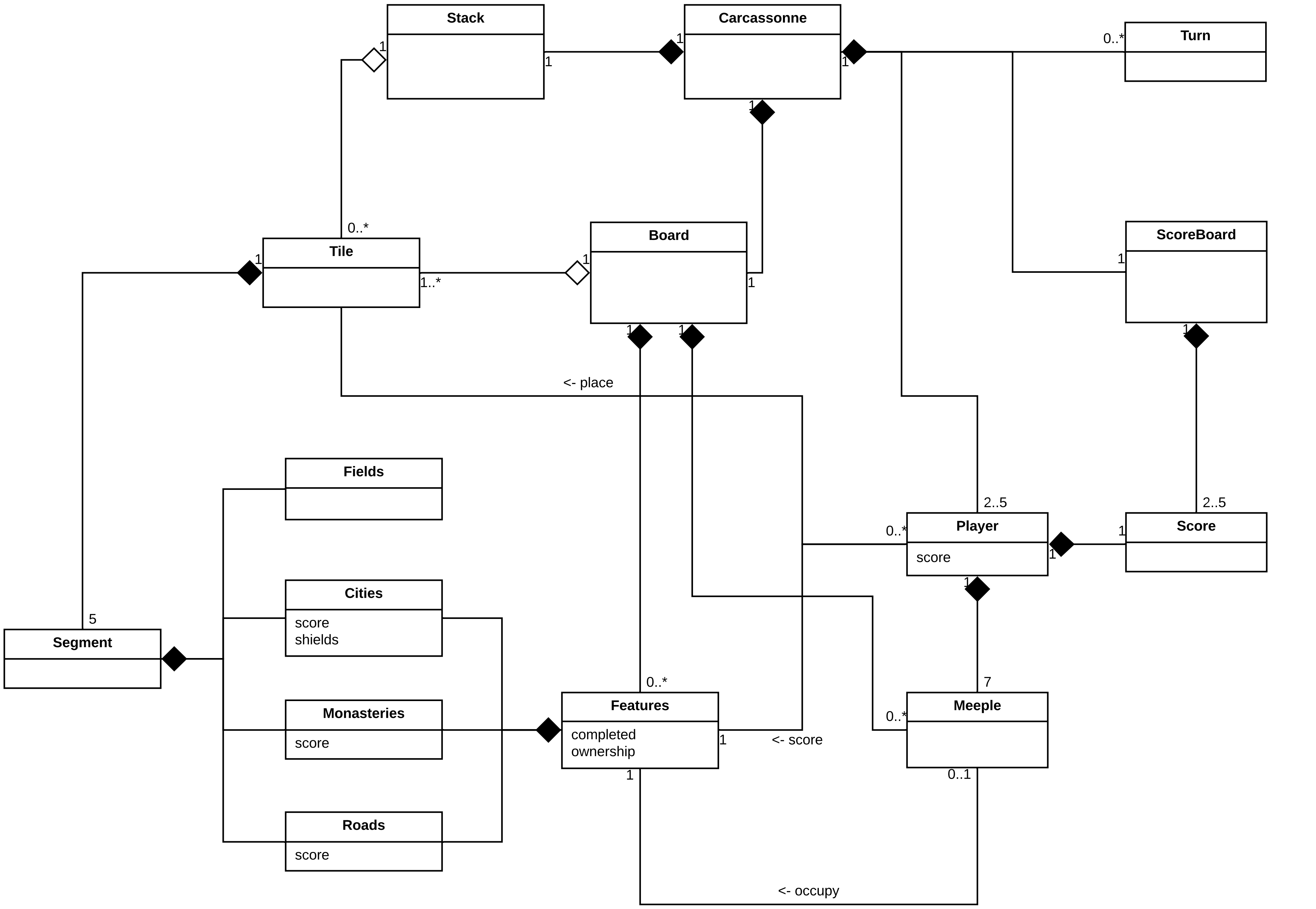
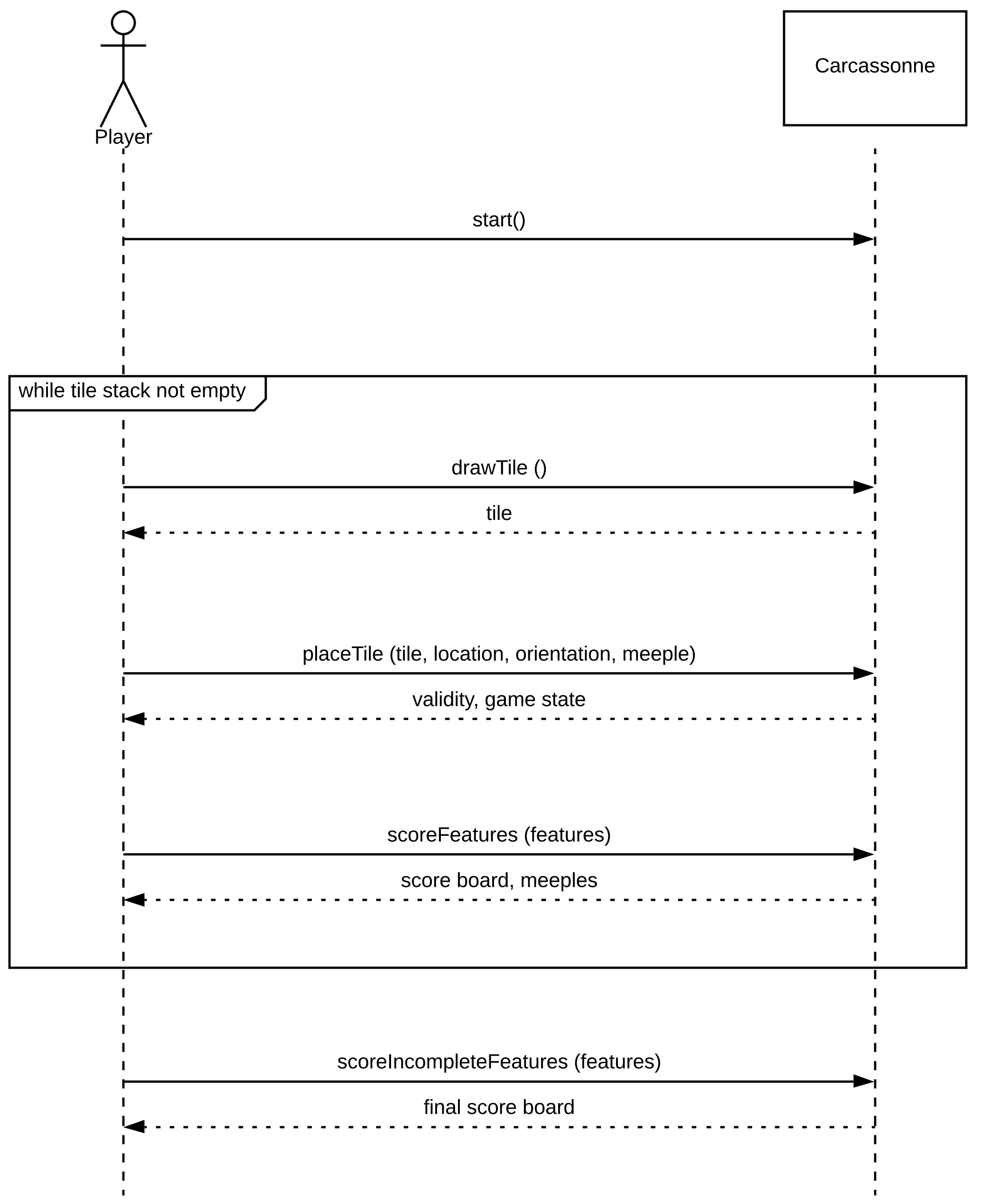
F19 214 HW4a
DOMAIN MODEL
////////////////////////////////////



Ledgend:
-> direction of action

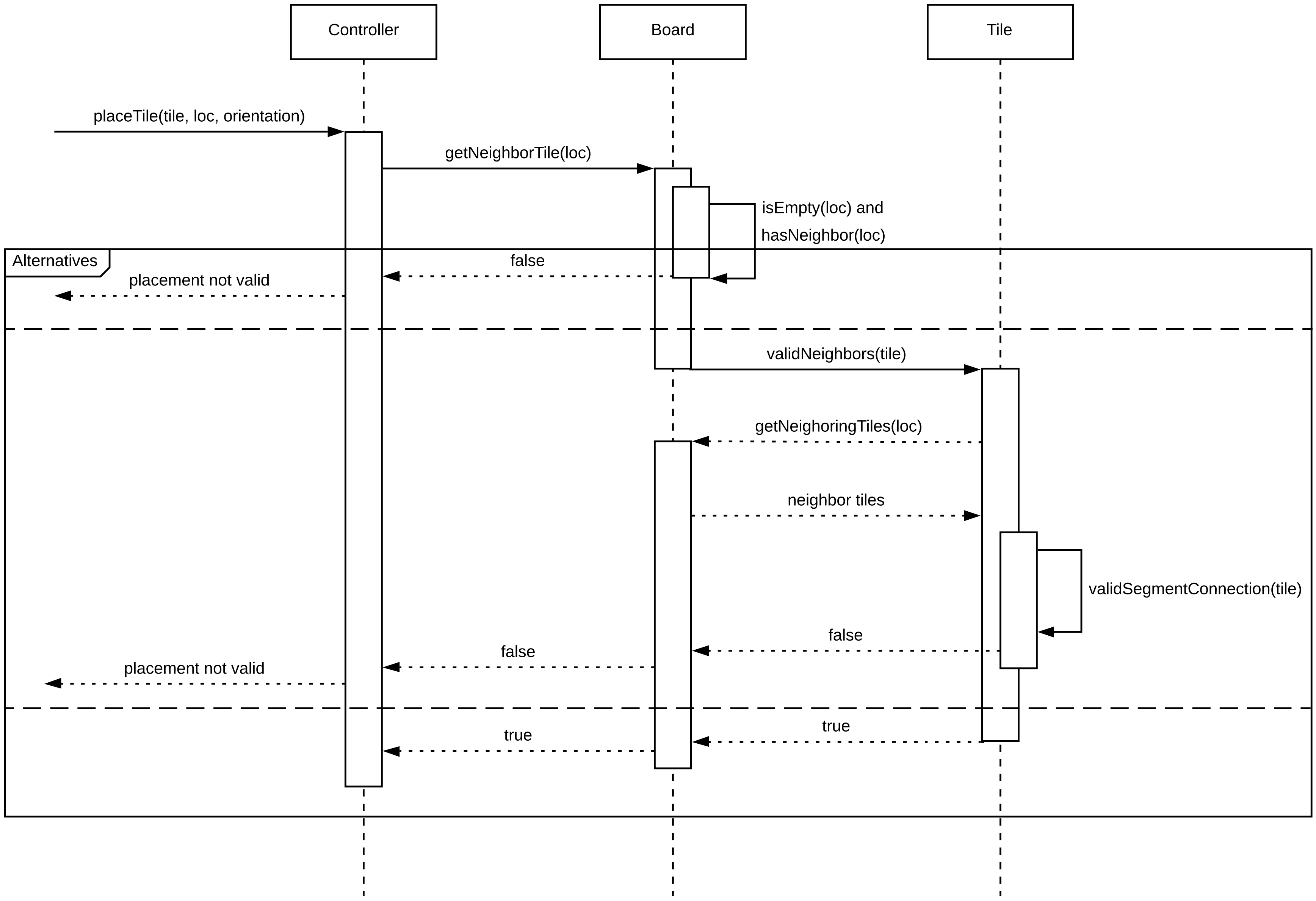
F19 214 HW4a
SYSTEM SEQUENCE
////////////////////////////////////



OPERATION:	The user attempts to place a tile, without a meeple.
PRECONDITIONS:	It is the player's turn to play. The location of the attemted placement is unoccupied by tiles. The segment(s) on the tile to be placed and their corresponding neighboring segments maintains continuity.
POSTCONDITIONS:	The new tile and the location of its placement was added to the game board. Additionally, had any segments on the new tiles completed one or more features, each completed feature's owner received a correspondig score which was added to their running score in the score board. The meeple(s) occupying any completed feature are no longer associated with the features and return to their corresponding players meeple supply.

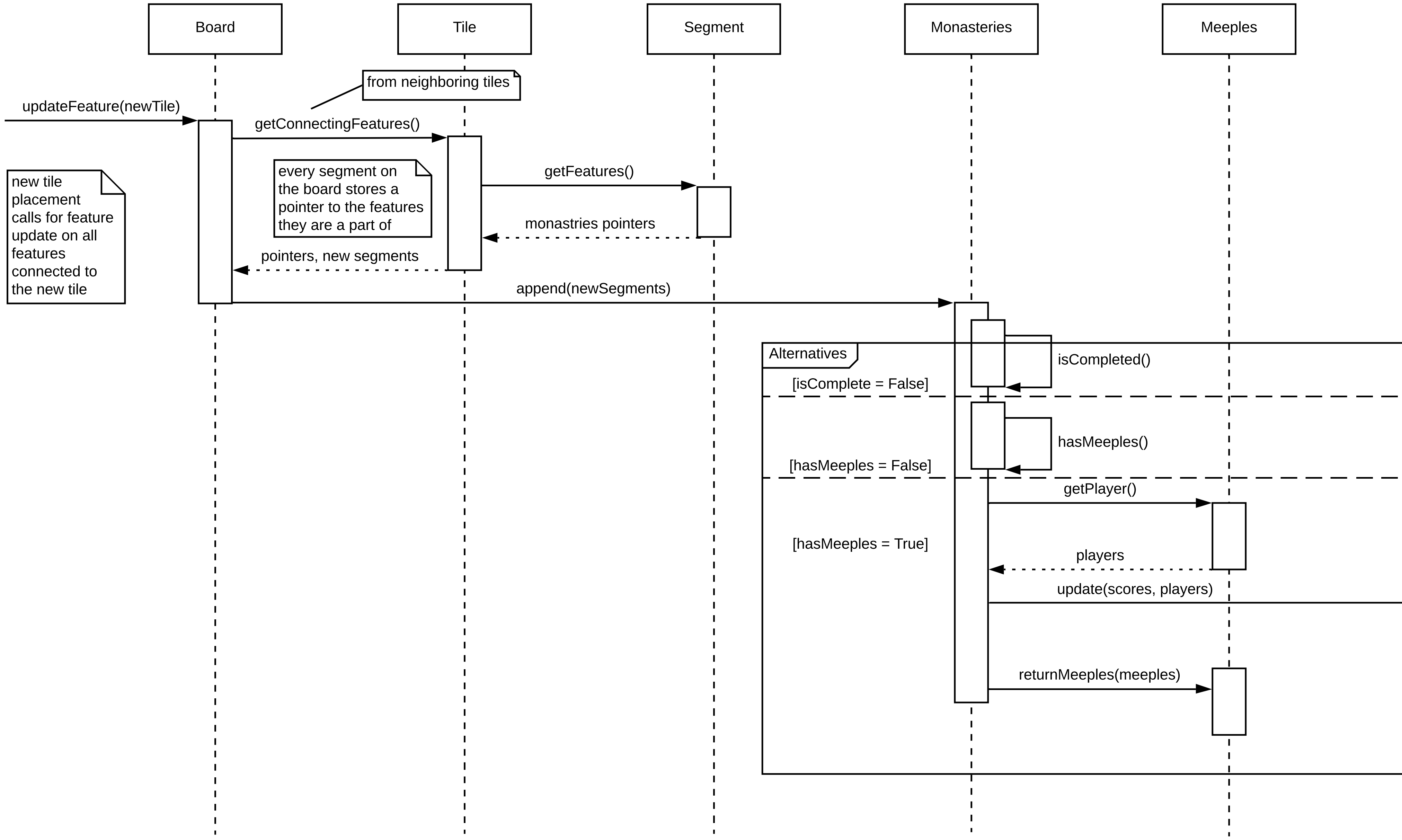
F19 214 HW4a
INTERACTION DIAGRAM
///////////////////////////////////

INTERACTION: Validation of tile placement without a meeple

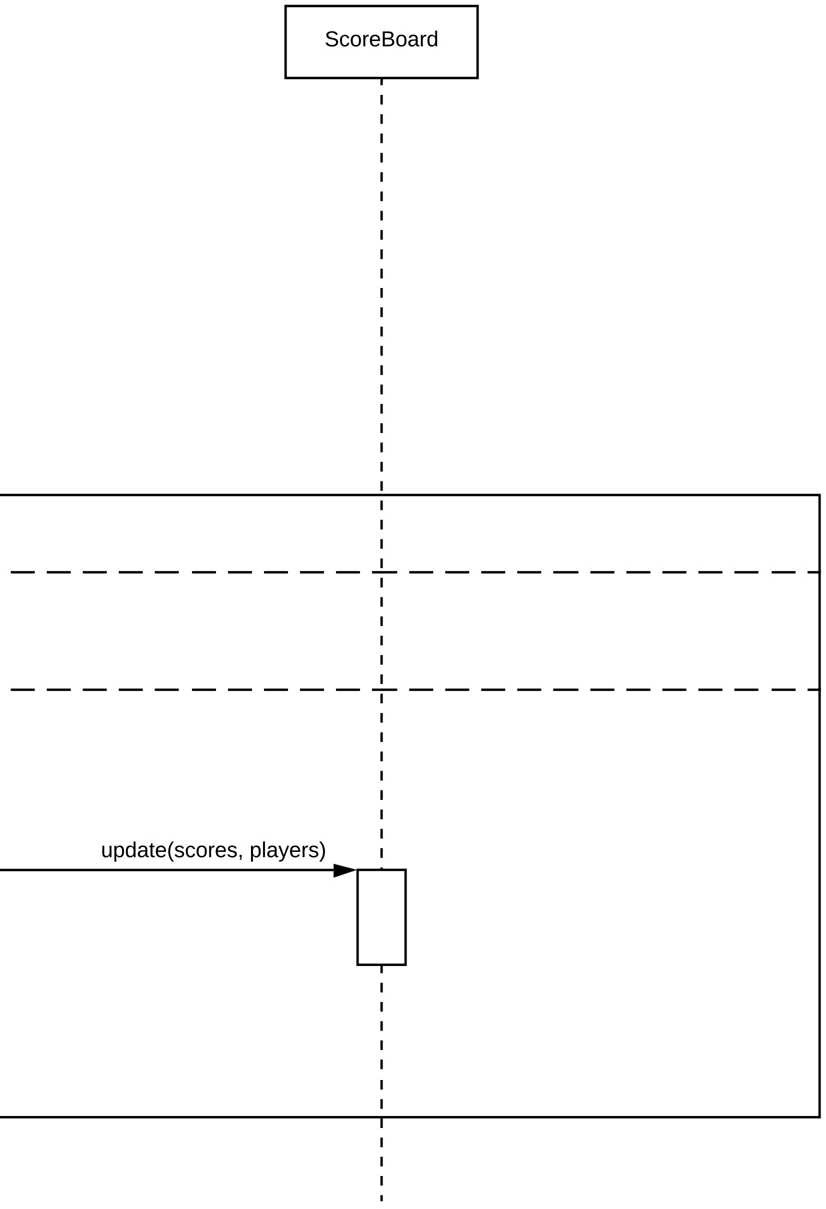


F19 214 HW4a
INTERACTION DIAGRAM
////////////////////////////////////

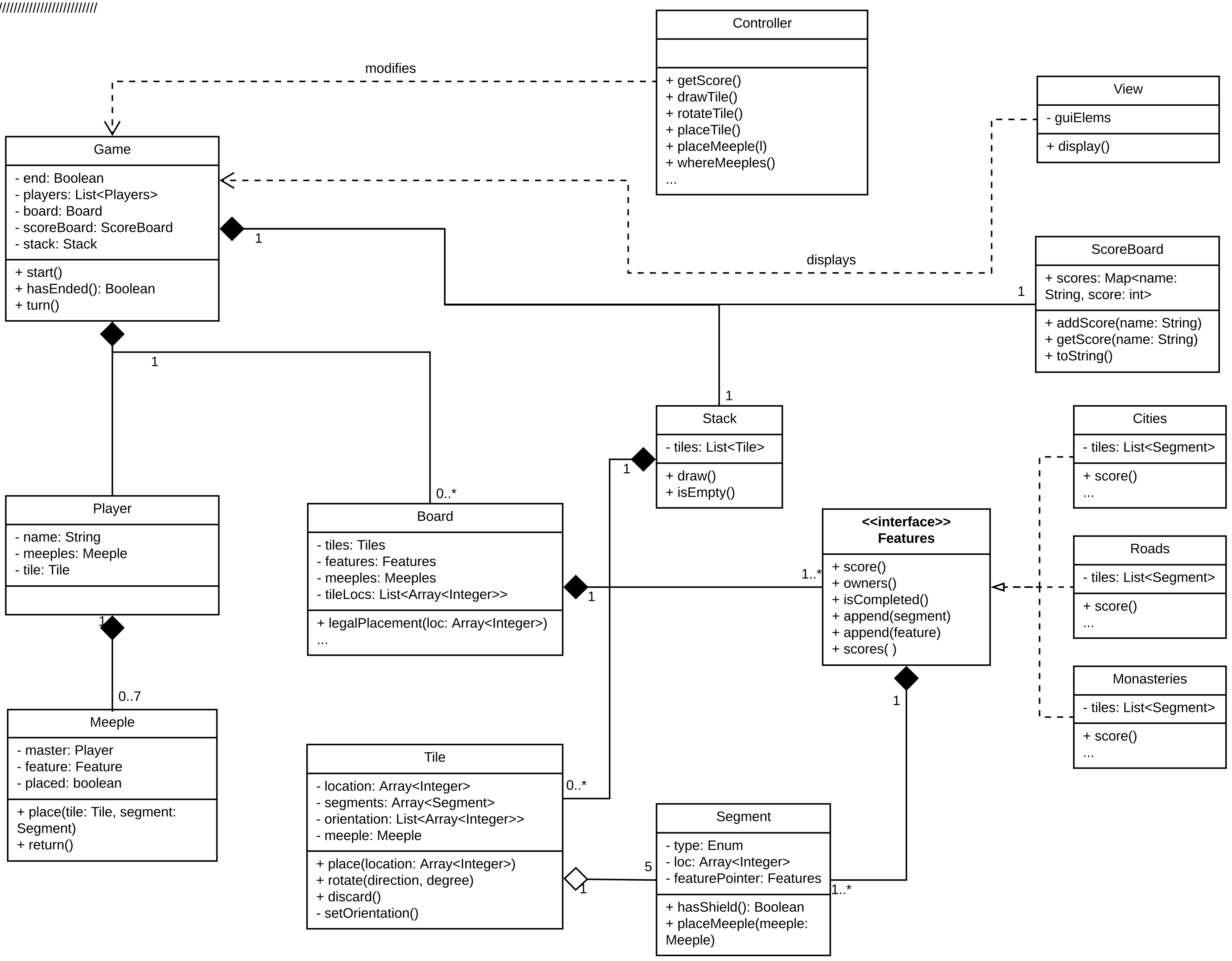
INTERACTION: Game detects newly completed, previously played monasteries, determines whether they contain meeples, scores the monasteries as needed, and returns any scored meeples to their players.



INTERACTION: Game detects newly completed, previously played monasteries, determines whether they contain meeples, scores the monasteries as needed, and returns any scored meeples to their players.



F19 214 HW4a
OBJECT MODEL
/////////////////////////////////////



The overall architecture of the design implements the conventional Model-View-Controller framework, where the model is responsible for storing all internal data of the game while the controller receives requests from users and modifies the model and the view renders the model data into graphical elements. For Part A of the homework, I will focus on the design of the model.

The design of the object model maintains a level of abstraction that follows the domain models as closely as possible in order to minimize the representational gap. All major classes have counterparts in the domain model, such as Tile, Meeple, Board, Segment and Features. Using the principle of encapsulation, each object is responsible for storing its own set of data and providing appropriate methods for modifying those data. For instance, the board is composed of a set of played tiles and is responsible for maintaining a record of each tiles locations, finding neighboring tiles given a board location as well as all of the features currently on the board. On a smaller scale, the tile is responsible for storing all of its segments and keeping track of their orientation while providing methods for its placement and rotation. To address the varying completion checking and scoring mechanisms of different features, the design employs a strategy pattern where each feature type implements a common feature interface while implements differing checking and scoring algorithms under the hood. The use of strategy pattern also decouples each features and ensuring future extensibility when new features are to be added.

The following elaborate on the design by addressing an important scenario in the game. For instance, suppose a valid tile placement (without meeple) leads to the completion of one or more features, how does the game detect feature completions and support different scoring methods? Placing a tile is essentially appending new segments to existing features since existing segments from a tile, once placed, constitutes features that are incomplete. Since features are composed of segments, each segment corresponds to the features that are a part of. Thus as a new placement is made, the board can query the placement location and collate a set of existing segments connecting to the new segments. From these existing segments, board can simply call the append and isComplete method common to all features. Each feature then execute its own completion checking algorithm. Should the feature be completed, it then automatically calls its own scoring methods, again implemented differently based on different features, check if it contains meeples (a field inside features) and returning scores and meeples to its players correspondingly.