Checkpoint 2: Move and Combine tiles

In this checkpoint, you would implement the four movement handling methods:

- `processLeftMove`
- `processRightMove`
- `processUpMove`
- `processDownMove`

Those four methods would update the board when the user inputs the corresponding movement key

This worksheet would walk you through how to implement for the left direction and how to apply the same idea to the other directions.

> **Tip**
>
> I encourage you to give it a try yourself first to break down this problem. Read further when you need more guidance

Process Left Move

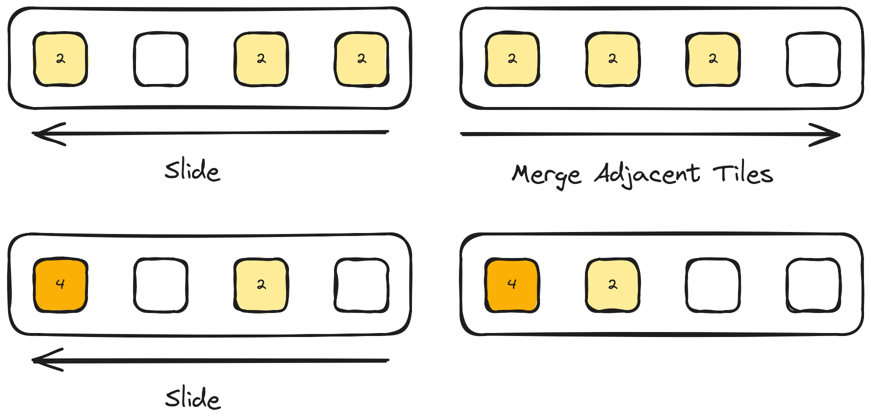
When a user moves left, all the tiles in each row would combine and slide left

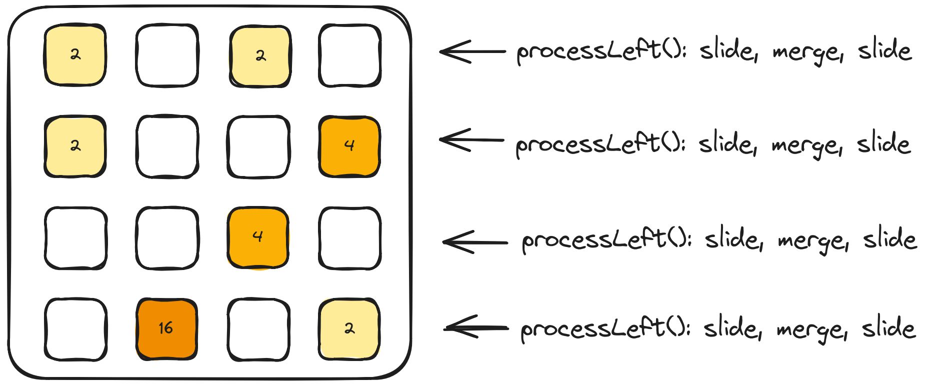
We can see how this may be implemented for a particular row and apply for all rows

Strategy 1

Strategy 1: Slide, Merge, Slide

The idea is to break the move sequence into 3 parts: slide, merge adjacent tiles, and slide again




2
2
processLeft(): slide, merge, slide
2
4
processLeft(): slide, merge, slide
4
processLeft(): slide, merge, slide
16
2
processLeft(): slide, merge, slide

With this strategy, you would break the problem for a particular row into 3 parts:

1. Slide all tiles to the left
2. Merge adjacent tiles
3. Slide all tiles to the left

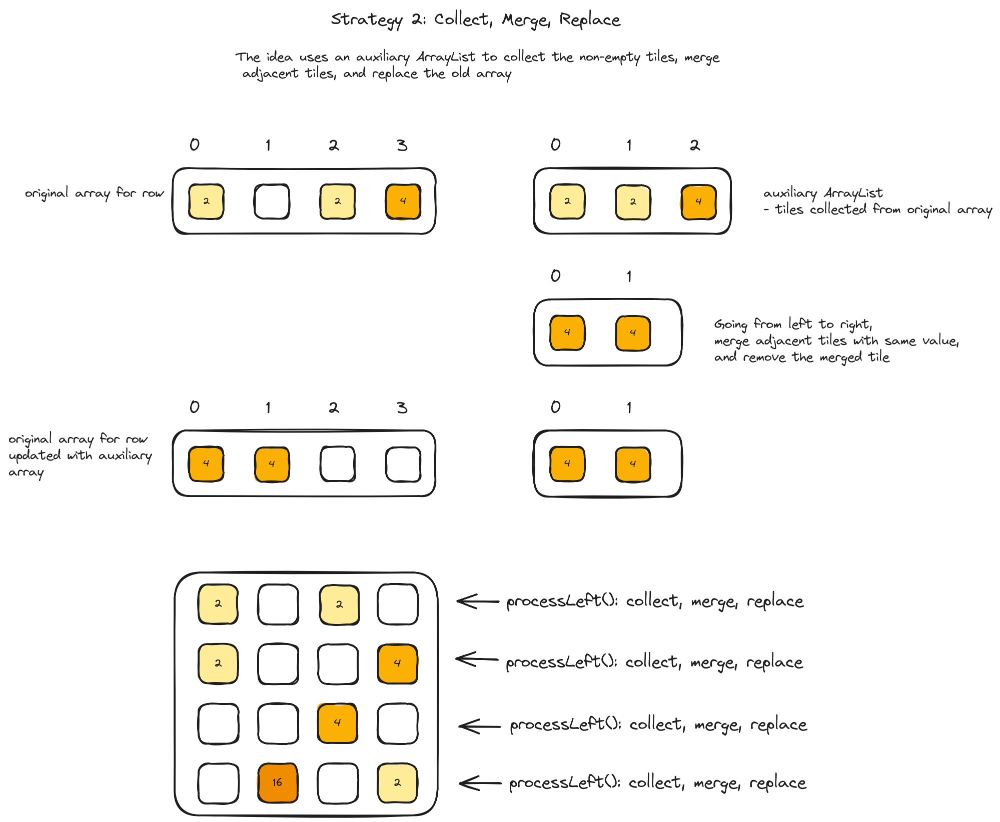
You may create and implement three private utility methods in the `Board` class:

1. `private int[] processLeft(int[] row)`: process sliding left and merging for a particular row.

Return the processed row

2. `private int[] slide(int[] row)`: return a new row with the tiles in the provided `row` slide to the left
3. `private int[] mergeLeft(int[] row)`: merge adjacent tiles in a `row` from left to right

Strategy 2


Strategy 2: Collect, Merge, Replace
The idea uses an auxiliary ArrayList to collect the non-empty tiles, merge adjacent tiles, and replace the old array
original array for row
auxiliary ArrayList - tiles collected from original array
Going from left to right, merge adjacent tiles with same value, and remove the merged tile
original array for row updated with auxiliary array
processLeft(): collect, merge, replace
processLeft(): collect, merge, replace
processLeft(): collect, merge, replace
processLeft(): collect, merge, replace

With this strategy, you would break the problem for a particular row into 3 parts:

1. Collect all tiles into a helper ArrayList
2. Merge adjacent tiles in the helper ArrayList, *removing* the merged tiles in the process from the ArrayList
3. Use the helper ArrayList to replace the original row

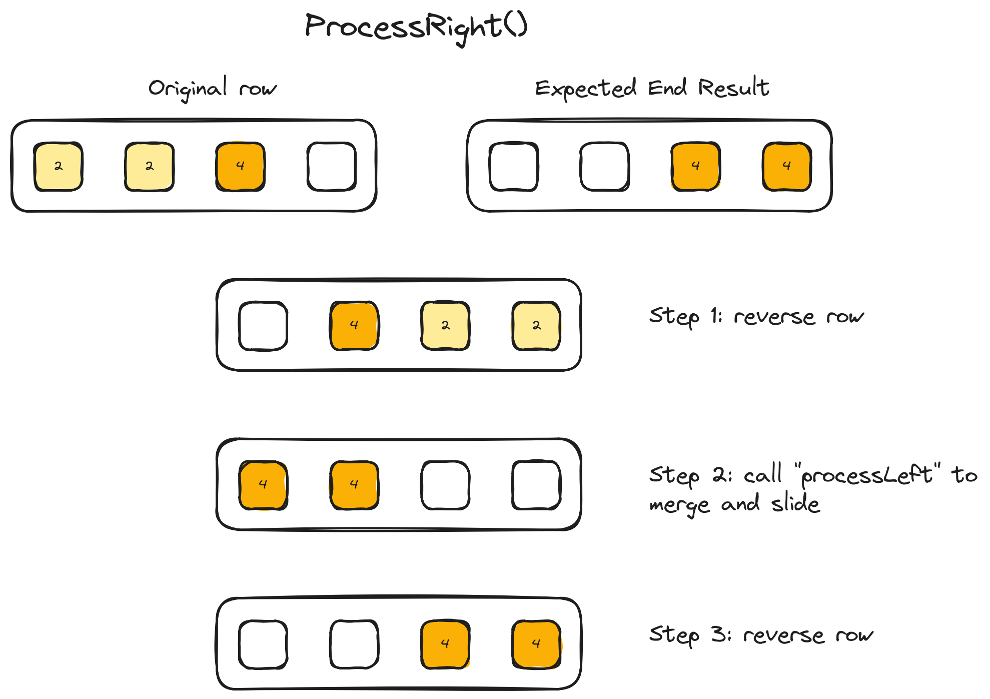
You may create and implement three private utility methods in the `Board` class:

1. `private int[] processLeft(int[] row)`: process sliding left and merging for a particular row. Return the processed row
2. `private ArrayList<Integer> getNonemptyTiles(int[] row)`: return a helper `ArrayList` with the non empty tiles in a row, collected from left to right
3. `private ArrayList<Integer> merge(ArrayList<Integer> row)`: merge adjacent tiles in `row` from left to right, *removing* the merged tiles in the process. Return the merged ArrayList

Process Right Move

Now we've implemented the left move, we may implement the similar logic but for the right move again.

However, this would be a lot of duplicate code. One trick is to think of the right move as the mirror of the left move. You would "flip" or reverse the row array, call `processLeft` on that row, and "flip" back the row



You may create and implement one private utility method in `Board` class:

1. `private int[] reverse(int[] row)`: return a new row with the tiles in the provided `row` reversed

Process Up and Down Move

We may think of column movements `processUp` and `processDown` in terms of the row movements and as opposite of each other

I'd leave you this exercise to figure it out :)