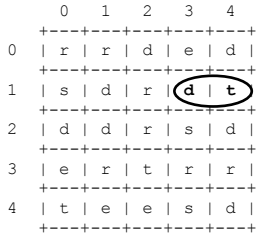
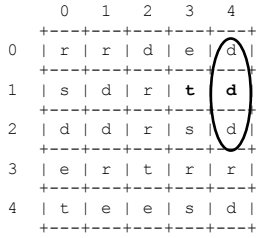
Task - Begemed

Begemed is a casual game which is played on ruled grids. The player is required to swap a gem in one of the four possible directions, namely “up”, “down”, “left”, “right”; after the swap, if a row or a column of 3 or more gems are formed, it’s considered a valid move and the connected gems will be destroyed. Otherwise, it’s considered an invalid move. Note that diagonal directions are not counted.

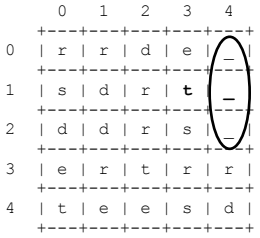
Below are some examples of the game demonstrated in a 5x5 grid. The letters "d", "s", "t", "r", "e" represents Diamond, Sapphire, Topaz, Ruby, Emerald respectively.



1. A swap between "d" at (1, 3) and "t" at (1, 4) is considered a valid move.

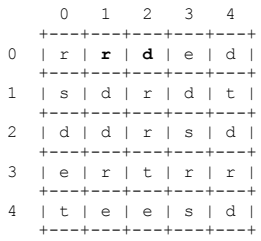


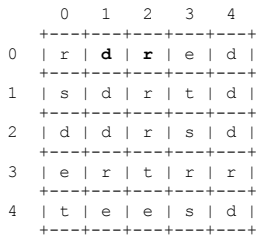
2. The grid will look like this after the swap. There are three "d"s found at (0, 4), (1, 4) and (2, 4) forming a column and they will be destroyed.

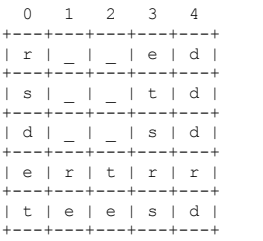


3. After the gems are destroyed, the respective grid cells will be filled with "_" temporarily.

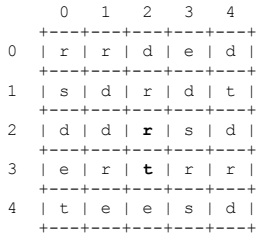
Some other valid swaps are:

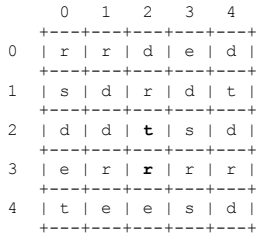


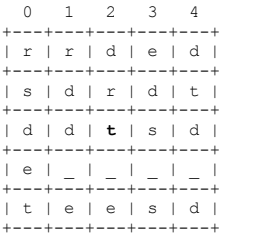




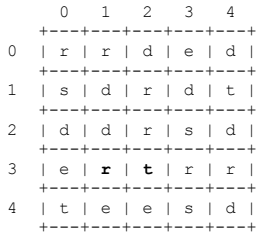
"r" at (0, 1) and "d" at (0, 2) can be swapped.

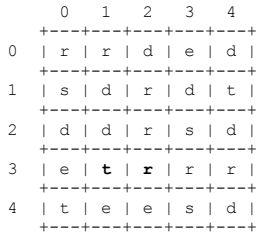


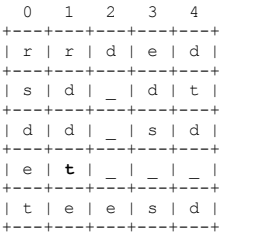




"r" at (2, 2) and "t" at (3, 2) can be swapped.







"r" at (3, 1) and "t" at (3, 2) can be swapped.

You are tasked to create a text-based interactive "Begemed" game in the following tasks.

Task 4.1

Implement `Begemed` class according to the UML class diagram and attributes/methods specifications given.

Board
- board: list
+ Board(size: int) + new_game(board: list) + check_connection(row: int, col: int): boolean + find_valid_moves(row: int, col: int): list + display(hint: boolean=False)

Attributes/Methods	**Specification**
`board: list`	`board` is a 2-dimensional `list` hosting the gems inside each grid.
`Board(n: int)`	`n` is the size used to define the dimension of the `board`. The `board` should be initialized to a (`n x n`) 2-dimensional list filled with string "_".
`new_game(new_board : list)`	`new_game` takes in a `list` named `new_board` and assign it to the class attribute `board`. *This method is just a temporary solution which help you in initial coding and debugging. In a later task, there will be further instructions to update its implementation.*
`check_connection( row: int, col: int): boolean`	`check_connection` takes in the `row` and `col` value of a particular gem, then check if there is a connection of 3 or more gems of the same type in its horizontal or vertical direction. Return `True` if such a connection is found, and `False` otherwise.

Attributes/Methods	Specification
`find_valid_moves(row: int, col: int): list`	`find_valid_moves` takes in the `row` and `col` value of a particular gem, then attempt to swap in the four directions (up, down, left, right). If there is a new connection of 3 or more gems of the same type formed, record as a valid movement. Return a `list` containing all valid movements. An empty `list` is to be returned if no valid movement is found. For example: `find_valid_moves(0, 2)` is called, the function should return `['d', 'l']` because when down swap or left swap is performed on gem at `(0, 2)`, a new connection of 3 or more gems of the same type will be formed.
`display(hint: Boolean=False)`	`display` will print out the `board` according to the sample format given. Take note that the `size` of the board can be changed and hence the grid outline should be dynamically adjusted according to its `size`. For example: `hint` is an optional argument with a default value of `False`. If `hint` is set to be `True`, the gems with valid moves should be highlighted by using the **uppercase** letters, and the valid moves for the **coordinates** and **directions** should be displayed too.

```
    0   1   2   3   4
  +---+---+---+---+---+
0 | r | r | d | e | d |
  +---+---+---+---+---+
1 | s | d | r | d | t |
  +---+---+---+---+---+
2 | d | d | r | s | d |
  +---+---+---+---+---+
3 | e | r | t | r | r |
  +---+---+---+---+---+
4 | t | e | e | s | d |
  +---+---+---+---+---+
```

	For example: (0, 1) ['r'] (0, 2) ['d', 'l'] (1, 2) ['u'] (1, 3) ['u', 'r'] (2, 2) ['d'] (3, 0) ['d'] (3, 1) ['r'] (3, 3) ['l'] 0 1 2 3 4 +---+---+---+---+---+ 0 | r | R | D | e | d | +---+---+---+---+---+ 1 | s | d | R | D | t | +---+---+---+---+---+ 2 | d | d | R | s | d | +---+---+---+---+---+ 3 | E | R | t | R | r | +---+---+---+---+---+ 4 | t | e | e | s | d | +---+---+---+---+---+

Task 4.2
Write a texted based menu which has the following options. Validation of the user input is needed.

```
Choose an option below:
1) Validate Move
2) Toggle Hint Mode
3) Move the Gem!
4) New Game
5) Exit
```

The descriptions for the options can be found below.

Option	**Descriptions**
`Validate Move`	Ask user to input a set of `row`, `col` and `direction`. Check and feedback if this swap is valid.
`Toggle Hint Mode`	For every new game, the `hint` mode by default should be `off`. Use this option to toggle the `on` and `off` state of `hint` mode. If `hint` mode is `on`, the menu interface should automatically highlight the gems with valid moves and print out a list of the coordinates together with its valid movement directions.
`Move the Gem!`	Move a gem in a chosen direction. *Note that the related class method will only be implemented in the **next task**. For the current menu, you*

	only need to take in user input for `row`, `col` *and* `direction`, *but no further action needs to be taken.*
`New Game`	Start a new game and reset `hint` mode to be off.
`Exit`	Exit program.

Task 4.3

Update the class `Begemed` with the following methods. Note that this task is time consuming and only worth **2 marks**.

<table>
<tr><th>Methods</th><th>Specification</th></tr>
<tr><td>new_game(n: int)</td><td>new_game will now take in a size of n and randomly generate a n x n board of gems. The newly generated gems should not have any connection of 3 or more gems with the same type.</td></tr>
<tr><td>move_gem(row: int,
col: int,
direction: string)</td><td>move_gem should take in a gem position and direction. If the swap is a valid move, detect any newly formed connection of 3 or more gems with the same type and cancel them.

<table>
<tr><td><pre>
    0   1   2   3   4
  +---+---+---+---+---+
0 | r | r | d | e | d |
  +---+---+---+---+---+
1 | s | d | r | d | t |
  +---+---+---+---+---+
2 | d | d | r | s | d |
  +---+---+---+---+---+
3 | e | r | t | r | r |
  +---+---+---+---+---+
4 | t | e | e | s | d |
  +---+---+---+---+---+
</pre></td><td><pre>
    0   1   2   3   4
  +---+---+---+---+---+
0 | r | r | d | e | d |
  +---+---+---+---+---+
1 | s | d | _ | d | t |
  +---+---+---+---+---+
2 | d | d | _ | s | d |
  +---+---+---+---+---+
3 | e | t | _ | _ | _ |
  +---+---+---+---+---+
4 | t | e | e | s | d |
  +---+---+---+---+---+
</pre></td></tr>
<tr><td>Swap "r" at (3, 1) with "t" at (3, 2)</td><td>Gems connected with 3 or more of the same type are cancelled.</td></tr>
</table>

After the gems are being cancelled, those gems on top of the current gems should “fall” down.
New gems will be randomly generated to fill up the board.

<table>
<tr><td><pre>
    0   1   2   3   4
  +---+---+---+---+---+
0 | r | r | _ | _ | _ |
  +---+---+---+---+---+
1 | s | d | _ | e | d |
  +---+---+---+---+---+
2 | d | d | _ | d | t |
  +---+---+---+---+---+
3 | e | t | d | s | d |
  +---+---+---+---+---+
4 | t | e | e | s | d |
  +---+---+---+---+---+
</pre></td><td><pre>
    0   1   2   3   4
  +---+---+---+---+---+
0 | r | r | r | s | s |
  +---+---+---+---+---+
1 | s | d | t | e | d |
  +---+---+---+---+---+
2 | d | d | t | d | t |
  +---+---+---+---+---+
3 | e | t | d | s | d |
  +---+---+---+---+---+
4 | t | e | e | s | d |
  +---+---+---+---+---+
</pre></td></tr>
<tr><td>Gems on top of the cancelled gem “falls” down.</td><td>New gems are generated</td></tr>
</table>

Check if there are more connections of 3 or more gems of the same type are formed, if found repeat the above cancel and refill steps.</td></tr>
<tr><td>menu</td><td>Adjust the menu to accommodate these new changes.</td></tr>
</table>

Evidence 20

Program code of above mentioned changes. [2]