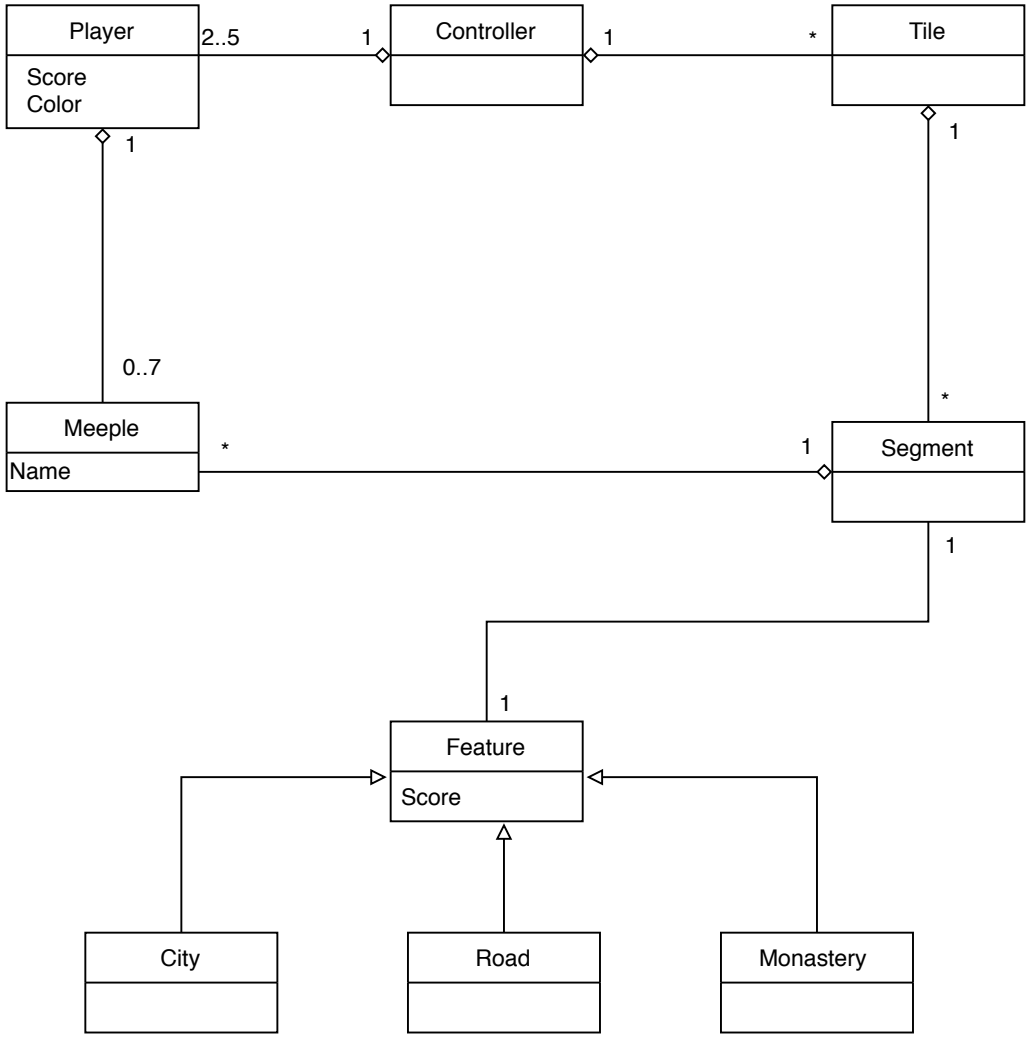
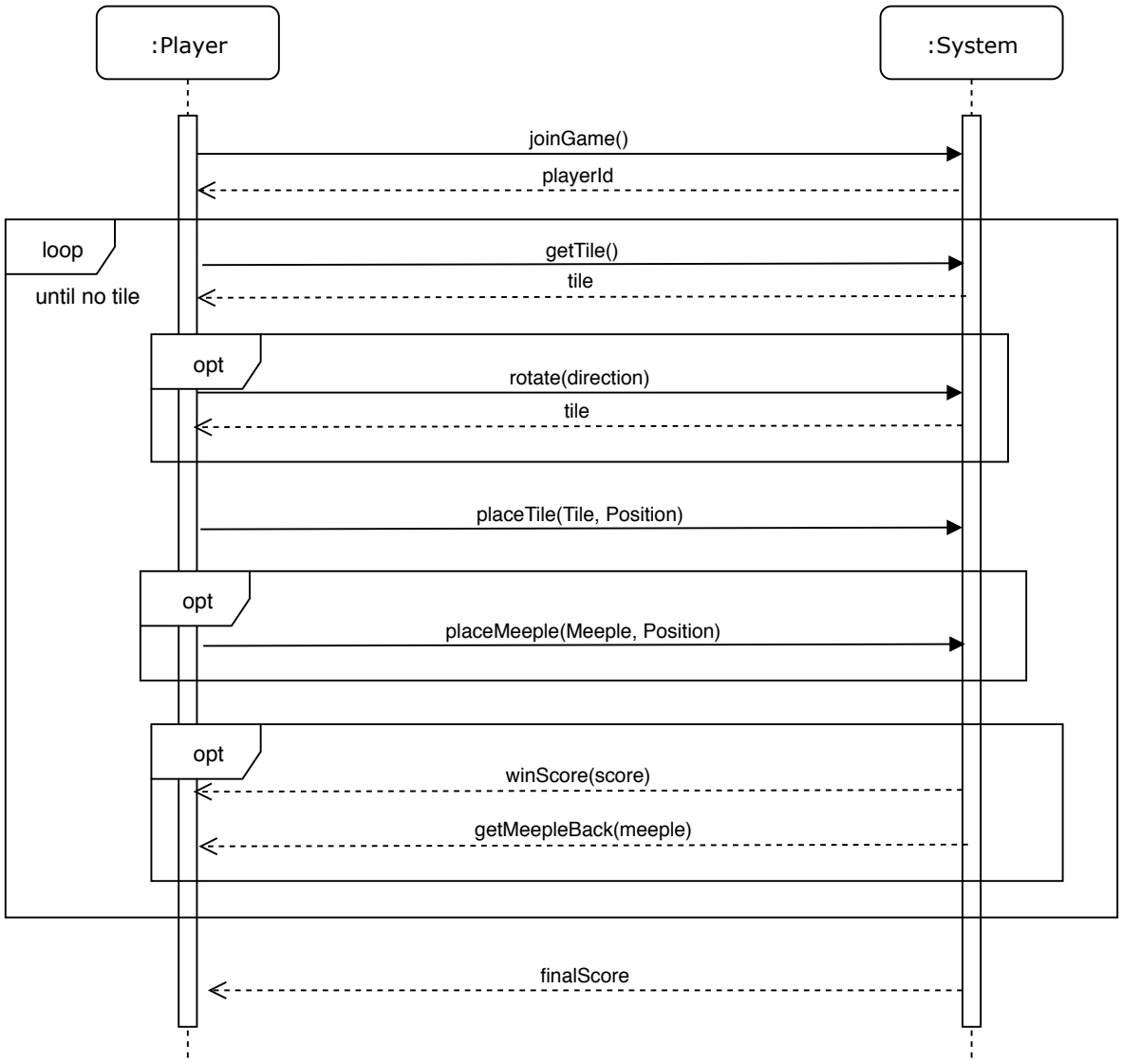

Player
Score
Color
2..5
1
Controller
1
*
Tile
1
1
0..7
*
Meeple
Name
*
1
Segment
1
1
Feature
Score
City
Road
Monastery


:Player
:System
joinGame()
playerId
loop
until no tile
getTile()
tile
opt
rotate(direction)
tile
placeTile(Tile, Position)
opt
placeMeeple(Meeple, Position)
opt
winScore(score)
getMeepleBack(meeple)
finalScore

behavioral contract

Interaction

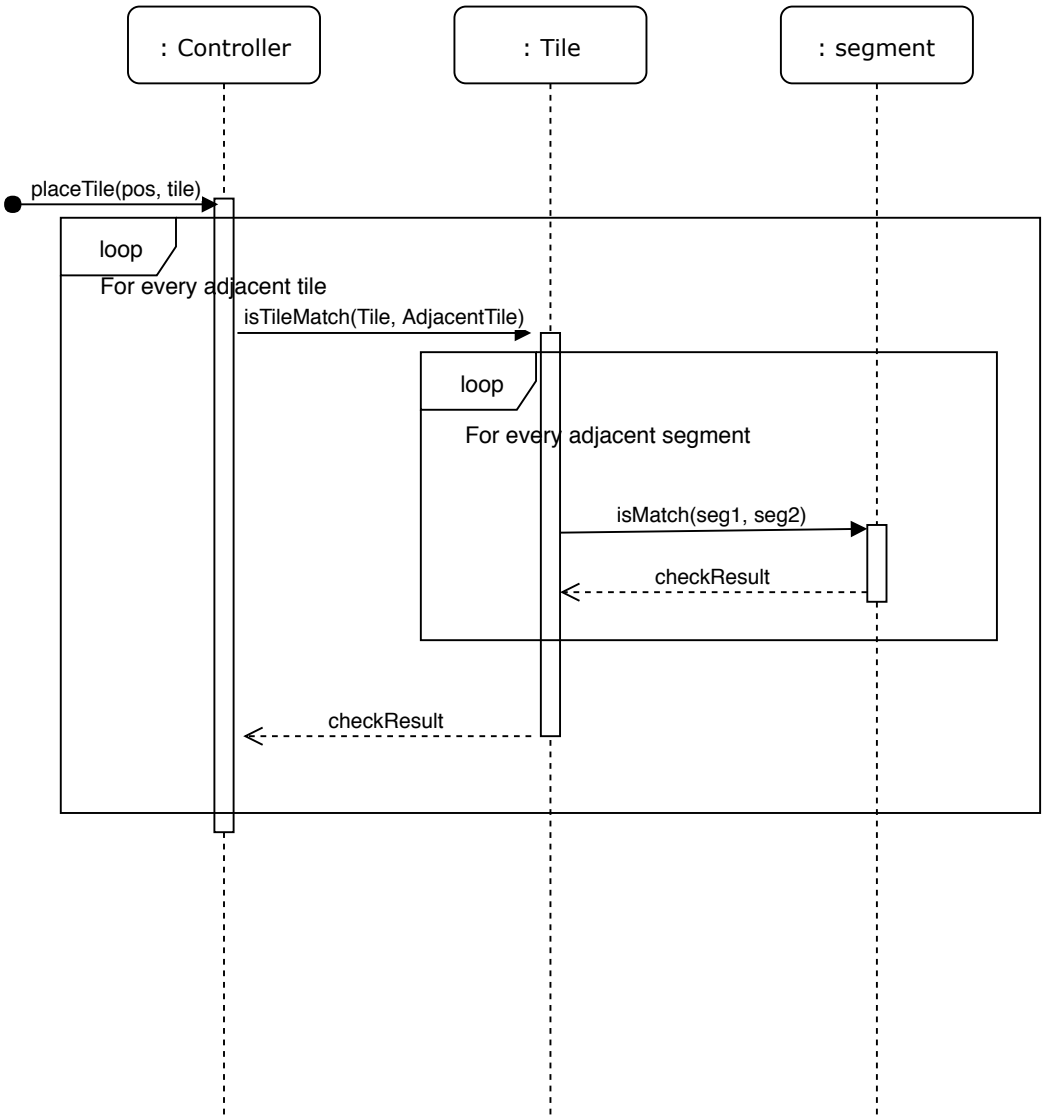
The user attempts to play a tile, without a meeple

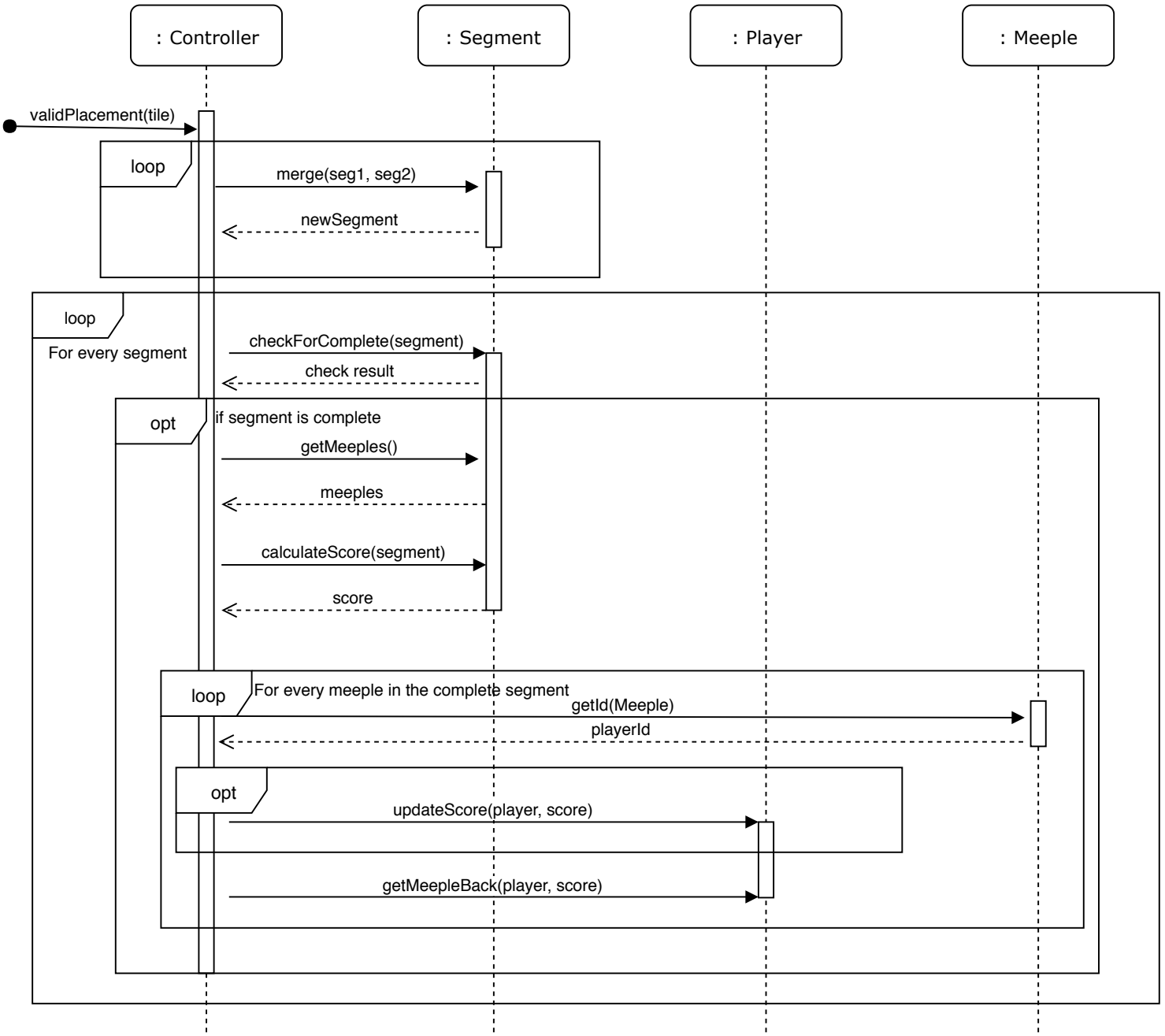
preconditions

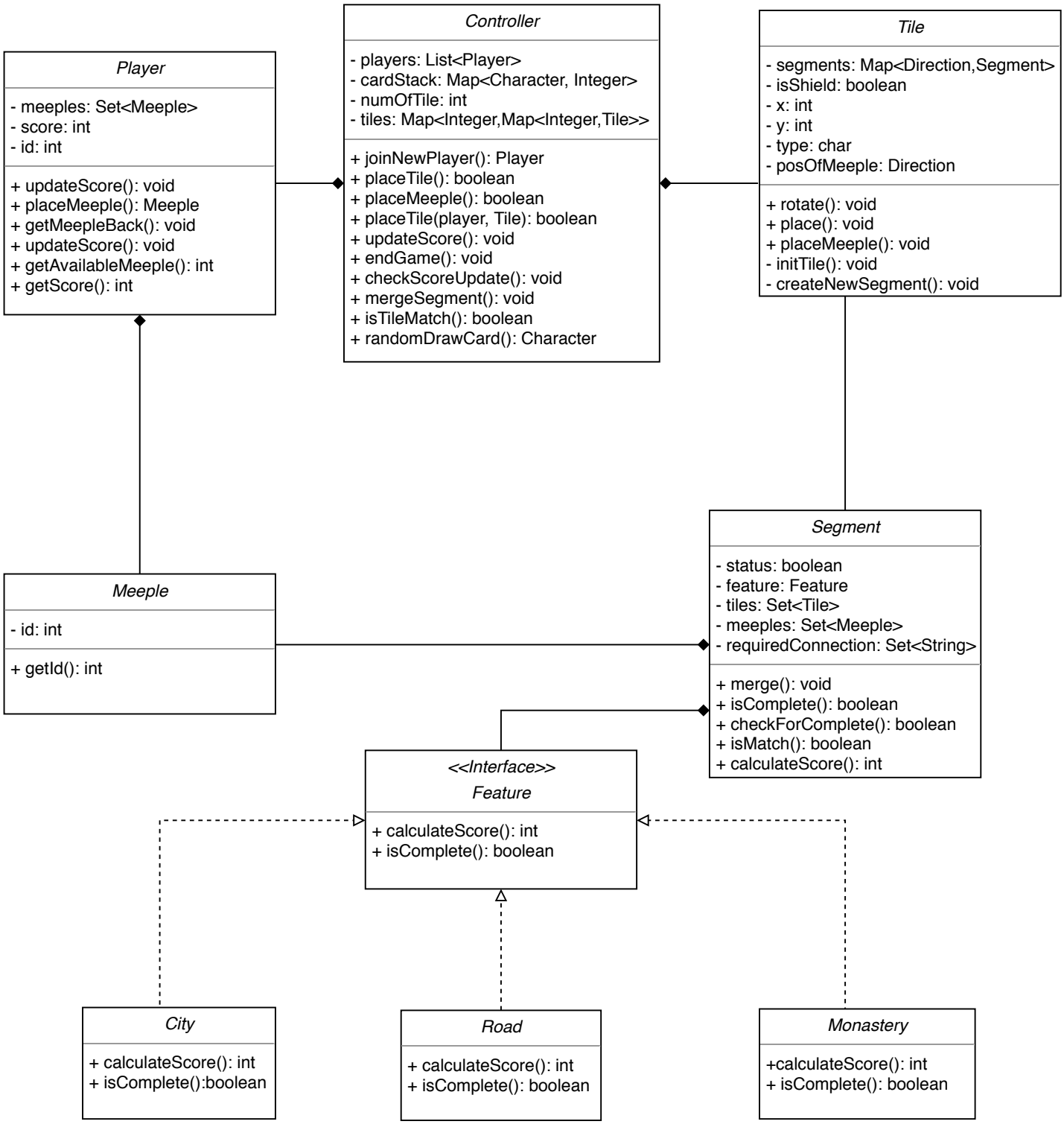
- The user place the tile into a valid position. Here valid position means this tile's image can perfectly combine with adjacent tiles' images. For example, if a tile contains a road, then its adjacent tile must contains a road too, so that they can be connected together.

postconditions

- Tile appears in the game board.
- If this tile's placement can complete a segment, like it results in a new closed road or city or monastery, and there exists at least one meeple in this newly complete segment, the score board will update. Besides, if this complete segment contains meeples, they will return to their owner.


: Controller
: Tile
: segment
placeTile(pos, tile)
loop
For every adjacent tile
isTileMatch(Tile, AdjacentTile)
loop
For every adjacent segment
isMatch(seg1, seg2)
checkResult
checkResult


: Controller
: Segment
: Player
: Meeple
validPlacement(tile)
loop
merge(seg1, seg2)
newSegment
loop
For every segment
checkForComplete(segment)
check result
opt
if segment is complete
getMeeples()
meeples
calculateScore(segment)
score
loop
For every meeple in the complete segment
getId(Meeple)
playerId
opt
updateScore(player, score)
getMeepleBack(player, score)


Player
- meeples: Set<Meeple>
- score: int
- id: int
+ updateScore(): void
+ placeMeeple(): Meeple
+ getMeepleBack(): void
+ updateScore(): void
+ getAvailableMeeple(): int
+ getScore(): int
Controller
- players: List<Player>
- cardStack: Map<Character, Integer>
- numOfTile: int
- tiles: Map<Integer,Map<Integer,Tile>>
+ joinNewPlayer(): Player
+ placeTile(): boolean
+ placeMeeple(): boolean
+ placeTile(player, Tile): boolean
+ updateScore(): void
+ endGame(): void
+ checkScoreUpdate(): void
+ mergeSegment(): void
+ isTileMatch(): boolean
+ randomDrawCard(): Character
Tile
- segments: Map<Direction,Segment>
- isShield: boolean
- x: int
- y: int
- type: char
- posOfMeeple: Direction
+ rotate(): void
+ place(): void
+ placeMeeple(): void
- initTile(): void
- createNewSegment(): void
Meeple
- id: int
+ getId(): int
Segment
- status: boolean
- feature: Feature
- tiles: Set<Tile>
- meeples: Set<Meeple>
- requiredConnection: Set<String>
+ merge(): void
+ isComplete(): boolean
+ checkForComplete(): boolean
+ isMatch(): boolean
+ calculateScore(): int
<<Interface>>
Feature
+ calculateScore(): int
+ isComplete(): boolean
City
+ calculateScore(): int
+ isComplete():boolean
Road
+ calculateScore(): int
+ isComplete(): boolean
Monastery
+calculateScore(): int
+ isComplete(): boolean

Design Explanation

Overview

I designed a class GameController to represent the game system. And this class contains a series of placed tiles. Besides, I designed the class Segment to represent if one feature has already completed or not. Every time, when a tile is placed, it will find some existing segments to join or create some new segments and call related functions to check if the joined segments are complete or not. If the placed tile can help complete an incomplete segment, then, the system need to update players' score. In the next sections, I'll explain my design based on several scenarios.

New Player Join Game

We can create a new Player class to represent a new user. And then use the "joinPlayer" function to add this new player into the game system. In the constructor of Player class, it will automatically create 7 Meeples.

Place Tile and Meeple

The GameController uses a map to store how many tiles for every type are still remaining in the tile stack. AndI implement a function to randomly draw the card from the stack. After receiving the tile, the player can rotate the tile as it wants by calling the "rotate" function and need to call the "placeTile" function. In the function "placeTile", the system will firstly check if this placement is valid. If it is valid, the system will insert the tile into the board, and then merge segments if they are adjacent have the same feature. Besides, the player can place the meeple after the successful placement of the tile by calling the function "placeMeeple" directly.

Complete features

After the placement of a tile or a meeple, the controller needs to check if there is any complete feature. In my design, I use the class Segment to represent a feature and its related tiles. It has two status, complete and incomplete. Every time when we place a tile, the system can create new segment for this tile and call the function "mergeSegment" to merge all of the segments that are adjacent and have the same feature. Then, after adding the segments, the GameController class can call the "checkScoreUpdate" function to check if there is newly complete segments and update scores if there are meeples in this segment. As for how to check if a segment is complete, I create a set named "requiredConnection" in the segment class. This set contains all of the coordinates to finish the segment, if this set is empty, then we can say that this segment is complete.

Update scores

If there is feature complete, or at the end of the game, the GameController updates scores for all of the players by calling the "updateScore" function. In more specific implementation, the complete segment will first execute the "calculate" function to calculate the scores and return it to the controller so that the controller can modify corresponding players' scores.

Design Pattern

I used the idea of strategy pattern to design the "Feature". I firstly build an interface named "Feature" and implement 3 different kinds of feature. They all have two functions, one named "calculateScore" which is used to calculate the score for the complete segment, the other named "isComplete" is used to check if the segment is complete. Besides, I use a Controller class to controller the whose system. It acts like a pivot that connects different modules and classes.

GUI Design

- I divided the game into 2 main panels. The first panel is "MenuPanel". This panel contains buttons like "new game" button, "add player" button and "rotate" button. Besides, this panel contains a score board which shows every player's score and available meeples. What's more,

there is a preview image for the current tile, so that players can see the tile they want to place. The second panel is "BoardPanel" which shows the board of the game.

- Based on my design, the board will only show available positions to place the tile. To be more specific, at the beginning of the game, there is only button for the player to place the first tile. Then, every time when players place a new tile, it will automatically show more buttons which are adjacent to the placed tile. These buttons can represent available positions for players to choose.
- Since the size of panel is limited, but the board of the game may be very large. I choose to use JScrollPanel and design an automatic expanding pattern. That is, when the tile arrives at the border of the board, the system will expand the board to create more available spaces to use.
- When the tile is used up, the game will show the final winner in a dialog box.