CS 251 Intermediate Programming
Gem Match: Part 1 – Gem Manager

Brooke Chenoweth

Spring 2025

In this project, you will begin work the on a gem[1] matching game. You will be working on internal logic and bookkeeping independent of the game GUI which you will write in the next assignment.

Game Description

In a match-three game such as Bejeweled[2] the objective is to swap one gem with an adjacent gem to form a horizontal or vertical chain of three or more gems of the same color. Gems disappear when chains are formed and gems fall from the top to fill in gaps. Sometimes chain reactions, called cascades, are triggered, where chains are formed by the falling gems.

Problem Specification

You are going to create a `GemManager` class to handle your gem bookkeeping. You will demonstrate its functionality in a `GemManagerTest` class. (You may want some additional classes/interfaces to accomplish this, but I leave that design up to you.) The GemManager should be independent of any future GUI code.

`GemManager`

Although the actual game will likely have a fixed board size, your class should be able to handle arbitrary positive values for the number of rows and columns.

Your class must provide the following functionality.

- Construct a new manager with given number of rows and columns.
- Fill the board with gems.
  - The final game will use random gems, but for this test program you should have reproducible results each time it is run.

[1] or candy, etc.

[2] See `https://en.wikipedia.org/wiki/Bejeweled`

  - One simple way to have "random" behaviour that can be repeated is to use a `Random` object that has been initialized with a fixed seed.
  - The initial board should have no chains of matching gems. One way to accomplish this is to use the gem matching and removing code mentioned below, removing matching gems and dropping in new ones until there are no more matches.

- Override `toString` to provide a string representation of your board.
- Select a gem at a given position and swap it with an adjacent one. Determine which gems will be part of a horizontal and/or vertical chain of at least 3 gems of the same color.
- Remove gems from the board and refill with new ones.
  - Given a chain (or chains) of matching gems, remove them from the board.
  - Any gems above the removed gems should fall down to fill the empty space.
  - Empty space at the top after gems drop should be filled with new gems.
  - Check for any matching chains formed as a result.

`GemManagerTest`

The purpose of this testing class is to demonstrate to us that your gem manager works correctly, so you need to put it through its paces.

- You should create at least two `GemManager` objects with separate sizes and fill patterns. Demonstrate that they operate independently of each other. (I want to make sure that you aren't locked in to just one configuration and that you aren't making things static inappropriately.)
- Test swapping and dropping gems on each manager.
  - Print out the string representation of the manager before you begin.
  - Select a gem and swap it with an adjacent one.
    1. Print out the gem manager after the swap.
    2. Print out how many gems are in chains and should be removed. (May be zero if there are no chains formed by the swap. In the actual game, you may not want to allow such a swap to proceed.)
    3. Remove the matching chains, drop gems, and refill with new ones.
    4. Print out the resulting gem manager.
    5. Print out how many gems need to be removed as a result of new matching chains formed by the dropped gems.
    6. Repeat from 3 until no more matches are formed.
    7. Swap gems another time or two. Don't use the same positions every time.

  - After demonstrating each manager, repeat the tests with the first manager again to prove that it is independent of the second.

- Don't just copy and paste the same test code for each manager. Structure your code sensibly with methods rather than writing one giant `main` method.

- Your testing output should come from `GemManagerTest`, not `GemManager`. Don't put console output in the middle of your gem manager logic.

I am providing you with example output from my testing code to give you an idea what I'm expecting this to look like.

Turning in your assignment

Submit your `GemManager.java`, `GemManagerTest.java`, and any other source files you used for the project to Canvas.