andrew id:adrianlc

Rationale

The principle of this design is to let me have a easier time writing the code and also function well. And the design goal is to achieve efficiency, robustness, and reusability. In this program I didn't consider a lot about scalability because I think there won't be much need to scale up this program since this program's max player number is already five players. And the flexibility is not very well because I don't think there's going to be a lot new rules added in the the program, and if it does I could be able to handle it.

To evaluate how my program works, first we can look at how I represent the tile. I divide a tile into four direction(north, south, west, east) which each contains a list of segments, a variable for having a cloister in the tile and a variable for the connection of segments. In this way, I can fully represent the tile and handle all conditions. Although I make cloister a special case, I think in this particular problem it's fine. There shouldn't be much need to scale up the program and if there is an expansion I can still modified my program and achieve it. Also, if a player wants to rotate or place a follower on the tile that he gets, he can just call tile.rotate or tile.placeFollower(the place). Second, to achieve the scoring function, I don't want to implement a lot of functions that will traverse through the map to find completed feature. I think the performance will not be good. Instead I update the AllFeatures object every time there's a tile been placed on the map(update the breaches of the feature and tiles). The AllFeatures object caches all the features on the map and their breaches. So every time I want to check for a complete feature I can just go into the AllFeatures object, get the feature and see the number of breaches in that feature. Also, I will add the new tile's neighbor information into the new tile. Third, to validate the placement of a tile, I will get the information of the new tile's neighbor and the features they are in. Then I can compare them with the new tile(ex: Are there followers on the feature already). In this way, there won't be a need to traverse through the map the find whether the placement of the new tile is valid or not. Four, after all 72 tiles are placed, I can just go into the AllFeatures object and count the score for each player. Last but not least, the implementation of current map is a hashtable<Pair(x,y), Tile> that will store the x y coordinate and tile. The game will only have to get the hashTable and show the map according to the xy coordinate. And everytime I want to add a tile on the map I just have to push it in the hashtable. In addition, there will be a border list that contain the border of the map so that I can compute whether there's no valid placement for a tile.

As for GUI, the game object will take the responsibility. The game object will call showMap() getNextTile getScore() method for every turn and let the players understand the situation of the game. And there are methods like placeTile() and rotateTile() to let the players control the game.