassonne game from a to b part of homework. Adding segment concept for the tile instead of just use feature for representing segment on the tile will make the representation more natural. Another major change I have made from a is that I only use a list of segment neighbors for each segment on a tile instead of connect all segment with bidirectional connection, such as this segment's left, right, up, or down segment is some other segment. This design decision really solved a lot problems because a segment does not need to know exactly what its left, right, up, or down segment is. It only need to know what are the neighbors and what is its current position on a tile. For the tiles generator part, I have referred to the Piazza's post approach and used JSON file to process the tile type, but the idea of adding all tile types to a list, shuffled them to a deque and make the tile object from the tile type popped out from the deque when calling the getTile method stays the same as I imaged in part a.

In c part, the logic part I have adjusted mostly is the scoring system part. How to check if a feature is complete for road and city stuck me. And after a deep thinking, I have come out the solution: when the feature has an open end, that is an incomplete. Other parts are little adjustment from b for being compatible for java swing for creating user interface.