UML changes explanation:

Class	Changed	Explanation (Why)
GameModel	Additional logic was added to allow for the use of proper scoring using the method scorePlacedTiles().	Properly handles both a player and an AI player's turn, and correct scoring logic including premium tiles.
GameController	No significant changes between milestones 2 and 3.	-
GameFrame	Added specific colours for each of the 4 premium tile spaces.	Allows for a visual representation of the premium tiles for easier gameplay.
GameView	Added a method to ask player for a value to assign to a blank tile.	Allows players to properly use blank tiles to fill in for any letter in a legal scrabble word. Maintains a score of 0 for blank tiles used on the gameboard.
AIPlayer	Created as a subclass of Player, with methods to automate move generation to support AI gameplay.	Added to handle logic for AI players in the game, including scoring and containing the specified hands for each player.
Player	Fixed methods for word validation (playWord(), allowing for proper validation of word placement and connectivity).	Changed to provide more correct scrabble gameplay.
Move	Added to represent a candidate move for AI with placedTiles and score	Changed to allow the AI players to function by creating moves based off the human player logic.
Board	Added methods extractPattern(),findAnchors()	These methods were added in order to be able to create logic for the AI player to base it's moves on.
Tile	Now supports blank tiles with new methods isBlank(), assignedLetter()	Functions with blank tiles to determine if a tile is blank, and to keep tiles registered internally as blank for scoring purposes (since blank tiles score 0 points regardless of the letter they are replacing).
TileBag	Now calculates score accurately and implements blank tile logic.	Added logic to assign each tile with a specific score value based on the rules of scrabble.

		This is handled through a switch case which will return the score based off the tile letter.
Dictionary	There are no significant changes between milestones 2 and 3.	-

Data structure

Class:

Board:

Data structures used:

2D array: Tile[][]

Explanation:

A scrabble board is always a fixed size, so a 2D array is appropriate as you'd never need to modify the size. It's also ideal as it allows for O(1) access to any cell.

Operations:

placeTile(row, col, tile) updates the 2D array cell
getTile(row, col) retrieves the tile from a specified location
removeTile(row, col, tile) sets the tile back to Null
display() loops over the array
isEmpty() loops over the array

Dictionary:

Data structures used:

ArrayList<String> acceptedWords

Explanation:

the dictionary contains a dynamic list of words loaded from a CSV. ArrayList allows dynamic resizing as words are added or removed, and it provides fast indexed access for checking or iterating through words

Operations:

acceptedWords.add(word) adds a new word
acceptedWords.contains(word) checks if the word already exists

Player:

Data structures used:

ArrayList<Tile> hand
ArrayList<PlacedTile> placedTiles

Explanation:

ArrayList is dynamic, which is important because the number of tiles can change during the game. ArrayList allows indexed access and iteration, which is useful for displaying hand or processing placed tiles

Operations:

hand.add(tile) adds a tile to the player's hand
hand.remove(tile) removes a tile once it's placed

placedTiles.add(new PlacedTile(...)) tracks moves for validation and scoring
showHand() iterates through the ArrayList
playWord() iterates through the ArrayList

PlacedTile:
- Data structures used:
  - no data structures, just 2 int variables and 1 Tile variable
  - Explanation:
    - the Tile object stored in the variable “tile” stores most of the needed information, the row and col int variables store the position where the tile is placed

Tile:
- Data structures used:
  - no data structures just 2 String variables, 1 int for scoring and 1 boolean for blank tiles.
  - Explanation:
    - only needs to store a single string, has a getter function and the String is set on construction of the object

TileBag:
- Data structures used:
  - List<String> tiles
- Explanation:
  - Array list allows for dynamic addition and removal of tiles
- Operations:
  - addTiles(letter, count) populates the bag
  - drawTile() randomly removes a tile and constructs a Tile
  - isEmpty() checks if the bag is empty

GameModel:
- Data structures used:
  - ArrayList<Player> players
  - Dictionary acceptedWords
  - ArrayList<Tiles> placedTiles
- Explanation:
  - Array list is dynamic and ordered, making it easy to iterate through the list for players turns
  - Dictionary is shared between all players and is static as the accepted words don’t change mid game.
  - Array list is dynamic and ordered, which makes it very easy to add tiles into it and remove them for each player’s turn.
- Operation:
  - addPlayer(Player) adds a player dynamically
  - nextPlayer() iterates through the player list to change the current player

The majority of the data structures and code between milestones 2 and 3 is again, very similar. The major differences arise from the way which AI players now interact with the user interface, as the logic for AI players relies on what moves the human players have made on the board. Besides the logic for AI players, very few changes were made to the existing GUI or code that implemented it, besides minor bug fixes and adding the logic for scoring.