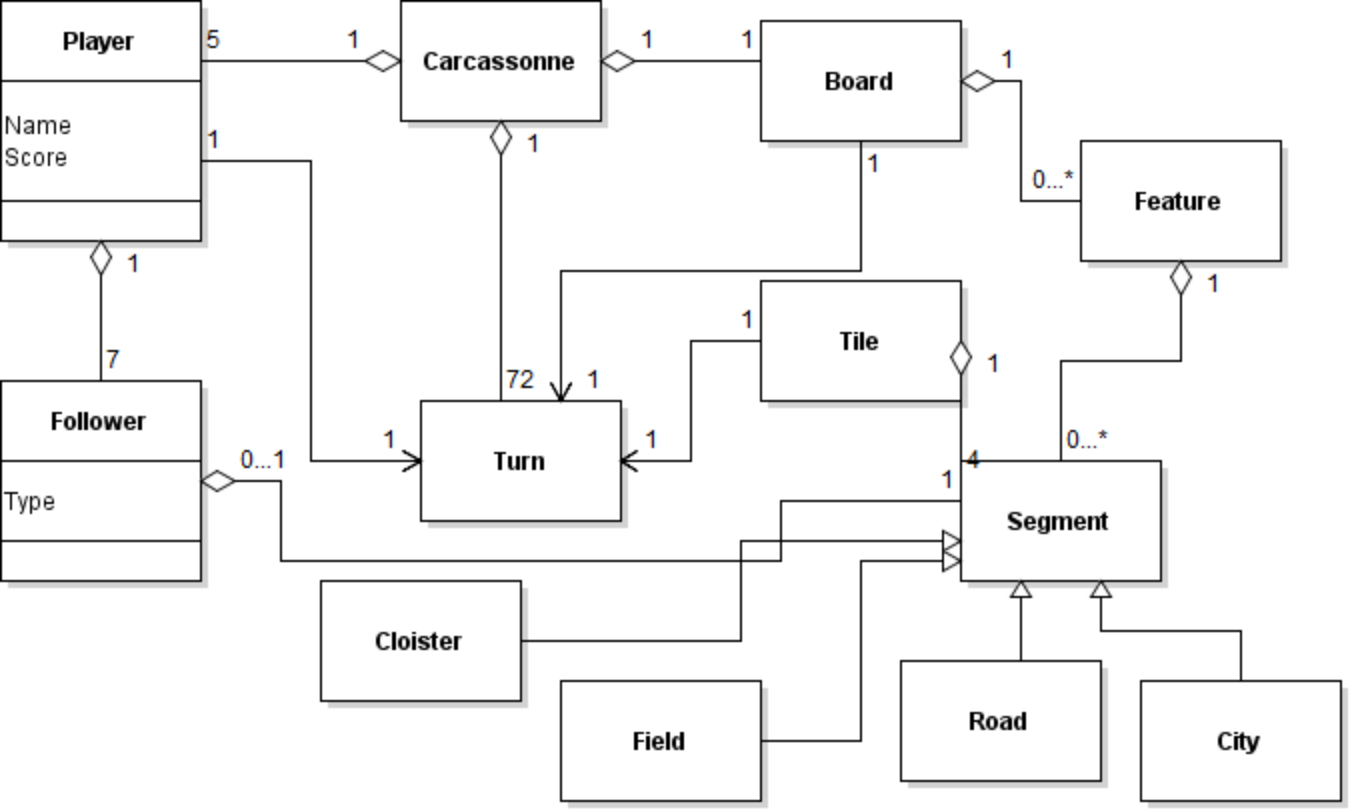
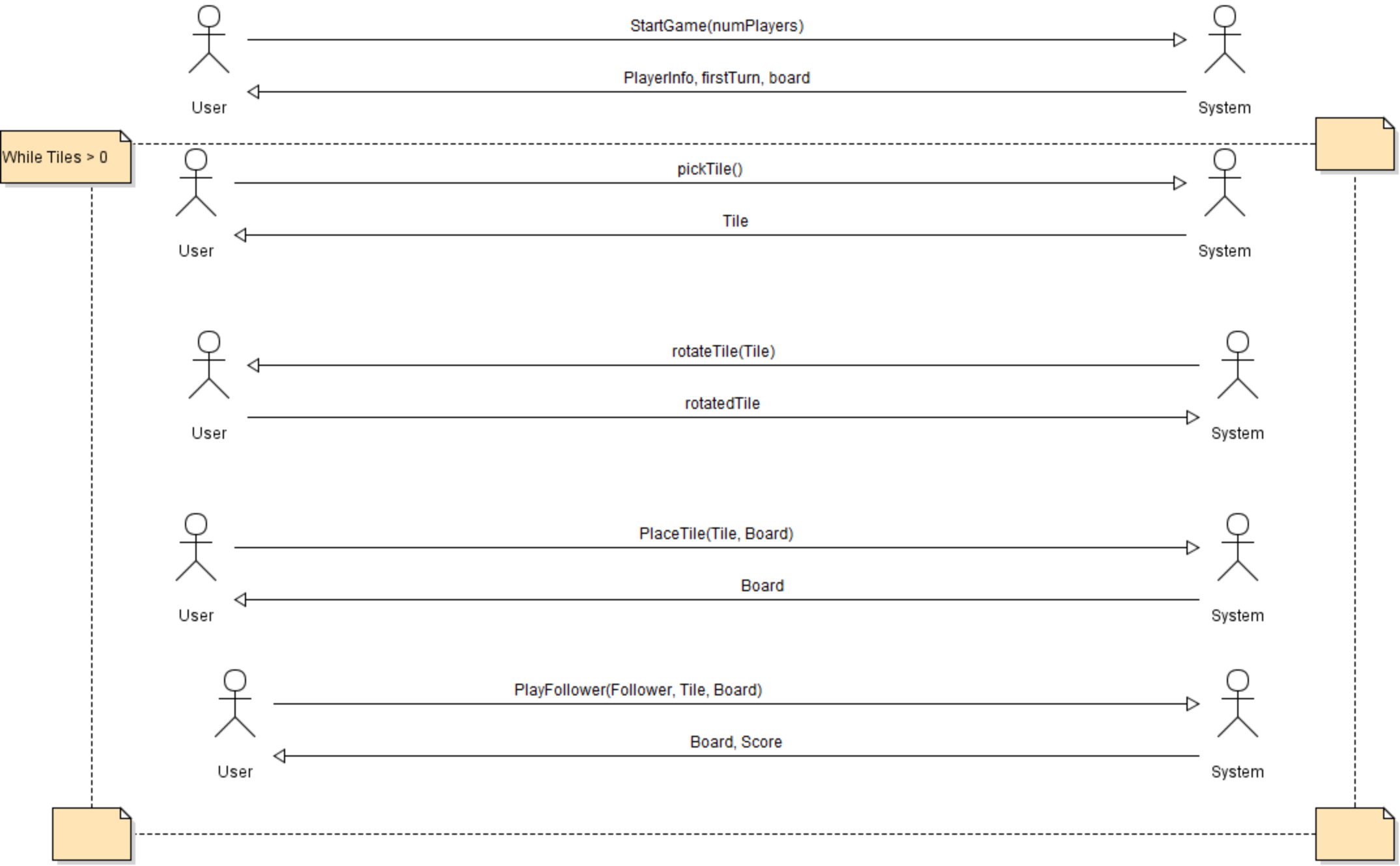
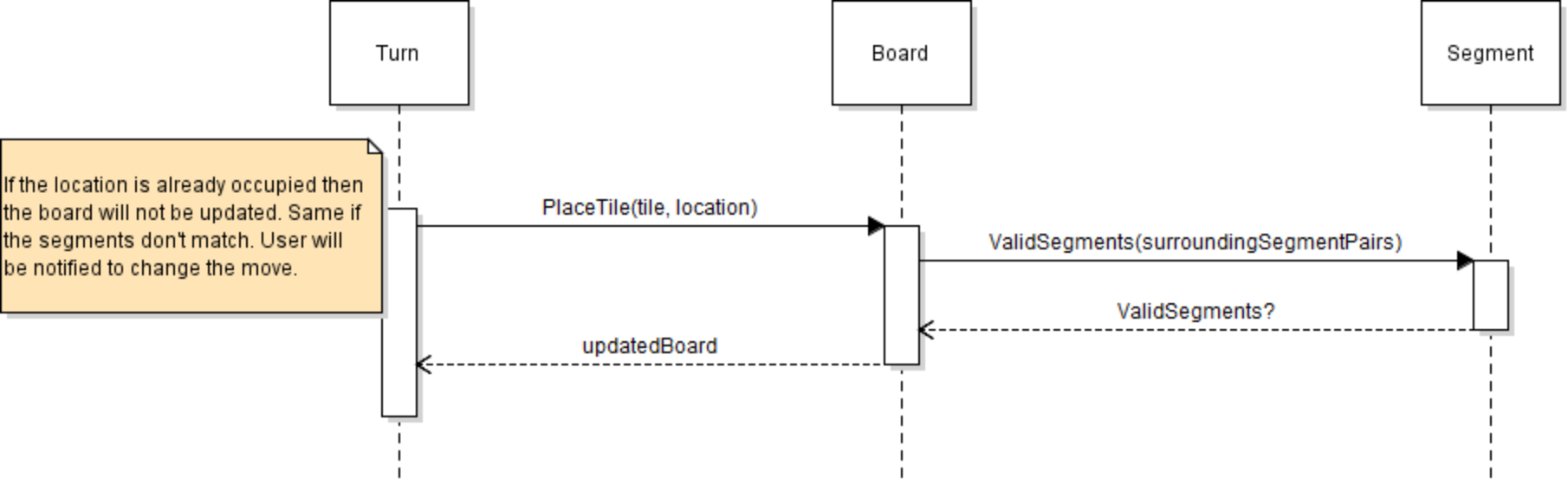

Player
Name
Score
5
1
Carcassonne
1
1
Board
1
0...*
Feature
1
1
1
1
1
1
1
7
72
1
Follower
Type
0...1
1
Turn
1
1
Tile
1
0...*
4
1
Segment
Cloister
Field
Road
City

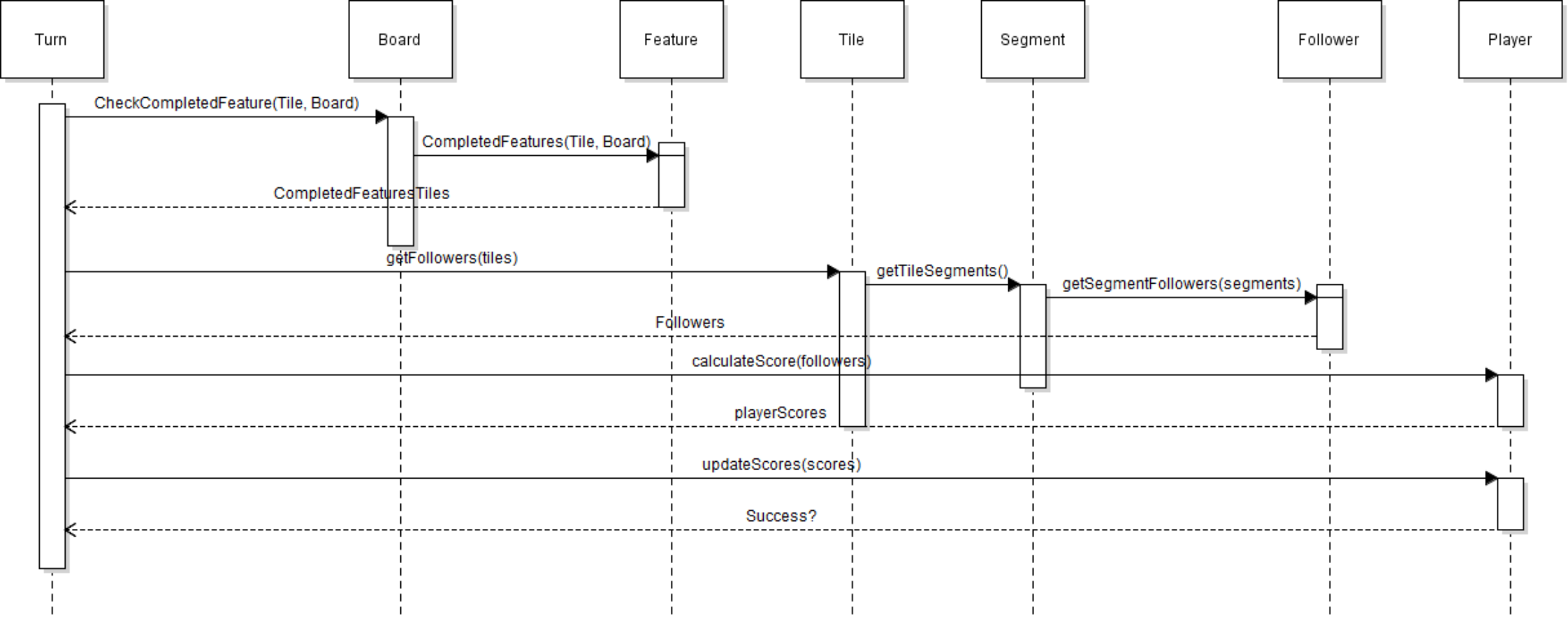

StartGame(numPlayers)
PlayerInfo, firstTurn, board
User
System
While Tiles > 0
pickTile()
Tile
User
System
rotateTile(Tile)
rotatedTile
User
System
PlaceTile(Tile, Board)
Board
User
System
PlayFollower(Follower, Tile, Board)
Board, Score
User
System

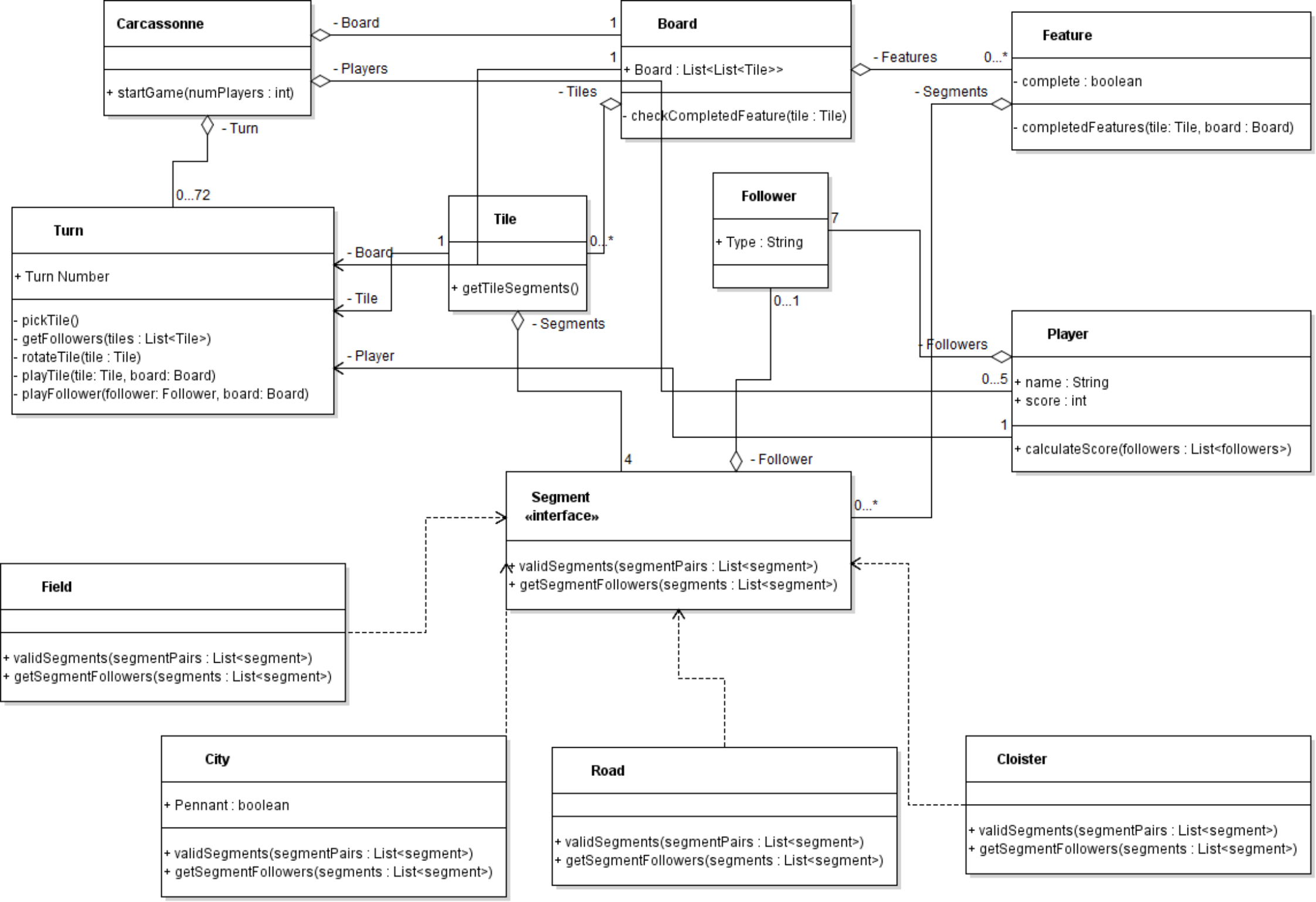
Operation: PlayTile(Tile, BoardLocation)

Preconditions: User has picked up tile. The boardLocation is not occupied by another tile. The segments on the placed tile match up with the abutting segments.

Postconditions: The board contains the tile in the specified location.


Turn
Board
Segment
If the location is already occupied then the board will not be updated. Same if the segments don't match. User will be notified to change the move.
PlaceTile(tile, location)
ValidSegments(surroundingSegmentPairs)
ValidSegments?
updatedBoard


Turn
Board
Feature
Tile
Segment
Follower
Player
CheckCompletedFeature(Tile, Board)
CompletedFeatures(Tile, Board)
CompletedFeaturesTiles
getFollowers(tiles)
getTileSegments()
getSegmentFollowers(segments)
Followers
calculateScore(followers)
playerScores
updateScores(scores)
Success?


Carcassonne
+ startGame(numPlayers : int)
- Board
1
Board
+ Board : List<List<Tile>>
- checkCompletedFeature(tile : Tile)
- Players
1
- Tiles
- Features
0...*
Feature
- complete : boolean
- completedFeatures(tile: Tile, board : Board)
- Segments
- Turn
0...72
Turn
+ Turn Number
- pickTile()
- getFollowers(tiles : List<Tile>)
- rotateTile(tile : Tile)
- playTile(tile: Tile, board: Board)
- playFollower(follower: Follower, board: Board)
Tile
+ getTileSegments()
1
0...*
- Board
- Tile
- Player
- Segments
Follower
+ Type : String
7
0...1
- Followers
Player
+ name : String
+ score : int
+ calculateScore(followers : List<followers>)
0...5
1
4
- Follower
Segment
«interface»
+ validSegments(segmentPairs : List)
+ getSegmentFollowers(segments : List)
0...*
Field
+ validSegments(segmentPairs : List)
+ getSegmentFollowers(segments : List)
City
+ Pennant : boolean
+ validSegments(segmentPairs : List)
+ getSegmentFollowers(segments : List)
Road
+ validSegments(segmentPairs : List)
+ getSegmentFollowers(segments : List)
Cloister
+ validSegments(segmentPairs : List)
+ getSegmentFollowers(segments : List)

One key element of my object model design is the segments. I designed it so each tile has 4 segments, one for each of the sides of the tile. So if the tile is a 4 way crossroads, it would contain 4 road segments. There are 4 different kinds of segments: field, city, road, and cloister. They all have shared functionality to check if a combination of segments is valid and to get the followers associated with a particular segment. Because of this they implement the segment interface. The strategy pattern will be used here with the differing algorithm being the differing calculations corresponding to the different segments.

I chose what objects to include with a heavy focus on real world representation. Most of the objects are taken directly from the rulebook, including the turn object. This serves as the workhorse of moving the game forward. It contains the functionality for the user making choices for the most part. I chose for the turn object to contain this rather than the player object because I wanted the player object to be more of an information chest for the player rather than the control board for the user. Giving the turn object this functionality also splits the game nicely into all of its turns rather than having them blend together.

One situation I made sure to account for in my design was the one where two followers exist on separate segments and are then combined. This results in multiple followers existing in a feature, whether that feature is complete or not. Since the segment has the follower instead of the feature its simple to build up a count of followers in the growing feature as more tiles are added. We can just examine all the segments that make up the feature and that will give us an exact count of the followers.

Another situation I based my design around is that of validating a placement of a tile on the board. The validity of a tile placement is first checked by the board to ensure that the tile is abutting another tile and it is not being placed in an occupied spot. After that there is a check done to ensure that the abutting segments match appropriately. I chose to have the first check done by the board and the second check done by the segment object. It felt more appropriate this way because its more intuitive that a segment object makes sure the segments are correct and the board object makes sure the tile is put in a valid spot on the board.

One thing that should be explained is how the board is being represented. It is a two-dimensional list of tiles. It starts with the starting tile right in the center. As tiles are placed we add the tile to the correct spot in this 2d list. We will make it a 73 by 73 list to begin with in order to contain any possible combination of tile placements.

The score is calculated by the player object as the score is stored as an attribute on the player object. I felt it would make the most sense if they were in the same object and in building the object interaction diagrams it worked out well this way. The Carcassonne object is mainly used just to start the game and be a sort of wrapper object for the rest of the game. It takes user input in the initialization of the game like the number of players desired.