Milestone 1 Detailed Description of the Choice of Data Structures and Relevant Operations

In this implementation of the text-based game, Scrabble's multiple data structures represent the states, resources, rules, and flow of play. Each structure was analyzed and designed to allow for easily implemented game features and functions.

Tile and TilePile

Each Tile simply stores one char for the letter and one int for its value. Getters and setters are used to modify and retrieve Tile values and letters. The method *assignValue()* maps the letters to their values using a switch statement ensuring smooth flow and correct representation per Scrabble rules.

TilePile is an ArrayList<Tile> flexible array used to manage the outgoing tiles to players. The TilePile is initialized with a full 'bag' of Tiles, where the quantities of each letter are created and placed in the bag and shuffled using the *addTile()* and *Collections.shuffle()* methods. The *deleteTile()* method is used to return a Tile out of the TilePile, to be added to a player's hand. We chose an ArrayList for its efficient resizing and ease of adding/removing elements, but a LinkedList could also provide a similar implementation.

Board

The board is structured as a 2-D array of size[15][15] storing Tile objects. A simple grid is initialized at the beginning of gameplay with a single random Tile placed in the center, all other tiles are assigned ' ' to be easily identified, stored, and displayed. *setTile(), getTile(), and displayBoard()* are used to place Tiles on specific coordinates, retrieve information about a tile at specific coordinates, and display the whole board to the user. A 2-D array provides direct and fast access to any tile by row and col index, making it suitable for this grid-based game. The fixed size (15x15) aligns with Scrabble's standard dimensions, and the direct indexing of arrays offers efficient access for viewing or placing tiles.

Player

Each player represents three attributes, their rack, points, and name (or playing order). To represent the rack we use an ArrayList containing Tiles which is initially given 7 Tiles from the TilePile. We chose an ArrayList for the rack to allow for easy removal and deletion of Tile objects. For the points we simply store an int, and for the name a String. *addTile() & removeTile()* are called whenever a user places Tiles to maintain the proper consumption and replenishment of game resources. We have helper method *hasTile()* and method *hasAllTiles()* to check if a user has sufficient tiles in their rack to potentially play a word. Finally, a *displayHand()* method is used to output the player's current rack.

Word

For our word validation, it was decided to implement a HashSet to contain 10'000 unique valid words. Using the HashSet fast and efficient add and contains operations ensures fast and reliable verification and initialization with the create *WordBank(), and isWord() methods.*

Game

The game structure contains an array of four players, a TilePile holding all the game Tiles, the game Board to store the layout of the placed words, and a word object to check and verify words before they are created. The *play()* method handles the textual user interface with the user(s) and uses the *canPlaceWord()*, and *placeWord()* methods to allow users to play Scrabble turn by turn, not allowing any invalid words to be placed, nor created adjacently. The *play()* method also consumes Tiles as they are placed by the player while refilling their rack from the TilePile. The game has a rudimentary points system that tracks a player's points as the game progresses, and future implementation for adjacent word points to be tracked and updated as well. Helper methods *verticalAdjacencyCheck()* and *horizontalAdjacencyCheck()* are used by the canPlaceWord() method to verify that any perpendicular or extended words, letters from a newly placed word, are also valid words, by Scrabble game logic.